

MX

developer's journal



FOR MACROMEDIA MX
DEVELOPERS & DESIGNERS

volume 2 issue 3 www.mxdj.com

ARE YOUR BRAIN CELLS COLLIDING?

freehand 
fixes

 **DREAMWEAVER**
The Server Behavior Builder

 **FLASH**
Introducing JSAPI

 **FIREWORKS**
Picture Perfect

 **COLDFUSION**
Customized ColdFusion

 **DIRECTOR**
Best Behavior



\$5.99US \$6.99CAN 03>



0 71486 02978 6

ColdFusion hosting is our complete focus

"CFDynamics is everything I look for in an ISP. With more features and fewer restrictions than similar provider plans, consistent top-notch customer service and incredible attention to detail, CFDynamics immediately comes to mind whenever CF developers ask me to recommend a host that delivers great service at a decent price."

Simon Horwith

Macromedia Certified Instructor
Certified Advanced ColdFusion
MX Developer
Certified Flash MX Developer



SIGN UP TODAY!

Use promo code: **MXDJ04C**
and receive a **FREE T-SHIRT!**

For years we have been involved in the ColdFusion 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 web hosts. Real service. Real satisfaction. Real value.

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 has complete redundancy in power, HVAC, fire suppression, bandwidth, and security
- 24 / 7 network monitoring

FANTASTIC SUPPORT SERVICES

- Comprehensive online support
- Knowledgeable phone support
- We focus on your individual needs



macromedia
ALLIANCE PARTNER

WWW.CFDYNAMICS.COM
866 - CFDYNAMICS

866-233-9626

What's the easiest way to update a web site?

Through your browser

Introducing:

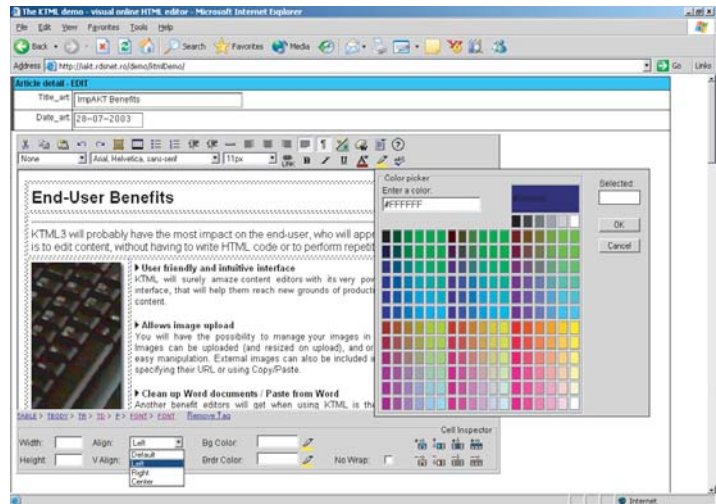
- Property inspector
- Spell-checker
- Image management
- Dreamweaver MX support
- ColdFusion, ASP, PHP

Experience:

- "MS Word" style text editing
- Text formatting using CSS
- Intuitive table alignment

Compatible with:

- MSIE/Netscape
- Windows/MAC/Linux



See all the features, benefits and a live demo at:

<http://ktml.interaktonline.com/>

KTML3 - Coming Soon to a Browser Near You





The Server Behavior Builder

A tool that offers speed and accuracy
by tom muck



Introducing JSAPI

How to harness the new extensibility layer in Flash MX 2004
by guy watson



Picture Perfect

Working with photographs to better effect
by charles e. brown

12



26



32



20

PHAKt
The InterAKT Dreamweaver MX PHP server model
by alexandru costin

on the cover



lash? FreeHand? There is a lot that both programs can do, but for real drawing power you need FreeHand; its toolset is second to none. This article explains some of the differences between FreeHand and Flash – some you'll like, some you can work around, and some you'll just have to get used to.



7

What's New with Director MX 2004
Building on a solid history
by miriam geller



10

The Year of CSS
The future has arrived
by dave mcfarland



Are Your Brain Cells Colliding?

Coping with program differences between Flash and FreeHand
by ron rockwell



Customized ColdFusion

An updated example
by sarge sargent



Best Behavior

Unleashing the real power of Director
by martin kloss



44

ColdFusion MX: A Web Services Example

Verify e-mail addresses at time-of-entry
by richard gorremans

56

Architecting with Director

Writing Xtras
by tab julius

58

xile
Cartoon
by louis f. cuffari

74

vanguard
Dangerville
by alec east



It's everybody's PDF™

Finally, a software company that offers affordable yet flexible PDF solutions to meet every customer's needs. Using activePDF™ to automate the PDF creation process eliminates the need for end-user intervention so your employees can concentrate on what they do best.

Licensed per server, activePDF solutions include a COM interface for easy integration with ColdFusion. Dynamically populate PDF forms with information from a database, convert ColdFusion web pages to PDF on the fly, dynamically print reports to PDF using CF and much more. Users can also merge, stitch, stamp, secure and linearize PDF, all at a fraction of the cost of comparable solutions. Download your free trial version today!



www.activePDF.com

Group Publisher Jeremy Geelan
Art Director Louis F. Cuffari

EDITORIAL BOARD
Dreamweaver Editor
Dave McFarland
Flash Editor
Jesse Warden
Fireworks Editor
Kleanthis Economou
FreeHand Editor
Louis F. Cuffari
Ron Rockwell
ColdFusion Editor
Robert Diamond

INTERNATIONAL ADVISORY BOARD
Jens Christian Brynildsen **Norway**,
David Hurrows **UK**, Joshua Davis **USA**,
Jon Gay **USA**, Craig Goodman **USA**,
Phillip Kerman **USA**, Danny Mavromatis **USA**,
Colin Mook **Canada**, Jesse Nieminen **USA**,
Gary Rosenzweig **USA**, John Tidwell **USA**

EDITORIAL
Executive Editors
Gail Schultz, 201 802-3043
gail@sys-con.com
Jamie Matusow, 201 802-3042
jamie@sys-con.com

Editors
Nancy Valentine, 201 802-3044
nancy@sys-con.com
Jean Cassidy, 201 802-3041
jean@sys-con.com
Jennifer Van Winckel, 201 802-3052
jennifer@sys-con.com

Technical Editors
James Newton • Sarge Sargent

To submit a proposal for an article, go to
<http://grids.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
135 Chestnut Ridge Rd.
Montvale, NJ 07645

Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ

List Rental Information
Kevin Collopy: 845 731-2684,
kevin.collopy@edithroman.com,
Frank Cipolla: 845 731-3832,
frank.cipolla@epostdirect.com

Promotional Reprints
Carrie Gebert, 201 802-3026
carrieg@sys-con.com

Copyright © 2004
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.

MX Developer's Journal (ISSN#1546-2242) is published monthly (12 times a year) by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

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. MX and MX-based marks are trademarks or registered trademarks of Macromedia, in the United States and other countries. SYS-CON Publications, Inc., is independent of Macromedia. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies.

MX
developer's journal

What's New with Director MX 2004

Building on a solid history
by miriam geller

For more than a decade, Director has set the standard for multimedia development. In fact, Director was the first multimedia authoring tool to combine animation with a scripting language so that developers could create interactive presentations, games, or computer-based training courses. It was also the first product that allowed authors to create multimedia content that could be deployed to both Mac and Windows (prior to this cross-platform innovation, developers literally had to double their work to reach both users).

With numerous innovations, Director has made an incredible impact in the multimedia industry. By the mid-1990s more than 70% of all multimedia CD-ROM titles were produced with Director. In 1996, Director fundamentally changed the way users experienced the Internet with the introduction of Shockwave Player, which brought multimedia to the Web.

Director MX 2004 continues to build on its history of innovation. As a member of the Macromedia MX 2004 family of products, Director now offers more power, a tighter level of integration with MX 2004 products, and a more streamlined and efficient workflow.

Power is about being able to incorporate whatever media helps you get your message across: audio, video, graphics of all sorts. Director supports more than 40 different media types, and in this release we've added the ability to use Windows Media and DVD-Video in Director projects.

Even if the media is not natively supported, you can still use it through Director Xtras (extensions). Hundreds of Xtras are available to take your projects in whatever direction you choose. For instance, if you want to add joysticks to a kiosk-based project or change the users' screen resolution, Xtras give you the power to do it.

With Director, power also means the ability to embed, control, and play back

DVD movies within your projects. You now have the ability to create cross-platform, enhanced DVD content. That means Director is a great companion application for people creating DVDs with tools like DVD Studio Pro and Adobe Encore. You can take the DVD projects that you've created with those applications and bring them into Director, where you can add additional interactivity and media types for an enhanced DVD-ROM viewing experience.

We know that many customers use Director and Flash together. Why? Because using them together is more powerful than using them alone. If you're a Flash user, Director will allow you to extend Flash content beyond the brows-

To learn more about Director MX 2004, visit www.macromedia.com/software/director.

director

er. You can reuse your Flash content inside Director and rapidly develop applications that leverage Director's broad media support, advanced memory management, interactive 3D, and authoring and runtime extensibility.

In Director MX 2004, developers can now import Macromedia Flash MX 2004 files and take advantage of the performance enhancements found in that release. Those enhancements – combined with performance work we've done in Director related to how we handle Flash files – mean that your Flash assets will perform better than ever.

We've also added Director-certified Flash components to the application, so that Director developers can get basic tasks done quickly.


In addition, we've updated the user interface to take advantage of MX 2004 workflow improvements, such as an integrated reference panel and start page. Plus, we've added Director-specific enhancements, such as a dockable Stage and dockable Movies in a Window. You can even customize your panel sets and create windows that are irregularly shaped, letting you create executables that are shaped like stars, donuts, triangles, or whatever shape you can imagine.

For the first time in the history of Director we've added an industry-standard scripting language to the product, in addition to Lingo. This language will bring Director to audiences who are already skilled with scripting languages and want to broaden their skills into Director.

We've added JavaScript syntax. Developers can use one or both, or a combination of the two. Web developers can easily leverage their existing knowledge to achieve immediate results in Director. Lingo programmers can also collaborate with experienced JavaScript coders and seamlessly use both scripting languages in the same project to reduce development time. We're very excited about the possibilities this new scripting language affords our users. And for those who use Lingo, we've made some updates to Lingo dot-syntax that will help them code more efficiently.

Another exciting new feature is our publishing panel. The publishing panel lets developers create cross-platform projects or Shockwave content for Mac and Windows with a single click. And it lets users save settings, so that they can publish projects much more quickly.

Finally, we've included something very techie-sounding but extremely well received from our beta testers: sprite and channel naming. This feature means it's much easier to make last-minute changes to your projects.

Whether you're a professional multimedia developer; a Flash user wanting to add different media types or deploy content to CD; an e-learning professional, game developer, or DVD professional, you should check out the latest release of Director. You won't be disappointed. 



SYS-CON MEDIA
President & CEO
 Fuat Kircaali, 201 802-3001
 fuat@sys-con.com
Vice President, Business Development
 Grisha Davida, 201 802-3004
 grisha@sys-con.com
Group Publisher
 Jeremy Geelan, 201 802-3040
 jeremy@sys-con.com

ADVERTISING
Senior Vice President, Sales & Marketing
 Carmen Gonzalez, 201 802-3021
 carmen@sys-con.com
Vice President, Sales & Marketing
 Miles Silverman, 201 802-3029
 miles@sys-con.com
Advertising Sales Director
 Robyn Forma, 201 802-3022
 robyn@sys-con.com
Director, Sales & Marketing
 Megan Mussa, 201 802-3023
 megan@sys-con.com
Advertising Sales Managers
 Alisa Catalano, 201 802-3024
 alisa@sys-con.com
 Carrie Gebert, 201 802-3026
 carrieg@sys-con.com
Associate Sales Managers
 Kristin Kuhnle, 201 802-3025
 kristin@sys-con.com
 Beth Jones, 201 802-3028
 beth@sys-con.com

PRODUCTION
Production Consultant
 Jim Morgan, 201 802-3033
 jim@sys-con.com
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 Director
 Richard Silverberg, 201 802-3036
 richards@sys-con.com
Assistant Art Director
 Tami Beatty, 201 802-3038
 tami@sys-con.com

SYS-CON.COM
Vice President, Information Systems
 Robert Diamond, 201 802-3051
 robert@sys-con.com
Web Designers
 Stephen Kilmurray, 201 802-3053
 stephen@sys-con.com
 Christopher Croce, 201 802-3054
 chris@sys-con.com
Online Editor
 Lin Goetz, 201 802-3045
 lin@sys-con.com

ACCOUNTING
Accounts Receivable
 Charlotte Lopez, 201 802-3062
 charlotte@sys-con.com
Financial Analyst
 Joan LaRose, 201 802-3081
 joan@sys-con.com
Accounts Payable
 Betty White, 201 802-3002
 betty@sys-con.com

EVENTS
President, SYS-CON Events
 Grisha Davida, 201 802-3004
 grisha@sys-con.com
Conference Manager
 Lin Goetz, 201 802-3045
 lin@sys-con.com
National Sales Manager
 Sean Raman, 201-802-3069
 raman@sys-con.com

CUSTOMER RELATIONS
Circulation Service Coordinators
 Shelia Dickerson, 201 802-3082
 shelia@sys-con.com
 Edna Earle Russell, 201 802-3081
 edna@sys-con.com
 Linda Lipton, 201 802-3012
 linda@sys-con.com
 Merline Noel
 merline@sys-con.com
JDJ Store Manager
 Brundila Staropoli, 201 802-3000
 bruni@sys-con.com

Complete source code and asset management in Dreamweaver MX—now possible with Surround SCM.

Dreamweaver users know a beautiful Web-based product is only skin deep. Underneath, it's a tangle of hundreds or thousands of ever changing source files. Without a good development process and strong tools, bad things happen. Surround SCM can help.

Surround SCM lets you...

Track multiple versions of your source files and easily compare and merge source code changes.

Check out files for exclusive use or work in private workspaces when collaborating on a team.

Automatically notify team members of changes to source files—push changes through your organization.

View complete audit trails of which files changed, why, and by whom.

Associate source code changes with feature requests, defects or change requests (requires additional purchase of TestTrack Pro).

Remotely access your source code repository from Dreamweaver MX.

Surround SCM adds flexible source code and digital asset control, powerful version control, and secure remote file access to Dreamweaver MX. Whether you are a team of one or one hundred, Surround SCM makes it easier to manage your source files, letting you focus on creating beautiful Web-based products.

Features:

Complete source code and digital asset control with private workspaces, automatic merging, role-based security and more.

IDE integration with Dreamweaver MX, JBuilder, Visual Studio, and other leading Web development tools.

Fast and secure remote access to your source files—work from anywhere.

Advanced branching and email notifications put you in complete control of your process.

External application triggers let you integrate Surround SCM into your Web site and product development processes.

Support for comprehensive issue management with TestTrack Pro—link changes to change requests, bug reports, feature requests and more.

Scalable and reliable cross-platform, client/server solution supports Windows, Linux, Solaris, and Mac OS X.

Achieve major improvements in Web and e-business development performance through better tool integration and process automation. Gain complete control over your source code and change process with Surround SCM. Manage defects, development issues, and change requests with award-winning TestTrack Pro. Completely automate product testing with QA Wizard. Streamline your development process with Seapine tools and help your team deliver quality software products on time, every time.

Learn more about
Surround SCM at
www.seapine.com
or call 1-888-683-6456



macromedia
ALLIANCE PARTNER



MXDJ Section Editors

Dreamweaver

Dave McFarland

Author of Dreamweaver MX 2004: The Missing Manual, Dave can be relied upon to bring Dreamweaver MX to life for MXDJ readers with clarity, authority, and good humor.



Flash

Jesse Warden

A multimedia engineer and Flash developer, Jesse maintains a Flash blog at www.jessewarden.com and says, referring to the MX product range, that "Things are changing, opportunity is on the frontier, a paradigm shift is occurring for Web design, Web applications, et al."



Fireworks

Kleanthis Economou

A Web developer/software engineer since 1995, now specializing in .NET Framework solutions, Kleanthis is a contributing author of various Fireworks publications and is the technical editor of the Fireworks MX Bible. As an extension developer, he contributed two extensions to the latest release of Fireworks.



FreeHand

Louis F. Cuffari

Cofounder and art director of Insomnia Creations (www.insomniacreations.com), Louis has spent most of his life as a studio artist, including mediums from charcoal portraits to oil/acrylic on canvas. In addition to studio art, he has been involved in several motion picture projects in the facility of directing, screenwriting, and art direction. Louis's creative works expand extensively into graphic design, and he has expertise in both Web and print media. He is deputy art director for SYS-CON Media and the designer of MX Developer's Journal.



Ron Rockwell

Illustrator, designer, author, and Team Macromedia member, Ron Rockwell lives and works with his wife, Yvonne, in the Pocono Mountains of Pennsylvania. Ron is MXDJ's FreeHand editor and the author of FreeHand 10 f/x & Design, and coauthor of Studio MX Bible and the Digital Photography Bible. He has Web sites at www.nidus-corp.com and www.brainstormer.org.

ColdFusion

Robert Diamond

Vice president of information systems for SYS-CON Media and editor-in-chief of ColdFusion Developer's Journal, Robert was named one of the "Top thirty magazine industry executives under the age of 30" in Folio magazine's November 2000 issue. He holds a BS degree in information management and technology from the School of Information Studies at Syracuse University. www.robertdiamond.com



The Year of CSS

The future has arrived
by dave mcfarland

I'm not one who normally tries to predict the future. I ignore stock market tips, advice from psychics, and weather reports. But I think I can safely say that 2004 will see some of the most fundamental changes in Web-site design since the birth of Netscape Navigator. This is the year of CSS.

Sure, Cascading Style Sheets isn't a new technology – the original recommendation came out in 1996. And if you've been building sites for the last few years, it's likely you've been taking advantage of the formatting control CSS offers – from typographic nuances like line-height and text-indent, to fine-tuned background, border, and margin controls. But odds are, for your most important sites, you've steered clear of CSS layout and stuck with the tried-and-true Web-layout workhorse – HTML tables.

The problem isn't CSS – the standard is robust enough to handle most design challenges. It's the browsers that we (or our bosses and clients) feel compelled to support; we fear that some vocal minority of our Web traffic still clings to Netscape Navigator 4, or Internet Explorer 3. (Please check your Weblogs! It just ain't so.)

But the dam is finally cracking. The groundbreaking work of CSS Zen Garden (www.csszengarden.com) has shown that CSS provides a level of design control that (in the right hands) can rival the best layout that print publications can offer. And it's not just personal Web zines, blogs, and "experimental" sites that are adopting CSS. Major corporations have seen the light and are following along – Wired.com, ESPN.com, FastCompany.com, and even AOL.com are using pure CSS on some if not all of their sites' pages. Yeah, that's right: AOL.

From a business perspective there's a lot to admire about CSS. In most cases, it can trim significant fat from table-heavy HTML files – meaning faster pages and lower bandwidth costs. ESPN.com, for example, estimates that they've shaved 50KB from the average page on their site, leading to a projected bandwidth savings of 2 terabytes of data per day. Amazingly, their home page (<http://msn>

espn.go.com) is still a visual feast of graphics and fine design details.

In addition, CSS-based designs provide great flexibility in site updates – a single CSS file can skin an entire site. Swap one CSS file for another and you can instantly change a site's look and feel – check out www.csszengarden.com to see this amazing feat in action.

Fundamental changes to a layout – such as moving the navigation bar from the top of the page to a left-hand sidebar – require changing only a few CSS rules, not hours of tedious reworking of HTML code.

The original benefits of Cascading Style Sheets remain as well: CSS encourages modular design and the separation of structure (HTML) from presentation (CSS); output can be customized to a wide variety of devices so content is accessible by printers, screen-readers for the visually impaired, cell phones, PDAs and other hand-held devices, and even text-based Web browsers like Lynx; and for designers, the most important benefit of CSS is that it just looks better than anything you can do with HTML alone.

CSS isn't a Macromedia technology, but everyone developing Web sites with Macromedia tools can feel its impact. Even Flash MX 2004 provides some support for CSS to provide more unified presentation between Web pages and Flash movies. And, of course, Dreamweaver users and ColdFusion developers can take immediate advantage of CSS in their workflows. In fact, the most significant additions to Dreamweaver MX 2004 relate to CSS – from better style creation and editing to greatly improved rendering of CSS designs within Dreamweaver. That's why in the next few months we'll present a variety of articles on CSS in the Dreamweaver section of *MXDJ*. We'll cover the basics of CSS, as well as troubleshooting advice, advanced tips, and tricks – information you'll need to stay ahead of the curve. Enjoy the future. ☺

Dave McFarland is the Dreamweaver editor of MX Developer's Journal and author of Dreamweaver MX 2004: The Missing Manual. davemcfarland@sys-con.com



HostMySite.com

Built for ColdFusion Pros

by ColdFusion Pros

plans from

\$8.95 / mo.

FREE Domain Name*

FREE Setup

FREE 2 Months

- 24 / 7 / 365 Phone Support
- 99.9+% Uptime
- Macromedia Alliance Partner
- "Full Control" Panel
- CFMX 6.1 or CF 5.0
- SQL Server 2000 or 7.0
- Custom Tags Welcome

Visit www.HostMySite.com/mxdj for:

2 Months Free

and **FREE Setup on Any Hosting Plan***



**"When it comes to ColdFusion hosting,
HostMySite.com rules them all!"**

James Kennedy
mbateam.com

*offer applies to any annual shared hosting plan

call
today!

877•248•HOST
(4678)



The Server Behavior Builder

by tom muck

D

reamweaver is at the top of the heap in the arena of Web-development applications. Part of the reason why it is so successful is its plethora of timesaving features that generate code for you. Probably the biggest timesaving feature of all – and the least-used tool in the Dreamweaver arsenal – is the Server Behavior Builder. I'll tell you why in a moment, but first I want to talk about your workflow.





Your Workflow in Dreamweaver

Dreamweaver attracts many types of developers and designers by its very nature. The program appeals to designers because of the real-time rendering of the HTML and server-side code in Design view. The program appeals to hand-coders because it allows you to type code in Code view without worrying about the program changing your code. The program appeals to Web application developers because it combines hand-coding, quick application development using built-in behaviors and server behaviors, and offers split Code/Design view for easy access to all aspects of your page.

The workflow for these developers will vary widely. A hand-coder may never use a behavior or server behavior, and never even open Design view. A person who works in Design view may never edit code by hand.

Different developers have different types of workflow, but the concept of the server behavior can fit into anyone's workflow. This is why: when you build your own server behavior, you are using your code. You aren't using some code written by Macromedia, or someone else's idea of how a particular function or section of code should be written. It is your code, packaged into a point-and-click interface that can be inserted into a document quickly and accurately. You can click a button faster than you can type, copy, or paste. Any code that you find yourself using more than once makes a good candidate for a server behavior.

Isn't a server behavior just like a code

snippet? In a way it is, but it is also much more than that. Server behaviors are handy because they are packaged functionality. They exist for the sole purpose of speeding up the development process. One aspect of server behaviors that makes them so useful is that they can consist of more than one code block – often in different parts of your page. For example, if you have a piece of code that generates the recordset, opens a file on the server, attaches the file to an e-mail, sends the e-mail, and then closes a recordset, you might be looking at three or four blocks of code. You can put the whole thing into a server behavior and save the hassle of inserting separate snippets or copying/pasting blocks of code into your page.

The other aspect of server behaviors that truly separates them from the code snippet or the copy/paste method is that they can be parameterized. The parameters can be typed in or you can use any of the built-in Dreamweaver controls, such as browse buttons for files; buttons for database fields; or dropdown lists of connections, database tables, or fields. The advantage here is that you can allow Dreamweaver to choose an item so that you don't risk mistyping a field name, or need to calculate relative paths to documents and images.

Now that you know why you should build your own server behaviors, let's get down to building one. Because Dreamweaver supports a wide variety of server environments, I'll show the code for each of the server environments that you can use with Dreamweaver.

Building a Server Behavior

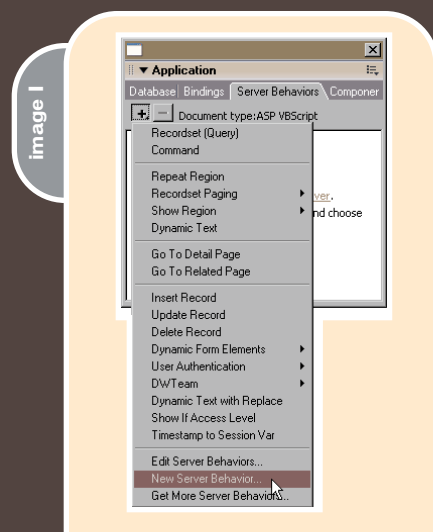
Not much skill is involved in building server behaviors. The biggest hurdle is changing your workflow to incorporate the Server Behavior Builder (SBB) into it. To build one, you need to know only two things: the code you want to insert in the page, and where you want to insert it. The Server Behavior Builder does the rest for you, with its wizard-like interface. After you've built the server behavior, it will be available to you from the Server Behaviors panel.

The SBB is located in the Server Behaviors panel. To access it, click the plus sign (+) on the panel and then click New Server Behavior (see Image I). This

will open up the builder. To get the ball rolling, we'll create a very simple server behavior that redirects a user to a different page if a session variable is not equal to a specific value. Use Code I for the server model of your choice to complete the example. For the examples I'm going to assume that you already have a site set up in the server model of your choice.

After clicking New Server Behavior, you are presented with the dialog box in Image II. Follow these simple steps to create the server behavior.

1. Pick the language your page uses from the dropdown list. The dropdown list should already be showing the server language of your page, but if it isn't, choose it. Dreamweaver supports these languages:
 - ASP/JavaScript
 - ASP/VBScript
 - ASP.NET VB
 - ASP.NET C#
 - PHP MySQL
 - ColdFusion
 - JSP
2. Give your server behavior a name. Call this one "Redirect On Session Value". The hardest part of building a server behavior is giving each server behavior a unique name that is 27 characters or less, and make it meaningful to you.
3. Click OK to move to the next dialog box, shown in Image III. Here is where you will add the code for your server behavior. So far, creating the server behavior has been a lot like creating a snippet.
4. Click the plus sign to insert a new code block. The code block will have an arbitrary name, like "Redirect On Session _block1", which you can accept by clicking OK or change it to something else. In most cases, the default name is fine, as it is a name that Dreamweaver will use internally to keep track of the code blocks. The only consideration is that the name has to be less than 26 characters if you want to package your server behavior. Unfortunately, Dreamweaver does not enforce this internally. In this case, remove all the spaces so that it reads



- "RedirectOnSession_block1". That will make the name 24 characters. Click OK to accept the name.
- Next you can paste your code into the Code Block box. The box will have some generic text in it to begin with that says "Replace this text with the code to insert when the server behavior is applied". This is exactly what you should do. Paste the code from Code I into the box.
 - In Code I notice that the code has three items that can be variables: the name of the session variable (SessionVarName), the value that you want to match (ValueToMatch), and the page that you will be redirecting the visitor to (PageToRedirect). Replace these now by highlighting each word in turn, then clicking the Insert Parameter In Code Block button. That brings up the dialog in Image IV.
 - The name of the parameter should be changed from the generic "Param1" to something meaningful. In this case, the name you use will show up in your final server behavior interface. Type "Session Variable Name" for the first parameter, "Value To Match" for the second parameter, and "Page To Redirect To" for the third parameter. Notice that Dreamweaver inserts two @ symbols around your parameter (@@Session Variable Name@@). Dreamweaver uses this pattern internally to replace your parameters.
 - Choose where you want the code inserted. There are several options here, but we'll choose "Above the HTML Tag" under the Insert Code option, and "The Top of the File" under the Relative Position option.
 - Click the Next button to move to the next section, where you will create the interface for your server behavior. You can use plain text fields for the Session Variable Name and Value To Match fields, but choose the URL Text Field for the Page To Redirect To field (see Image V). This will place a browse button next to the field so that you can choose a file in your site. Dreamweaver will calculate the rel-

ative paths.

- Click OK to complete your first server behavior. It will show up now in the Server Behaviors panel.

Choose the new server behavior "Redirect On Session Value" from the panel to see the interface you just created. It should look like Image VI. If you fill in the values and apply it to the page, the code will be inserted at the top of your page.

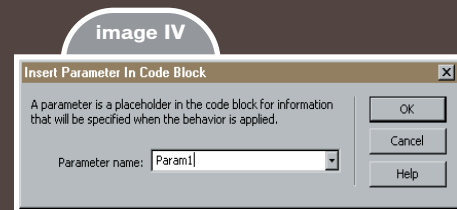
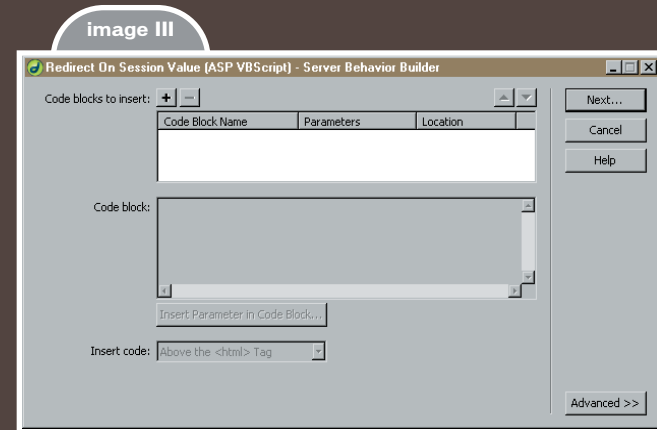
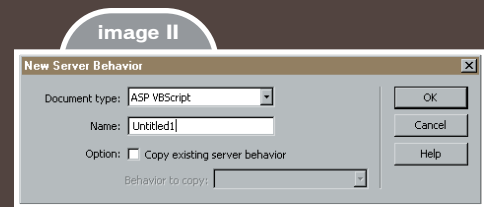
A More Complex Server Behavior

The first server behavior was really basic, to get your feet wet with the SBB. Now we'll add a slightly more complex server behavior to your collection. This code will work with a search/results page, giving you a keyword highlight on your search word. The code has to be versatile enough to recognize the case of the letters in the search word. For example, if you type "ColdFusion" or "coldfusion" you want the word to be highlighted in the search results using the same case as appears in the text rather than using the case that was typed in the box. Code II will accomplish this.

To build this server behavior we'll follow the exact same steps as before, only this time we have two code blocks to add to the server behavior. The first code block will be a function that is added to the top of the document (with no parameters) and the second block will be added at the current cursor location and will replace whatever is selected. We'll use this to display a recordset field – with the function call already in place.

Following are the steps to build this server behavior:

- Choose New Server Behavior from the Server Behaviors panel.
- Pick the language that your page uses from the dropdown list.
- Give your server behavior a name. Call this one "Highlight Search Word".
- Click OK to move to the next dialog box.
- Click the plus sign to insert a new code block. The code block will

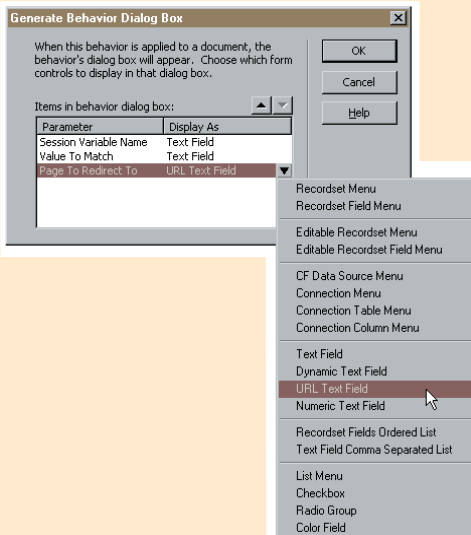


Tip: One of the first things you should learn about building server behaviors is that you want your code to be solid and completely debugged before you build it as a server behavior.

have an arbitrary name. Click OK to accept the name.

- Next, paste the code from Code II into the Code Block box. Start with Block 1. This block has no parameters.
- Click the plus sign to insert a new code block. The code block will have an arbitrary name. Click OK to accept the name. This is Block 2.
- In Code 2, Block 2, the code has three things that will be variables: the name of the recordset (rs), the recordset field (field), and the search field name (searchfield). Replace these now by highlighting each word in turn, then clicking the Insert Parameter In Code Block button.
- Type "Recordset Name" for the first parameter, "Field" for the second parameter, and "Search Field" for the third parameter.
- The next step is to choose where you want the code inserted. You will have to choose a position for each code block. Choose "Above the HTML Tag" under the Insert Code option, and "The Top of the File" under the Relative Position option for the first code block (the function). Choose "Relative to the Selection" and "Replace the Selection" for the second code block.
- Click the Next button to move to the next section, where you will create the interface for your server behavior. Using the arrows, move

image V



Recordset Name to the top and Field to the second position.

- Choose Recordset Menu for the Recordset Name parameter, Recordset Field Menu for the Field parameter, and leave the Search Field parameter set to a plain text field. Your server behavior interface will now show a drop-down list of all recordsets on the page, and after the user chooses a recordset, it will show all fields in that recordset.
- Click OK to complete the server behavior. It will show up now in the Server Behaviors panel.

image VI

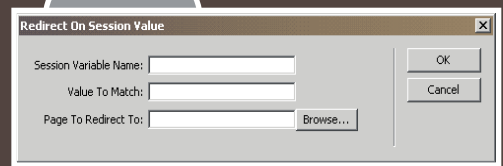
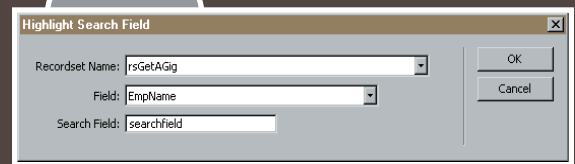


image VII



Using the Server Behavior

Now let's set up a page where you can use this new server behavior. The basic structure of a search/result page is to have a search box and button, a recordset, and a dynamic table that displays the results. To set this page up, we'll use some built-in Dreamweaver server behaviors (using a sample database of your choice). We'll build the entire page from Design view.

- Build a form on your page using a text field and a submit button. The easiest way to do that is to open the Form tab of the Insert bar and click the following three buttons in turn: Form, Text Field, Button. Give your text field the name searchfield and set the method of the form to GET.
- Add a recordset to your page that is filtered by the text field. A recordset can be added from either the Bindings panel or the Server Behaviors panel, among other places. For this simple demonstration, choose any two fields in your database for the result, and filter one of them by the text field named searchfield.
- Add a table to the page to display the results. To do this, open the Common tab of the Insert bar and click the Table icon. Give your table one row and two columns. Put your cursor in the second cell of the table.
- Open the Server Behaviors panel and find your new server behavior – Highlight Search String (see Image VII).

Choose the recordset and database field that you are searching. Fill in the name of the text field. Click OK. Your new server behavior should be displayed in the server behaviors panel.

- Finally, add a Repeat Region server behavior to the table row. The easiest way to do this is to select the <tr> tag in the tag selector at the bottom of the document in Code view, then click the Repeat Region button on the Application tab of the Insert bar.

At this stage you should have a complete working search/results page with custom text highlighting. If you open the page in Code view, you can see the code that has been inserted by each of the server behaviors, including the custom server behavior you just built.

Another great thing about server behaviors is that after you have inserted them into the page, you can call up your interface at any time in the Server Behaviors panel and edit your parameters. As you build more complex code snippets and convert them to server behaviors, you'll soon find this method easier than modifying the code by hand.

Packaging Your Server Behaviors

Server behaviors can be packaged as extensions, giving you an easy way to install your server behaviors onto another computer, or to share among your fellow programmers. Unfortunately, Macromedia did not provide an automated method of packaging server behaviors after you've built them with the SBB. Creating an extension package is easy, however, and a matter of simply writing a small XML file and using the Extension Manager to package the files.

NORM MEYROWITZ
PRESIDENT OF PRODUCTS



WEB DEVELOPERS ARE USING OUR MX PRODUCTS
IN WAYS WE NEVER DREAMED. JUST IMAGINE
WHAT THEY'LL DO WITH THE NEW MX 2004.

I've been endlessly amazed by the things our customers have done using our MX generation of products. And with all the new features in Studio MX 2004, it's going to be even easier and faster for them to realize their visions.

Dreamweaver MX 2004 helps you get that picture in your head turned into a web site faster than ever. That should be welcome news to the millions of web professionals who use Dreamweaver to create sites and applications. We've added things like CSS support, target browser check and improved code hinting to help you get through projects in far less time. And with Fireworks MX 2004 you can optimize web graphics up to 85% faster.

The new Flash MX Professional 2004 takes our industry standard tool for building rich content and applications to a whole new level. It really helps stretch your abilities—no matter where you fall on the designer-developer continuum. For development, we've added things like data-aware components and an extensibility layer so you can add your own features. For design, we've added high-quality, long-format video that's really impressive.

These are just a few of the new features. The products are available individually, but they really work well together in Studio MX 2004. Don't take my word for it. Download a free trial and read more at our web site.

Let me know what you dream up. norm_01@macromedia.com

Copyright © 2003 Macromedia, Inc. and its licensors. All rights reserved. Macromedia, the Macromedia logo, Dreamweaver, Flash and Macromedia Flash are trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries. Other marks are the properties of their respective owners. *Offer valid for English, French and German products only.



Introducing
Macromedia MX 2004.

www.macromedia.com/go/2004



The XML file is stored with an .mxi file extension, which stands for Macromedia Extension Information. The file for the Redirect On Session Value server behavior is shown in Code III. To package the extension, simply put the .mxi file into a folder with the files that were generated by the SBB and double-click the .mxi file to open the Extension Manager. The Extension Manager is like a zip program – it zips all the files into a package along with path information. Double-clicking on an extension package will install the extension into Dreamweaver.

The SBB generated three files – an HTML file that contains the server behavior interface, an EDML file that contains information about the extension, and an .edml file that contains the code block that will be inserted into the document. The three files will be located in the Dreamweaver Configuration folder. In most modern operating systems, the folder will be located in the local multi-user folder. For example, on Windows XP, the folder is located here by default: C:\Documents and Settings\[your username]\Application Data\Macromedia\Dreamweaver MX 2004\Configuration\ServerBehaviors\ASP_VBS.

Conclusion

If you followed the exercises, you have successfully become a Dreamweaver extension writer, but more important, you've added a new tool to your arsenal that should significantly speed up your workflow. Server behaviors are powerful because they insert large blocks of interrelated code into your document, with user-defined parameters presented in an easy-to-use interface. The Server Behavior Builder is a tool that can increase your workflow by speeding up repetitive tasks and making code modifications quickly and accurately using a visual interface. ∞

Tom Muck is the coauthor of nine Macromedia-related books, including O'Reilly's Flash Remoting: The Definitive Guide, and Osborne's Dreamweaver MX 2004: The Complete Reference. He is an extensibility expert focused on the integration of Macromedia products with ColdFusion, ASP, PHP, and other languages, applications, and technologies. Tom is also a founding member of Community MX (www.communitymx.com), a site that focuses on the MX product line. Tom's Dreamweaver extensions can be found at www.dwteam.com/extensions/. tom@tom-muck.com

code I

ASP/VBScript

```
<%
If Session("SessionVarName") = "ValueToMatch" Then
    Response.Redirect("PageToRedirect")
End If
%>
```

ASP/JavaScript

```
<%
if(Session("SessionVarName") == "ValueToMatch") {
    Response.Redirect("PageToRedirect");
}
%>
```

ASP.NET VB

```
<%
If Session("SessionVarName") = "ValueToMatch" Then
    Response.Redirect("PageToRedirect")
End If
%>
```

ASP.NET C#

```
<%
if(Session["SessionVarName"] == "ValueToMatch") {
    Response.Redirect("PageToRedirect");
}
%>
```

JSP

```
<%
if(session.getAttribute("SessionVarName") == "ValueToMatch"){
    response.sendRedirect("PageToRedirect");
}
%>
```

ColdFusion

```
<cfif IsDefined("Session.SessionVarName") AND
    Session.SessionVarName EQ "ValueToMatch">
    <cflocation url="PageToRedirect">
</cfif>
```

PHP MySQL

```
<?php
if(isset($_SESSION["SessionVarName"]) &&
    $_SESSION["SessionVarName"] == "ValueToMatch") {
    header("Location: PageToRedirect");
}
?>
```

code II

ASP/VBScript

Block 1

```
<%
Function highlightSearchWord(theField, theSearchWord)
    Dim before, after, myRegExp
    before = "<strong>"
    after = "</strong>"
    Set myRegExp = New RegExp
    With myRegExp
        .Pattern = "(" & theSearchWord & ")"
        .IgnoreCase = True
        .Global = True
    End With
    highlightSearchWord = myRegExp.Replace(theField, before & "$1" & after)
    Set RegularExpressionObject = nothing
End Function
%>
```

Block 2

```
<%=highlightSearchWord(rs.Fields.Item("Field").Value,
    rs__searchfield)%>
```

ASP/JavaScript

Block 1

```
<%
function highlightSearchWord(theField, theSearchWord){
    if(theSearchWord == "") return theField;
    var before = "<strong>";
    var after = "</strong>";
    theSearchWord = new RegExp("(" + theSearchWord + ")", "ig");
    theField = theField.replace(theSearchWord, before + "$1" + after);
}
```

```

return theField;
}
%>
Block 2
<%=highlightSearchWord(rs.Fields.Item("Field").Value,
rs__searchfield)%>
ASP.NET VB
Block 1
<script runat="server" language="VB">
Public Function highlightSearchWord(ByVal theField As
String,
ByVal theSearchWord As String) As String
If theSearchWord = "" Then
return theField
End If
Dim before as String = "<strong>"
Dim after as String= "</strong>"
theSearchWord = "(" & theSearchWord & ")"
theField = Regex.Replace(theField, theSearchWord,
before & "$1" & after, RegexOptions.IgnoreCase)
highlightSearchWord = theField
End Function
</script>
Block 2
<%=# highlightSearchWord(rs.FieldValue("Field", Container),
(IIf((Request.QueryString("searchfield") <> Nothing),
Request.QueryString("searchfield"), ""))) %>

```

ASP.NET C#

```

Block 1
<script runat="server">
public string highlightSearchWord(string theField, string
theSearchWord){
if(theSearchWord == "") return theField;
string before = "<strong>";
string after = "</strong>";
theSearchWord = "(" + theSearchWord + ")";
theField = Regex.Replace(theField, theSearchWord,
before + "$1" + after, RegexOptions.IgnoreCase);
return theField;
}
</script>

```

```

Block 2
<%=# highlightSearchWord(rs.FieldValue("Field", Container),
(((Request.QueryString["searchfield"] != null) &&
(Request.QueryString["searchfield"].Length > 0)) ?
Request.QueryString["searchfield"] : ""))%>

```

JSP

```

Block 1
<%!
public String highlightSearchWord(String theField, String
theSearchWord){
if(theSearchWord == "") return theField;
java.util.regex.Pattern pat =
java.util.regex.Pattern.compile(theSearchWord =
"(?im)(" + theSearchWord + ")");
java.util.regex.Matcher matcher = pat.matcher(theField);
String before = "<strong>";
String after = "</strong>";
theField = matcher.replaceAll(before + "$1" + after);
return theField;
};
%>

```

```

Block 2
<%= highlightSearchWord(rs.getString("field"), rs__search-
field)%>

```

ColdFusion

```

Block 1
<cffunction name="highlightSearchWord">
<cfargument name="theField" type="string" default="">
<cfargument name="theSearchWord" type="string"
default="">

```

```

<cfif theField EQ "">
<cfreturn theField>
</cfif>
<cfset before = "<strong>">
<cfset after = "</strong>">
<cfset theSearchWord = "(#theSearchWord#)">
<cfset theField = REReplaceNoCase(theField,
theSearchWord,
"#before#\1#after#", "all")>
<cfreturn theField>
</cffunction>
Block 2
#highlightSearchWord(rs.Field, URL.searchfield)#

```

PHP MySQL

```

Block 1
<?php
function highlightSearchWord($theField, $theSearchWord){
if($theSearchWord == "") return $theField;
$before = "<strong>";
$after = "</strong>";
$theSearchWord = "(" + $theSearchWord + ")";
$theField = preg_replace($theSearchWord, $before + "$1"
+ $after, $theField);
return $theField;
}
?>
Block 2
<?php echo highlightSearchWord($row_rs['Field'],
$searchfield_rs); ?>

```

```

<!-- Redirect On Session Value Copyright 2004 by Thomas
Muck -->

```

```

<macromedia-extension
name="Redirect on Session Value
version="1.0.0"
type="ServerBehavior"
requires-restart="true">

```

```

<products>
<product name="Dreamweaver" version="6" />
</products>

```

```

<author name="Thomas Muck"/>

```

```

<description>
<![CDATA[
Redirect a user to another page depending on the value
of a session variable.
]]>
</description>

```

```

<ui-access>
<![CDATA[
Access this extension by choosing:<br>
Server Behaviors >> Redirect On Session Value
]]>
</ui-access>

```

```

<!-- Describe the files that comprise the extension -->
<files>
<file name="Redirect On Session Value.htm"
destination="$dreamweaver/Configuration/ServerBehaviors/ASP_
Vbs/"> </file>
<file name="Redirect On Session Value.edml"
destination="$dreamweaver/Configuration/ServerBehaviors/ASP_
Vbs/"> </file>
<file name="RedirectOnSession_block1.edml"
destination="$dreamweaver/Configuration/ServerBehaviors/ASP_
Vbs/"> </file>
</files>
</macromedia-extension>

```

d

f

fw

fb

fd

db

PHAkt

Originally, I was going to use this article to present a tutorial on using PHAkt in conjunction with PostgreSQL to create dynamic Web sites. However, as I've just returned from MAX 2003 (a nice show, really), my vision changed a bit. I was repeatedly told, "Dreamweaver users don't work with server behaviors – they prefer to code manually."

I decided to shift the focus of this article to include a clearer view of server behaviors as well as the PHAkt product overview.

Introduction

Dreamweaver Ultradev 4 was a great tool for building dynamic Web sites, but did not include support for Apache and PHP, a popular duo.

That's how PHAkt – the PHP Server Model for Dreamweaver – was born. It was the most complex effort of its kind, and it was released as an open source product (free for download) to a huge market success.

PHAkt History

Let me give you a hint: PHAkt comes from PHP and InterAKT. The naming convention has remained since then, even if many people reach us by searching "phpakt".

PHAkt was very popular in the Ultradev 4 era – people downloaded it more than 200,000 times, and we received raves across the board from the Dreamweaver community. But starting with the MX version, Dreamweaver began supporting PHP in the PHP_MySQL server model instead.

"Why would one still need PHAkt?" you might wonder. The answer is simple: PHP_MySQL (as its name implies) can connect only to the MySQL database server. While a large number of Web developers will live with this, some will opt not to use Dreamweaver for PHP development because of its inability to use other databases. Despite its extreme speed, MySQL is still a "toy" database, with no support for triggers, stored procedures, or referential integrity.

Even if you use MySQL, PHAkt can still help you create powerful applications. With a PHAkt site, you will be able to switch to another database later without regenerating all your pages – something impossible with PHP_MySQL.

Web Development with Dreamweaver

Most of today's Web sites are dynamic – the content is stored in the database and rendered on the front end in the visitor browser.

Multiple scripting languages may be used to create dynamic Web sites; you might be familiar with some of them: ASP (Microsoft Active Server Pages), CFM (Macromedia ColdFusion), and JSP (JavaServer Pages). One of the newcomers in this area is PHP (PHP: Hypertext Preprocessor).

To create such Web sites, developers need to create pages that connect to a database and then write the actual HTML output to the client browser (see Image I).

To write script pages, you can use a text editor or you can choose to use a professional tool to leverage your work. Of course, the leader of this latter market is Dreamweaver MX, the best Web IDE.

Despite popular belief, Dreamweaver MX is not only a designer tool, but also a dynamic Web programmer tool. Starting with the fact that 90% of dynamic Web development consists of creating lists with database information and creating forms to update the database, Dreamweaver provides a suite of "server behaviors" to automate those tasks. This is one of the most important features in Dreamweaver MX, and is often not used to its full potential (read "at all").

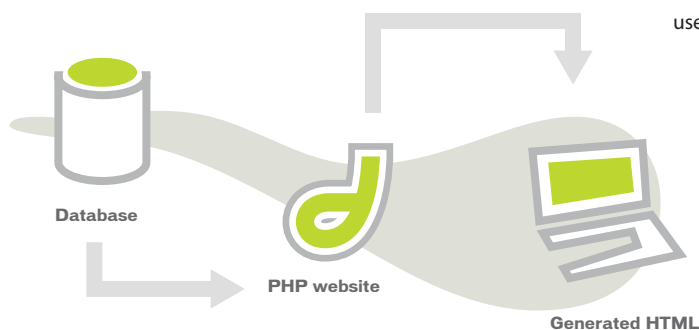
Server behaviors are reusable code blocks you can use in your Web sites. Aside from the code, they come with a useful GUI that allows you to set various parameters to the code block.

Let's look at one of the most used server behaviors (SBs), the recordset. Its role is to connect to a database, perform a SQL query, and return a recordset (a data structure that will store all the rows and fields returned from the database).

As you can see in Image II, we can set the name and the database connection to be used. Notice that most of the SB interface parameters are bounded properties, meaning that they are dynamically retrieved from the database server without the developer having to memorize or type them.

After you select a database (the database dropdown is loaded with the available database connections), the table menu is updated with all the tables in the current database, and so on.

Applying this SB on the page will out-



put the following code:

```
<?php
//Connection statement
require_once('Connections/phakt.php');

// begin Recordset
$query_Recordset1 = "SELECT * FROM
name_nam";
$Recordset1 = $phakt-
>SelectLimit($query_Recordset1) or
die($phakt->ErrorMsg());
$totalRows_Recordset1 = $Recordset1-
>RecordCount();
// end Recordset?>

<snip>

<?php
$Recordset1->Close();
?>
```

What makes SBs useful is that they not only generate the code for you, but they also recognize it on the page and allow you to update one code block by loading the configuration interface with all the parameters correctly set. This allows you to change them and reapply

the SB. If we go behind the scenes, there is a very complex regular expression engine that stands behind this mechanism, and there are also some limitations in recognizing the already-applied code blocks, but you can easily adapt to them.

Even if some programmers choose not to use SBs ("they restrict my options," "I don't like the generated code," etc.), you should understand that used wisely, they can really improve how you create Web sites.

Picture two situations where you want to create a dynamic Web site:

1. You are a programming guru. You know how to code, you understand algorithms, SQL, and LEFT JOINS. What you do when creating Web sites (and this means that you list and update information from the database) is a lot of redundant pointing to tables and fields – a very boring task. I don't think anyone in the world enjoys it.
2. Conversely, you are a Web designer, skilled in HTML and CSS, who wants to make a section of your site dynamic. You don't know any scripting language, SQL, or other complicated programmer notions.

How would SBs help both the programmer and the designer? By writing the code for them. With SBs, the guru would be able to have the boring sections of code automatically generated, saving time for the really interesting stuff, and the designer would have correct script code generated, saving time to focus on the aesthetics of the site.

Enough about SBs. I encourage you to look closely at what they can do for you, as you will be amazed. If you want to find more, Dreamweaver has tons of documentation on them and you will find everything you want in there.

Configuring Your Site for PHAKt

To use PHAKt, you have to install the MXP extension using the Extension Manager. Just go to our PHAKt site and download the product; you will find the phakt-2.6.x.mxp in the PHAKt2 folder. Clicking it should do the install.

You will need access to a computer with Apache, PHP, and a database installed (the database will probably be MySQL, but any database will do).

You will find a lot of information

Introducing the new FuseTalk.

Collaboration
Discussion Forums

Flexible Security, 508 Compliance, Offline Capabilities, Reporting, Easy Integration and much more...

1-866-477-7542

www.fusetalk.com

FuseTalk™

Discussion forum solutions that make web-based collaboration risk-free and easy.

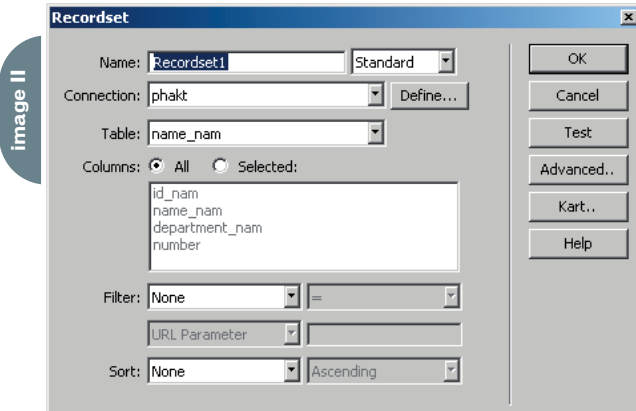


image II

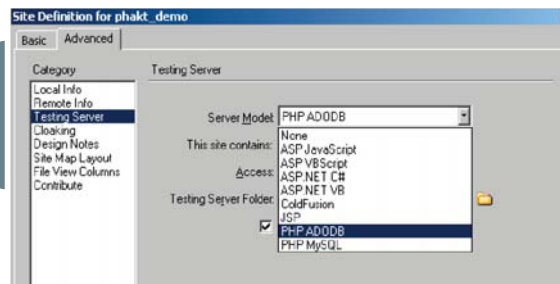


image III

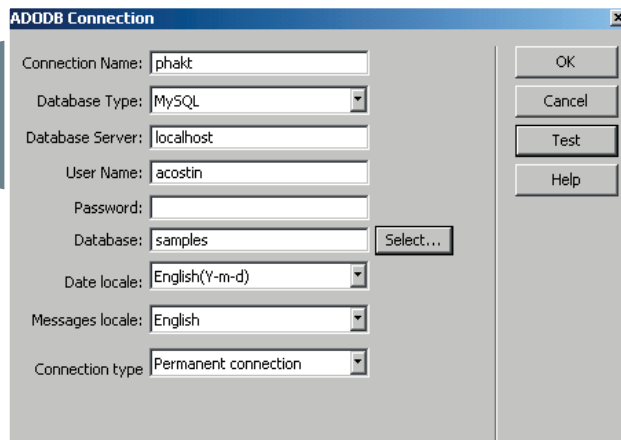


image IV

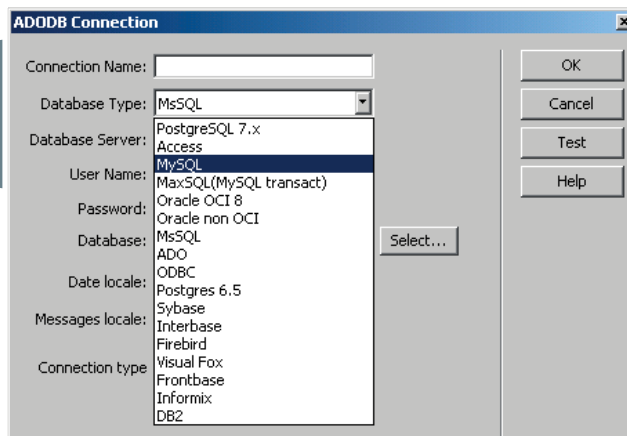


image V

about installing PHP and preparing a workstation for serving dynamic pages at www.dreamweavermxsupport.com.

If you want to test PHAKt at work with another database server, you can get PostgreSQL and install it. PostgreSQL is one of the most advanced open source database servers available, and can compete seriously with any major player in the industry.

Multiple ports for Windows are available; we've found UltraSQL to be the easiest to install (<http://techdocs.postgresql.org/guides/Windows>).

Once your Apache/PHP server is set up, configure Dreamweaver to edit PHP files using PHAKt. The first step in starting to use PHAKt is to set up the Dreamweaver site (I assume you are already familiar with this, so I won't elaborate)

As shown in Image III, instead of PHP_MySQL, you have to select PHP_ADODB for the Testing Server of your configured site. PHP_ADODB is the server-model name for PHAKt: a confusing naming convention we set up a long time ago, but we have to live with it to keep the backwards compatibility.

When you create new .php files in this type of Dreamweaver site, they will be "stamped" with a small PHP comment in their upper section:

```
<?php //PHP ADODB document - made with
PHAKt 2.6.2?>
```

This is our way of recognizing a page as a PHAKt page, so you should leave this comment as is. It will not affect your site performance, nor will it be included in your site HTML output.

Improved Database Connection

Let me say this one more time: the most powerful PHAKt feature is its capability to transparently connect to multiple databases from your dynamic Web site. To allow this, we rely on a PHP objects library – ADOdb. Please don't mistake this for the Microsoft ADODB, as they are completely different and share only the API function names.

Our ADOdb (actually it's not ours, it's an open source library written by John Lim – <http://php.weblogs.com> – but we have improved it a little bit) simply consists of some PHP files that will dynamically choose the correct native database connection library from PHP and use it. When installing a PHAKt site, you will not have to install anything on your PHP server; just copy the site's files and

Dreamweaver Extension of the Month

by dave mcfarland

ADOdb will be automatically copied, too.

Let's illustrate the difference between ADOdb and pure PHP code by comparing the native PHP_MySQL code with PHAKt code for a simple SQL query.

The PHP_MySQL code looks like:

```
mysql_select_db($database_mysql_demo,
$mysql_demo);
$query_Recordset1 = "SELECT * FROM
name_nam";
$Recordset1 =
mysql_query($query_Recordset1,
$mysql_demo) or die(mysql_error());
```

You can see the mysql_* API calls, and it's pretty clear that this code is database specific.

The PHAKt code looks much cleaner, as the SelectLimit() method called is not database specific:

```
$query_Recordset1 = "SELECT * FROM
name_nam";
$Recordset1 = $p-
>SelectLimit($query_Recordset1) or
die($p->ErrorMsg());
$totalRows_Recordset1 = $Recordset1-
>RecordCount();
```

Of course, ADOdb will call the mysql_query function, but it uses encapsulation to hide this implementation detail for you.

Creating the Database Connection

In order to benefit from PHAKt's capabilities in a site, you must create a connection from the Databases Panel.

When you click on the + sign, Image IV appears. As you can see, the configuration screen is much more complex than the regular PHP_MySQL database configuration interface. This initial complexity is in fact the window to many useful and available features.

Apart from selecting the database type, you can define a format for the DATE fields in the database and a location for the application messages to be used later in the application. You can also decide whether or not the database connection is persistent (persistent connections use a pool of connections to the database layer to avoid the reconnection time penalty, and thus are slightly faster but more resource intensive).

Links for Simplicity

Like the air we breathe, links are such an omnipresent part of Web development that we often don't give them much thought. This month, I'll present three free extensions that are simple but useful tools for expanding Dreamweaver's link-creating abilities.

Tom Muck's Quick Link extension has been around for years, but is still one of my favorite extensions. It's simple, efficient, and free; it also greatly simplifies the process of converting plain text into a link. Suppose you've received a text file from a client; you copy and paste the text into Dreamweaver and realize that the document is riddled with URLs – www.sys-con.com, www.yahoo.com, and so on. Unfortunately, they're simply plain text and you've got to change them into real links. The most basic method, selecting the URL, copying it, and pasting it into the Property Inspector's Link box – is a pain; and if the protocol – http:// – is missing, there's the extra typo-prone step of adding that as well.

Quick Link makes the process fast and easy. Just select the text and choose Quick Link from the Commands menu. The text is wrapped in an anchor tag, with the correct href attribute attached. If you left out the http://, don't worry; the extension is smart enough to figure that out and add it for you. It also works with e-mail addresses – select an address, run the command, and the text is wrapped in a link complete with mailto: added to the address. You can find this helpful tool at www.dwteam.com/articles/quicklink/index.asp.

Speaking of e-mail addresses – are the addresses listed on your site safe from spammers? Many spammers use spiders to crawl Web sites and collect e-mail addresses for use in their network-clogging e-mail campaigns. Web masters go to great lengths to hide their e-mail addresses from such tools – I've even seen e-mail links replaced with graphics containing e-mail addresses. Fortunately, a handy

extension has come to the aid of Web developers who are sick of letting robots pillage their pages for e-mail addresses.

Hide Email, from Linecraft, is a good solution. This free extension (available at www.linecraft.com/downloads.htm) breaks up and scrambles e-mail addresses so robots won't detect them. When a user clicks the e-mail link, a JavaScript program recombines the e-mail address into a functioning mailto link.

Dreamweaver's Open Browser Window behavior is a tried-and-true classic; great for popping open windows with announcements, Flash movies, and (unfortunately) ads. But if you've ever wanted a bit more control over the pop-up window – for example, placing it at a precise location on the screen – you've had to add the JavaScript code yourself. Until now.

FlevOOWare's Popup Link extension offers all the options available in the Open Browser Window behavior – width, height, chrome controls – plus the ability to position the window in the left, center, right, top, or bottom of the screen. In addition, you can offset the window a set number of pixels from the screen's left, right, top, or bottom edges. You can find it at www.flevooware.nl/dreamweaver/extdetails.asp?extID=8.

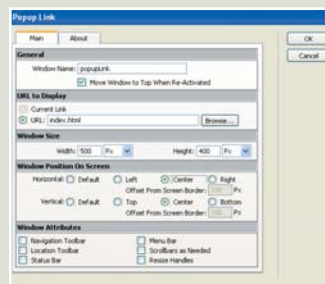
Dave McFarland is the Dreamweaver editor of MX Developer's Journal and author of Dreamweaver MX 2004: The Missing Manual. davemcfarland@sys-con.com

Quick Link
Extension Developer:
Tom Muck
Developer Web Site:
www.dwteam.com/articles/quicklink/index.asp

Hide Email
Extension Developer:
Linecraft
Developer Web Site:
www.linecraft.com/downloads.htm

Popup Link
Extension Developer:
FlevOOWare
Developer Web Site:
www.flevooware.nl/dreamweaver/extdetails.asp?extID=8

Price: Free



Do you have a favorite extension? Is there an extension you just can't live without? We're always on the lookout for awesome extensions, so drop me a line at davemcfarland@sys-con.com.



Supported Databases

PHAKt supports a large list of databases. Unfortunately, we weren't able to test PHAKt effectively on all databases, but the open source community around the ADOdb project did, so you should have a very healthy starting point.

We have used PHAKt mostly with MySQL, PostgreSQL, MsSQL, and Access. We've heard various success stories from our clients who use Oracle, Informix, and Firebird (see Image V).

Once you create the connection, you can start using PHAKt to create Web sites. As we have followed the Macromedia GUI's "unwritten" design rules (believe me, they are *very* strict when it comes to uploading on the Exchange), our interfaces look very similar to the PHP_MySQL ones for all of the server behaviors provided.

Other Improvements

I'll continue the discussion of PHAKt by describing some of the other improvements we've made to help programmers create PHP dynamic Web sites visually.

Date Locales

InterAKT is located in Europe. Because we also do contract work, we have clients around the globe – France, Germany, Australia, the U.S., to name a few – and in creating Web applications for them we've had real problems supporting multiple date formats.

French clients want to edit dates like "m-d-Y", Americans want "Y-m-d", and Germans want "d.m.Y". Because they used MySQL (where you don't have any real way of setting the date locales on the server) on hosted environments (where we can't access the server to change the server locale or anything), we were forced to think of a solution that would allow us to read and write dates in a specific format in the database.

Server Formats from Dreamweaver is a solution when displaying dates, but it can't handle updates at all.

Our solution was to change the ADOdb wrapper and the PHAKt underlying libraries to allow us to set the date locale for a specific connection, and then use this locale transparently when displaying dates and when inserting/updating records in a table. Thus, it's very straightforward for any developer to create internationalized applications (from the date's point of view, at least).

The date-locale support was thoroughly tested with MySQL, PostgreSQL, and MsSQL, but we will continue its implementation based on a client request.

We have also created PHAKt as a complete server model, providing some features Dreamweaver 6.1 didn't have (MX 2004 seems to have corrected this issue).

Supplemental Server Behaviors

When working with PHAKt, you will be able to use:

- User authentication server behaviors
- Master/Detail server behaviors
- Go to Detail Page server behaviors

PHAKt has several other advantages as well.

- **Code reuse:** We keep the code that can be reused in a functions.inc.php file that is available when needed.
- **Apache2 header redirect support:** In Apache 2 on Windows, relative redirects don't work anymore.
- **Advanced Recordset:** This allows you to define other data sources and still reuse the current server behaviors.

This advanced Recordset is probably one of the most powerful features in PHAKt, since we overcome a major Dreamweaver limitation. It normally allows you to use a specific server behavior for only one data source. If you want to define a new data source, for a shopping cart, for example, you will have to reimplement all the features like "Repeat Region", "Show If", etc.

Extending PHAKt

Dreamweaver MX is a platform built with extensibility in mind, extensibility that allowed InterAKT to create PHAKt. Using the same approach, developers around the world can create new features that will plug into Dreamweaver to enable it to support more functionality.

Only a few companies create extensions for PHAKt (they usually focus on PHP_MySQL), but you will find all you need in the list below (shopping cart extensions, query builders, form generators, horizontal loopers and nested repeat regions, etc.):

- **InterAKT MX Kollektion:** www.interakt.ro/online.com
- **Felixone Extensions:** www.felixone.it/extensions/dwextensions.asp
- **Advanced Query Wizard:** www.advancedextensions.com/

Note: Most of the PHAKt extensions provided by these companies are commercial.

PHP in the Enterprise

Even if PHP has recently been designed from scratch, it has rapidly evolved into a fully featured, easy-to-use programming language for productive, dynamic Web site development.

To get some real information on the PHP market, we conducted extensive

Satisfied with your current Hosting provider?
The grass really is greener on the *ServerSide*.



ColdFusion Hosting is our specialty. Find out why ColdFusion Developers nationwide choose ServerSide for high quality, high availability web hosting and support.

The grass is greener at www.serverside.net



Mention code #MXD04 : and we'll waive the set-up fee



(888) 682.2544
hosting@serverside.net
www.serverside.net

Advertising Index

market research, and the results may be helpful if you are really serious about PHP. Our research involved 700 respondents, and we managed to get a clearer view on the software development market for PHP and its relationship to medium and large enterprises. Read the survey to see our estimate of the number of PHP developers, the average price of a PHP Web site, and some golden rules to promote PHP in an enterprise setting. Our full market research analysis is located at www.interaktonline.com/index_article_11.html.

Conclusion

Let's take one last look at a summary of PHAKt's features.

Upside

The most important PHAKt feature is its ability to connect to multiple database types. This allows you to create advanced PHP sites, but if you use MySQL, you'll be able to change the database after developing the site without having to re-implement it.

Overall, PHAKt is a step further along in creating PHP sites with Dreamweaver MX, as it integrates a set of improvements that ease software development.

Downside

Unfortunately, there is also a downside to PHAKt, and the most important is that there are only a few extensions available for PHAKt.

Also, because of the ADOdb usage, PHAKt is slightly slower than native connections, but if you want to create something extensible you will find ways to correct this using various techniques (PHP accelerators, indexes in the database, etc.).

PHAKt's Future

We will continue the work on PHAKt to provide the professional alternative for PHP Web site development in Dreamweaver.

We plan to merge PHAKt with the InterAKT Transaction Engine (tNG), a much better way of creating forms with associated events (send mail, upload image, etc.). We want to include PHP 5 support and enlarge the included server behaviors and commands set.

Advertiser	URL	Phone	Page
ActivePDF	www.activePDF.com	(866) GOTO PDF	6
CFDynamics	www.cfdynamics.com	866-233-962-6427	Cover II
CFXHosting	www.cfxhosting.com	866-CFX-HOST	43
CTIA	www.ctiashow.com	(202) 736-3895	Cover III
EdgeWebHosting	www.edgewebhosting.net	1(866)EDGEWEB	47
FuseTalk	www.fusetalk.com	866-477-7542	21
HostMySite.com	www.hostmysite.com/mxdj	877-248-4678	11
Interakt	http://ktml.interaktonline.com		3
Macromedia	www.macromedia.com/go/2004	800-470-7211	17
Macromedia	www.macromedia.com/into	800-470-7211	Cover IV
Paper Thin	www.paperthin.com	800-940-3087	25
Seapine Software	www.seapine.com	888-683-6456	9
ServerSide	www.serverside.net	888-682-2544	24
TechSmith	www.techsmith.com	800 517-3001	57

Next Steps

As all regular server model features are present in PHAKt, you can rely on standard Dreamweaver MX Help to get used to the server behaviors and commands.

You will find server behaviors to create lists with records from a Table – Repeat Region, to apply conditional regions – Show if Recordset is Empty, and to create insert, update, and delete forms for a table.

PHAKt also comes with several tutorials and documentation, so you should be able to get started quickly.

I hope you have a better view of what Dreamweaver MX, server behaviors, and PHAKt can do for you. Just go and download our extension and start creating dynamic Web sites now! ☺

Resources

- PHAKt can be downloaded from the InterAKT Web site: www.interaktonline.com/products/PHAKt
- Various PHAKt discussions: www.interakt.ro/products/bbs/index_0.html
- ADOdb database abstraction layer: <http://php.weblogs.com/ADODB/>

Alexandru Costin is one of the founders and chief operating officer at InterAKT Online. As one of the InterAKT product architects, he has contributed to PHAKt's initial development and now designs the next generation of InterAKT products. He co-authored Professional PHP Web Services and posts regularly to the Dreamweaver MX-related boards. acostin@interakt.ro

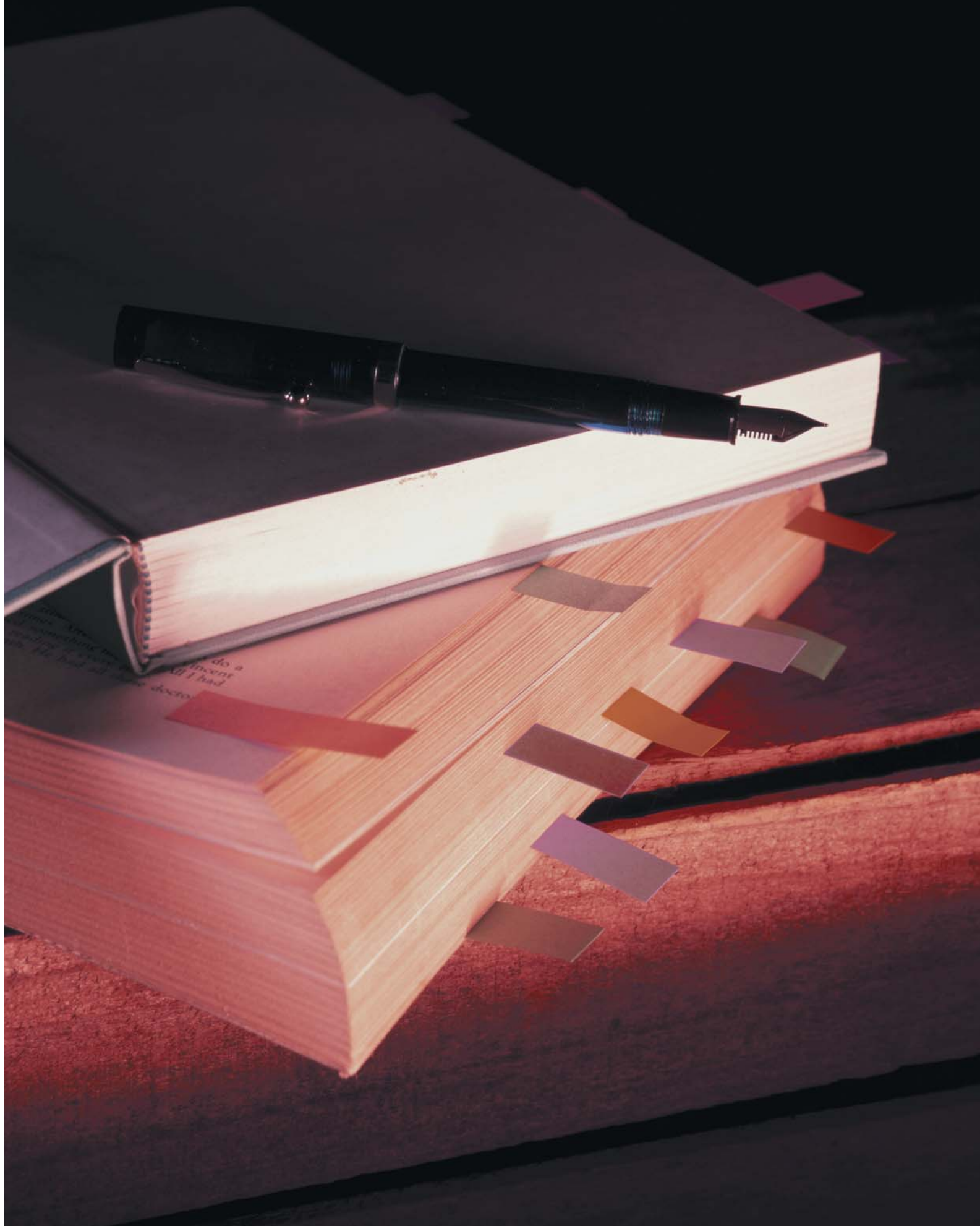
introducing

Flash

part I

with a new version of a well known product, Adobe new show
of features introduced - Macromedia Flash MX 2004 is no
different. It has been built for the ground up with a new
integrated ActionScript API (Flash JavaScript) which enables
developers to extend the functionality
of the thing to this top pat
article explores features of this new
architecture. Part 1 of the series with a
general introduction to the new
Layer, the fundamental Document Object
Model, and the relationship between different
parts of a Flash document and the associated
objects. Part 2 will cover building your own Flash
panels and will include a depth look at MX 2.0 UI -
Flash Dialog Boxes by Guy Watson

W





The 'Extensibility Layer'

With any new version of a software product, a whole new host of features is introduced – Macromedia Flash MX 2004 is no different. The most significant new feature in the latest release of Flash is the "Extensibility Layer."

The "Extensibility Layer," as Macromedia calls it, is a general term that covers a range of new, exciting features that make it possible for Flash developers to create and implement their own new features directly into the Flash Authoring Environment. Various third parties are already commercially distributing their own Flash Extensions that add new features to the IDE. For example, the makers of SWiSH, the ever-popular text effect tool, are now selling a Flash extension

called SWiSHpowerFX that allows Flash designers to select a text field on the stage and apply various SWiSH text effects to that text.

The Extensibility Layer makes it possible to write macros that will automate common tasks, write tools that manipulate objects on the stage, create panels that contain graphical user interfaces, write Timeline effects that animate objects on the stage and much, much more...

Introducing JSFL

Taking advantage of these new possibilities requires knowledge of a new



scripting language that lets us talk to the Flash MX 2004 IDE and tell it what to do. This new language is commonly called JSFL.

Those of you who have seen snippets of JSFL floating around may have noticed a striking similarity between that and ActionScript or JavaScript. Well spotted! The good news is that Macromedia based JSFL on the Netscape JavaScript API, which means that ActionScript coders like me, or anybody who has dabbled in

JavaScript, won't have to learn a whole new programming language. The syntax is exactly the same, dot syntax, and we still work with the same data types: objects, arrays, strings, numbers, and functions.

As this article is written for Flash developers, it assumes knowledge of ActionScript.

JSFL is based around a

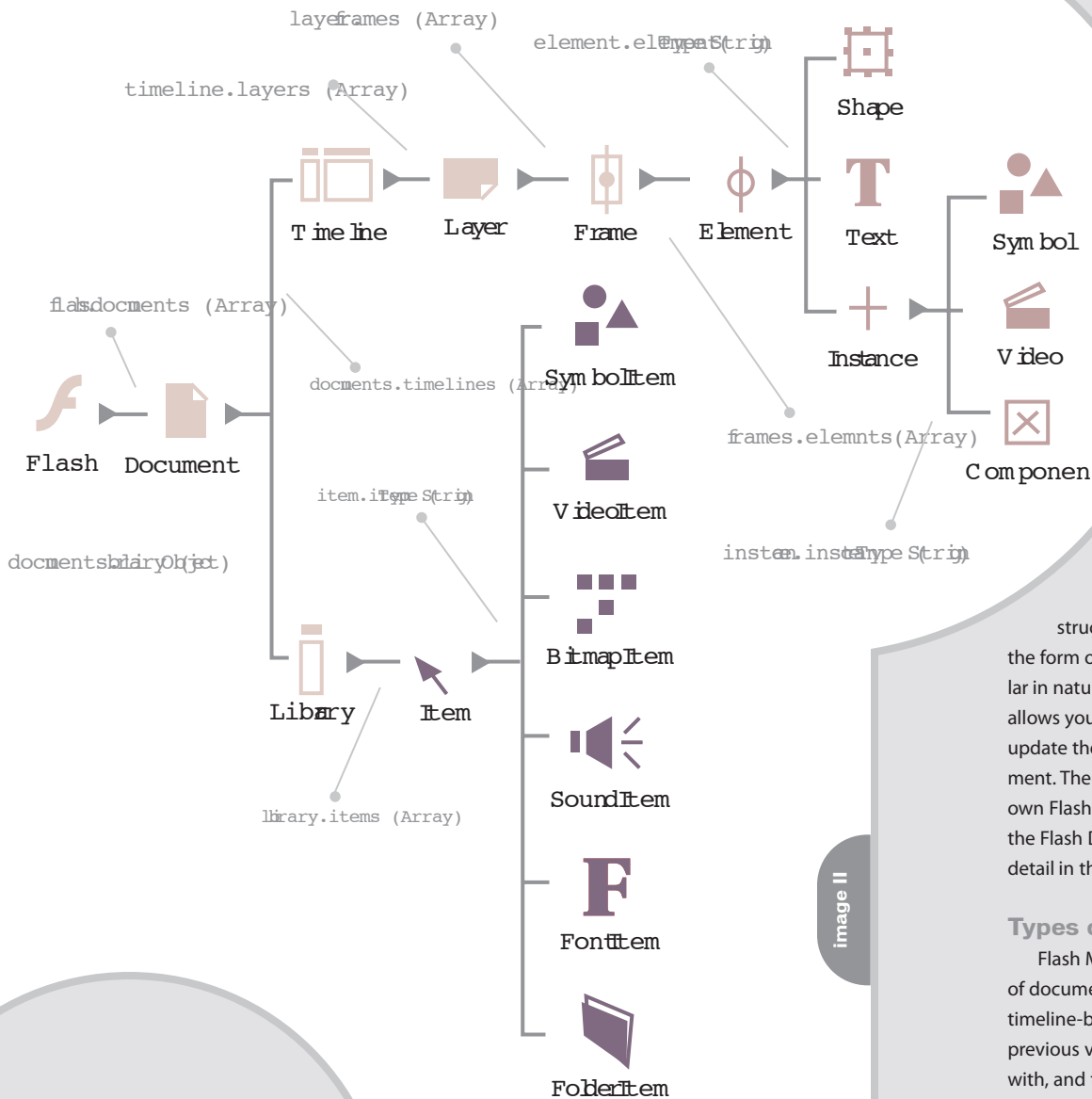
Document Object Model (DOM) that

exposes a hierarchy of objects that represents the

structure of a Flash document in the form of a hierarchical tree, very similar in nature to a family tree.

Each object allows you to dynamically access and update the structure of a Flash document.

The key to learning to write your own Flash extensions is to understand the Flash DOM, which I cover in great detail in this article.



into Flash MX 2004. Screen-based documents use a different metaphor for organizing your movies, and you will know them as Form Applications and Slide Presentations.

The DOM of a screen-based document differs slightly from that of a timeline-based document, and because timeline-based documents are the most common document I cover only their DOM in this article.

Anatomy of a Flash Document

To begin with, think about each of the different parts of a Flash document. Flash documents comprise:

- Timelines
- Layers
- Frames
- Symbols
- Shapes
- Text fields
- Library

Different types of symbols are stored in the library:

- MovieClips
- Buttons
- Graphics
- Bitmaps
- Sounds
- Videos
- Components

Timeline-based documents have a library and one or more timelines (scenes). MovieClips, buttons, components, and graphics have their own timelines. Timelines contain layers; layers contain frames; and keyframes contain instances of symbols from the library, shapes, and text fields.

Note: It's important to realize that Flash works with groups of frames (not individual frames); a group of frames begins with one keyframe and ends at the frame before the next keyframe on that layer, or the last frame in that layer, whichever is first. Each frame in a group of frames refers to the first frame in the group (the originating keyframe).

The Flash DOM represents this hierarchy and each separate element of a Flash document as objects. So there is a unique object for each layer in a Flash document; there is also a unique object for every frame in a Flash document, and so on.

These objects have properties that describe that particular element; for example, all layer objects have a "name" property that contains the name of that particular layer.

Image I contains a timeline that contains one layer; this layer is named "Layer Name". In JSFL, we can access the name of that particular layer using

```
theLayerObject.name;
```

Each type of object may also have associated methods that manipulate that particular object in its visual form in the Flash IDE; for example, all timeline objects have a method called "deleteLayer" that removes a particular layer from that particular timeline.

In JSFL, we can get Flash to delete the first layer of a timeline using:

```
theTimelineObject.deleteLayer(0);
```

Object Relationships

These objects have a parent/child relationship; each layer object contains references to its child frames, which are grouped together in an array. It is therefore said that a frame object is a child of a layer object. The parent of a layer object is the timeline object, which represents the timeline in which that particular layer resides. These relationships are similar in practice to a nested MovieClip hierarchy in a Flash movie.

When a JSFL script is executed, an object is created for each separate element in every Flash document that is presently open in the Flash IDE. Thus your scripts can refer to any element of any of the open Flash documents using dot syntax to traverse that particular document's DOM.

Image II shows the parent-child relationship between the different elements of a Flash document and the hierarchical tree structure.

Think of each section in Image II as a separate class or type of object. Child objects that are the same color as their parent are simply subclasses of their parent, or extensions of the parent object type. For example, Shape is a subclass or extension of the Element object; similarly, FontItem is a subclass or extension of the Item object.

In Image II labels point to the inter-

sections between various objects. If an intersection represents a subclass or object relationship (the color of the parent and children is the same), then the label will contain the property you need to access to determine the type of object you are working with.

If, however, the parent-child relationship represents an object type, which can be accessed as a child of the parent, then the label will tell you the property you need to access to reference each child element. For example, the relationship between the Library object and the Item object is not a subclass relationship since the colors of parent and child are different. Thus, the label tells us that we can access the child items of the Library object using the library.items property, which is an array in which is a collection of Item objects.

Here's a simple example: let's say I have a Flash document that contains one MovieClip on the stage of the first frame of the first layer of the first scene. To access that MovieClip's object, I would have to traverse the DOM using dot syntax as follows:

```
theMovieClipObject=flash.documents[0].timelines[0].layers[0].frames[0].elements[0]
```

In this particular example, we know that the one and only element on the stage is a MovieClip, and at times there will be numerous elements on the stage. As illustrated in Image II, an element can be of type Shape, type Text, or type Symbol; and the properties available to those objects differ, thus we need to determine what type of element we are working with. To do this we use the elementType property:

```
theElement=flash.documents[0].timelines[0].layers[0].frames[0].elements[0]
elementType=theElement.elementType
```

The elementType property can have three possible values: "shape", "text", or "symbol." If the element is a MovieClip, graphic, or button, then the value of the elementType property will be "symbol", and thus we now know that we are working with a Symbol object. This means that our Element object will contain all the properties that that Symbol object con-



tains.

Top of the Tree

A parent-child relationship has to stop somewhere; therefore, there is always one root object, the “top of the tree” or the “daddy of daddies,” so to speak. In the Flash DOM, the root object is the object that represents the Flash application.

Note: The Flash application has its own object, and each Flash document is a child of that object. This object is always accessible in JSFL scripts and goes by the name “flash”. It can also be accessed using the shorter name “fl”.

The root object contains the child objects that represent each Flash document presently open in the Flash IDE; these document objects are grouped together in a property called “documents” that is an array:

```
flash.documents
```

To access the object of the first document open in the Flash IDE with a JSFL script, I would use:

```
firstDocumentObject=flash.documents[0];
```

The “flash” object also has various methods that allow you to manipulate the actual Flash application and mimic the actual functionality the Flash application provides; for example, in the Flash IDE it’s possible to close the Flash application by simply opening the File menu and then selecting the Quit option. We can do the same thing with JSFL in our scripts by calling:

```
flash.quit();
```

In the Flash IDE, it is also possible to close the currently focused document simply by opening the File menu and selecting the Close option. Again, we can do the same thing with JSFL in our scripts by calling:

```
flash.closeDocument(theDocumentObject);
```

For a full list of the various objects and their associated methods and properties, check out Macromedia’s official JSAPI Documentation at <http://live-docs.macromedia.com/flash/mx2004/jsapi/index.htm>.

Predefined Classes

As with ActionScript, JSFL has a set of predefined classes containing methods and properties:

- Array
- Boolean
- Date
- Function
- Math
- Number
- Object
- RegExp
- String

ActionScript coders will notice one more class, the RegExp class. No, that isn’t a mistake; you can use Regular Expressions in JSFL!

In JSFL, as in ActionScript, every string is an instance of the String class, every array is an instance of the Array class, every number is an instance of the Number class, and so on. This means that

strings, numbers, arrays, Booleans, objects, etc., are treated in exactly the same way as they are in ActionScript.

All Strings have the usual String methods: “substr”, “charAt”, “indexOf”, etc. All Arrays have the usual Array methods: “splice”, “pop”, “push”, etc.

For example, we can do:

```
myStr="A,B,C,D";
bits=myStr.split(","); //use a string method
```

Or:

```
myStr="abcd";
myStrLen=myStr.length; //get a string property
```

In both JSFL and ActionScript the above code snippets produce the same result. For a complete list of classes, available methods, and properties refer to the Netscape JavaScript API Web site (<http://devedge.netscape.com/central/javascript/>).

Top-Level Properties and Methods

JSFL also contains some top-level functions; these are functions that are not related to any particular object or class and thus can be used anywhere in your scripts.

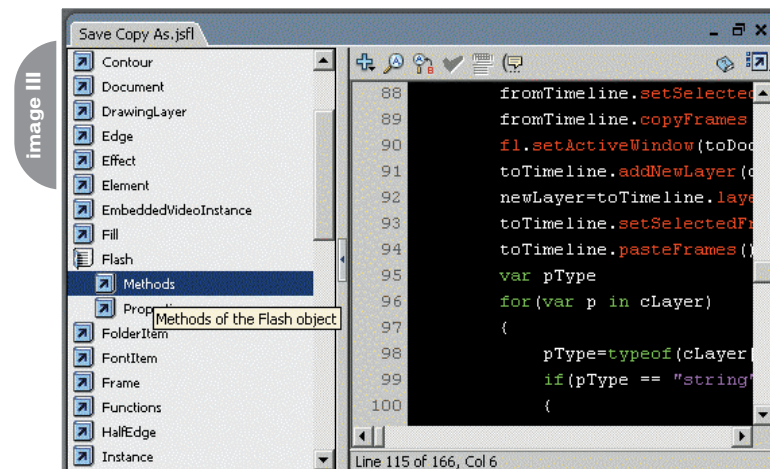
- encodeURI
- decodeURI
- eval
- Infinity
- isNaN
- Number
- parseFloat
- parseInt
- alert

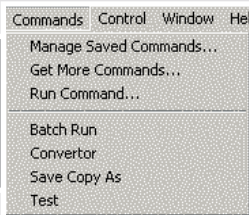
For information on the functionality of these functions, see the Netscape JavaScript API Web site.

Writing a JSFL Script

You create a JSFL script like you would an external ActionScript file. Using your favorite script editor, write your JSFL code, then save the file with a .jsfl extension. You can call the file whatever you like as long as it has the correct file extension.

The Flash MX 2004 Authoring Tool has a built-in JSFL Script Editor, which is the same as the ActionScript Editor. You





can use it to create a new JSFL script by opening the File menu and then selecting New > Flash Javascript File.

You can also open JSFL scripts for editing in Flash MX 2004. Naturally, your JSFL script will automatically open for editing in the JSFL Script Editor. To edit a JSFL script in Flash MX 2004, open the File menu and then select Open. From there you want to navigate through your local machine to find the correct JSFL script to edit.




As with the ActionScript Editor, the JSFL Script Editor also contains a list of every available object and its available methods and properties along the left-hand side. This comes in very handy as there are a lot of them to remember. Roll over a particular method or property on the left-hand side and you will get a tool tip that briefly describes what that particular method or property does (see Image III). The JSFL script editor also has syntax highlighting and code hinting.

Executing a JSFL Script

To execute a JSFL script, the Flash MX 2004 IDE must be open. The simplest form of a Flash extension, the command has its own special place in the Flash MX 2004 IDE – the Commands menu. In the Commands menu is a Run Command option. When you choose this option, you can then locate a JSFL script to execute. The Commands menu displays three default options, but it also adds a new option for each JSFL script contained within a special Commands directory in the Flash MX 2004 Configuration folder. The location of the special Commands directory on your local machine differs with different operating systems.

If you place your JSFL scripts in this directory, you can execute them directly from the Commands menu by selecting the appropriate Commands name from the list of options. The name that is displayed in the Commands menu is simply the name of your JSFL Script file (see Image IV).

As shown in Image IV, I have four JSFL

	Special Commands Directory	Flash MX 2004 Directory
 Windows XP Windows 2000	C:\Documents and Settings\ <user>\local 2004\<language>\configuration\commands\="" <="" data\macromedia\flash="" mx="" settings\application="" td=""> <td> C:\Documents and Settings\<user>\local 2004\<language>\configuration\windowswf\="" <="" data\macromedia\flash="" mx="" settings\application="" td=""> </user>\local></td></user>\local>	C:\Documents and Settings\ <user>\local 2004\<language>\configuration\windowswf\="" <="" data\macromedia\flash="" mx="" settings\application="" td=""> </user>\local>
 Windows 98	C:\Windows\Application Data\Macromedia\Flash MX 2004\ <language>\configuration\commands\ <="" td=""> <td> C:\Windows\Application Data\Macromedia\Flash MX 2004\<language>\configuration\windowswf\ <="" td=""> </language>\configuration\windowswf\></td></language>\configuration\commands\>	C:\Windows\Application Data\Macromedia\Flash MX 2004\ <language>\configuration\windowswf\ <="" td=""> </language>\configuration\windowswf\>
 Macintosh OS X	HardDrive/Users/<username>/Library/ApplicationSupport/Macromedia/Flash MX 2004/<language>/Configuration/Commands/	HardDrive/Users/<username>/Library/Application Support/Macromedia/Flash 2004/<language>/Configuration/WindowSWF/

Scripts, located in my Configuration /Commands directory, with the file names:

1. Batch Run.jsfl
2. Convertor.jsfl
3. Save Copy As.jsfl
4. Test.jsfl

Selecting any of these options in the Commands menu will execute the JSFL script.

When a JSFL script is executed, all functions that are called update the state of a particular element in a Flash document immediately, which differs from ActionScript execution.

For example, in ActionScript, if we were to move a MovieClip across the stage in a for loop, we would see only the final state of the loop as the stage is not updated until the ActionScript has been executed. However, if we were to call the “deleteLayer” function of a Timeline object in JSFL inside a for loop, the layer is immediately deleted and the Authoring Environment will refresh to display the new state of the timeline, before the code after the for loop is executed.

The History Panel

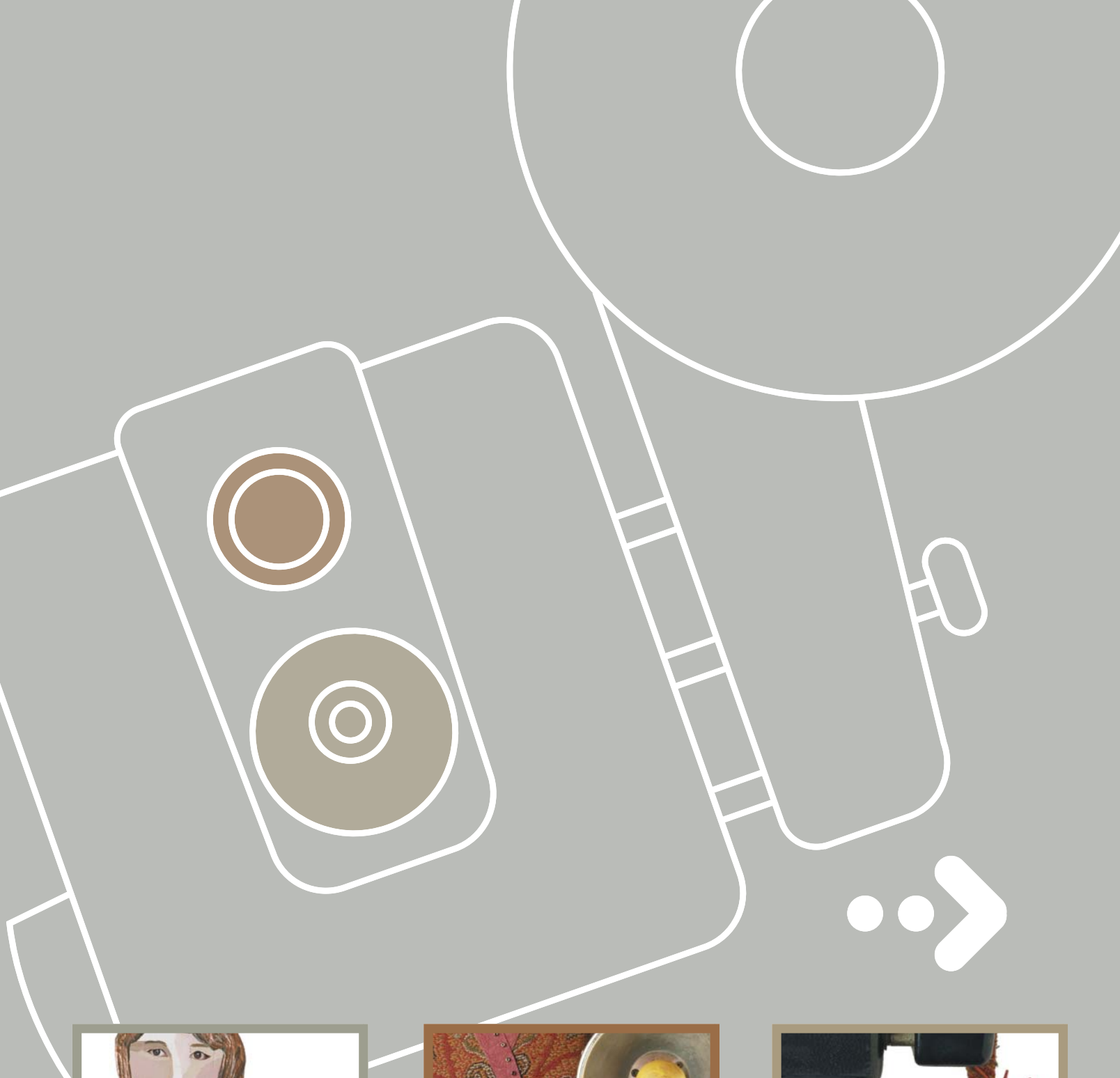
The Flash MX 2004 IDE contains a History panel: Window>Other

Panel>History. The History panel tracks every interaction you make with the Flash MX 2004 IDE. Each interaction is listed in the History panel as a separate action in the order in which the actions took place. At any time, you can select one or more of the actions you previously performed and replay them. For example, if you draw a new rectangle on the stage, a new action is listed in the History panel (see Image V).

If you then deleted that rectangle from the stage, you could draw that rectangle in exactly the same place, at exactly the same size, by simply selecting the Rectangle action from the History panel and then clicking the Replay button, or by right-clicking the action and selecting the Replay Steps option from the dropdown list that appears (see Image VI).

When you undo a change you have made in Flash MX 2004, using File>Undo (CTRL + Z), that particular action is then removed from the History panel. The opposite is true when you then choose to redo that action because you didn't mean to undo it in the first place: File>Redo (CTRL + Y). The action is then added to the bottom of the actions list in the History panel.

–continued on page 60



PICTURE PERFECT

by charles e. brown

I often receive
photographs from
readers asking me
what went wrong. The
picture files are too
large or of poor quality.





This month, we're going to look at some ways of improving your quality both within, and outside of, Fireworks MX. We'll see how to get the picture into Fireworks MX and, once in, how to make it look better.

Getting the Picture into Fireworks MX

Getting the picture into Fireworks MX properly is the first step of the process. This may seem pretty obvious at first. However, there are a lot of little tricks that even seasoned professionals frequently overlook.

The first thing you need to determine is whether the source of the photograph is going to be a scanner or digital camera. The price of both these items has dropped considerably over the past couple of years. A good quality scanner is around \$100 (you can even get them bundled with a printer and a copying machine in one), and a digital camera costs around \$300.

A word about digital cameras is in order. The quality of a camera is measured in megapixels. You can get a 2 megapixel camera for around \$300, or spend upwards of \$1,000 for a 4 megapixel camera. In most Web situations, 2 megapixels is more than sufficient. I personally own a 2 megapixel and, at the Web setting, it provides very high quality.

Most cameras and scanners have a Web setting that will determine the size and then give you the smallest dpi (dots per inch) without significant loss of quality. For instance, a Web output might be around 72 dpi; but a high-quality printing job might be around 300 dpi.

Recently, a friend called me about all of his scanned pictures being blurry in spots. After some examination, we determined that the problem was fingerprints all over the glass of the scanner. If you do a lot of scanning, use a good lens cleaner to keep your glass free of smudges, dust, and fingerprints.

Assuming your scanner is installed properly, with all the proper drivers, you should make sure that the source that needs scanning is placed properly in the scanner. Once that is done, from within Fireworks MX, select File > Scan > TWAIN Acquire. (Note: if you are working with

more than one scanner, you may first need to select File > Scan > TWAIN Select.) The image should load right into Fireworks MX.

Most digital cameras use the USB port and are recognized as another hard drive letter. All you usually need to do is use Windows Explorer (or the operating system of your choice) to drag and drop the photographs from the camera to the folder you need to put them in on your local hard drive. From there on, you can simply open them up in Fireworks.

Once the Image Is In

Once the image is inside Fireworks MX, you can do all sorts of things to improve its quality. The photograph shown in Image I is a bit scratchy and poorly shaded.

By simply selecting Filters > Adjust Color > Auto Levels, you can clean up a lot of the problems so that the Image I photograph now looks like Image II.

Understanding the Histogram

With a picture open, you can select Filters > Adjust Colors > Levels. This is called a histogram and an example of one is shown in Image III.

The histogram shows three characteristics: Shadow, Midtone, and Highlight colors. There are three sliders underneath the histogram, one for each of the characteristics.

If a picture has too much shadow, details will be hidden; too much highlight will give the picture a washed out look; and too many midtones will make the picture look dull.

There are three ways in which you can change the characteristics: you can use the sliders (if you are a beginner, this might be the easiest technique); you can use one of the three corresponding eyedroppers; or you can type the amount in one of the three Input Level fields.

You will notice that this particular photograph looks very unbalanced. We can use the histogram to do some repairs. As an example, I can select the first eyedropper (for shadow) and click it in the hair region of the photograph. The results will be something like Image IV.

image I

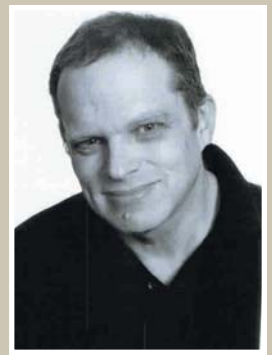


image II



image III

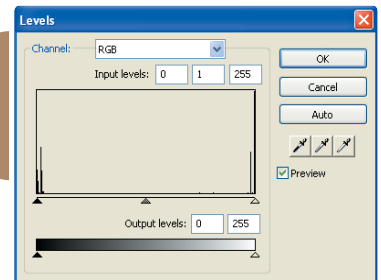


image IV



image V



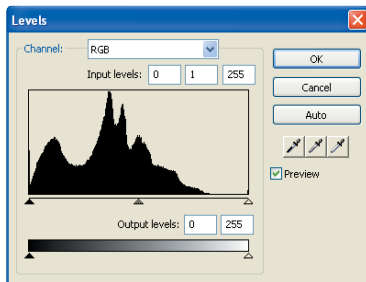


Image VI

Add a Drop of Color

You can do some interesting highlighting to a photograph by using the Brush tool and a low opacity percentage. For instance, in Image VI I used the Brush tool and set the color to soft gray. I used the Soft Rounded head with a diameter of 2 and I set the opacity to about 20%. I touched up the hair and a little of the shirt area as shown in Image V.

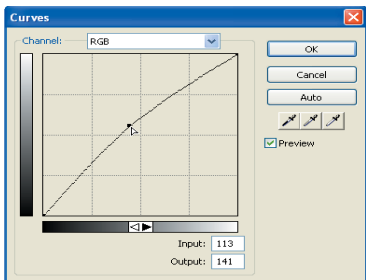


Image VII

This will take a little experimentation, and a good eye, on your part. However, you can see that there are many possibilities. I strongly suggest that you look at my discussion of blending modes in the February issue of this journal (*MXDJ*, Vol. 2, issue 2). Many of the ideas I show there can be easily applied to this discussion.

If you open a color photograph you will see an even more pronounced histogram (see Image VI).

The photograph that this histogram is referencing has a lot of peaks around the midtone area.

Another possibility for adjusting color

is to change the curve. You can do that using Filters > Adjust Color > Curves (see Image VII).

The Curves feature is similar to the Levels feature but it provides more precise control. Levels uses highlights, shadows, and midtones to correct the tonal range, while you can use Curves to correct for a color cast caused by improper lighting.

We simply click on the curve and drag it to find that precise balance.

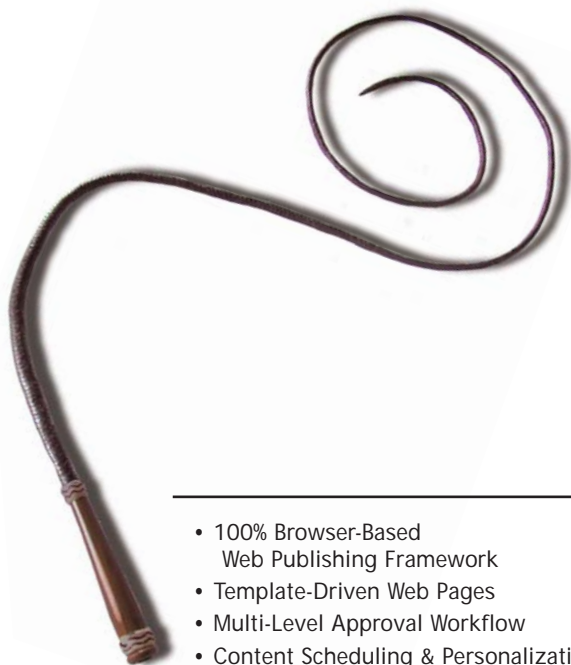
The final idea you might want to try is to take an image, such as the black and white portrait used in Images I and II, and create a background for the canvas. Experiment with color, opacity, and blend modes.

Next month, we'll discuss filters and how to create some cool effects with them. ☺

Charles E. Brown is the author of Fireworks MX: From Zero to Hero and Beginning Dreamweaver MX. He also contributed to The Macromedia Studio MX Bible. charles@charlesbrown.net

This will help sharpen your photograph considerably. However, you may notice that there now seem to be too many midtones, which are creating a blotchy effect. You can use the midtone slider in the histogram to cut that down a bit.

Content can be unruly. Make it behave with CommonSpot™



CommonSpot™ Content Server.
Put content management in the hands of content owners.

To tame your unruly web site, whip your content into shape with CommonSpot. With a rich, out-of-the-box feature set, you'll be up and running in weeks – not months. Built in ColdFusion, it's easy to integrate and customize CommonSpot to meet your requirements. CommonSpot's intuitive, browser-based interface makes authoring a snap, while its template and data-driven architecture and granular permissions allow for flexible yet powerful control.

Find out why CommonSpot was voted Best Web Content Management Tool in the CFDJ Reader's Choice Awards. Visit www.paperthin.com, or call today to schedule an online demonstration.

With CommonSpot, you'll have content tamed in no time.

- 100% Browser-Based Web Publishing Framework
- Template-Driven Web Pages
- Multi-Level Approval Workflow
- Content Scheduling & Personalization

- Rich Custom Metadata Support
- 508 Accessibility Compliance
- Seamless ColdFusion Integration
- More than 50 powerful, out-of-the box features

Paper Thin

800.940.3087
www.paperthin.com

ARE YOUR BRAIN CELLS COLLIDING

?

Whenever I tried to win an argument with my mother, she'd usually end up saying the final words: "Like it or get used to it." That's the way it is with the multitude of programs graphic artists use today. Incorporating multiple programs into a single, seamless suite is an extremely difficult concept. When programs are developed by different companies, and those companies are merged into conglomerates, technologies collide and so do our brain cells as we try to cope with program similarities and differences.

written by ron rockwell







Macromedia Studio MX is certainly no different. Having used FreeHand since the late '80s, it's only natural for me to think that newer Macromedia programs like Flash and Fireworks would utilize the same tools and tool functions that FreeHand introduced and developed. But since other teams with their own priorities developed the newer programs (sometimes from other companies),

we've ended up with quite a few inconsistencies. Due to user expectations for each program, Macromedia reworked tools to create as much uniformity as possible. In most instances this works, but at other times it falls considerably short of uniform.

In the absence of one program that does it all, we users must adapt. Flash does many things that could be done in

FreeHand, but Flash is an animation program. If you want real drawing power, you'll have to turn to FreeHand. FreeHand was built from the ground up as a drawing program, and its toolset is second to none (one if you're a die-hard Adobe Illustrator user). The accuracy and repeatability inherent in FreeHand surpass anything Flash can do, but there are several differences that you'll have to "like or get used to." ↓

STROKE

PROBLEM: Stroke ends become rounded when imported or pasted into Flash.

SOLUTION: Get used to it. There is a simple work-around however if you know that you won't be modifying the shape of the path in Flash. Use the Trace tool in FreeHand to trace a path or dashed line. Give the tracing a stroke of none and a solid fill to retain sharp-cornered dashes. You can copy and paste the traced path into Flash or export it as a Macromedia Flash SWF file. Another solution is to use the Expand Stroke Xtra, opting for square end caps; give the path a stroke of none and whatever fill color you want.



SUBSELECT

PROBLEM: The Pointer tool changes curves in Flash, but only moves paths in FreeHand.

SOLUTION: You should learn to like this. Use the Pointer tool or Subselect tool to adjust point control handles. Change the type of point from Corner to Curve to Connector in the Object panel. To adjust path sections with control handles while using the Pen or Bezigon tool, hold down the Command/Control key to temporarily switch to the Pointer tool; add the Option/Alt key to reach the Subselect tool.

You can also use the Pointer tool to push or pull a line segment by holding down the Option key as you drag the path segment. If you see a plus sign (+), you are duplicating the path – be careful.

CURVES

PROBLEM: Curves created in FreeHand are modified when imported or pasted into Flash.

SOLUTION: Due to the way each program creates paths (see the Bézier Curve sidebar), you will have to tweak the path in FreeHand and check the import in Flash until it meets your expectations.



FREEFORM

PROBLEM: The Pointer tool is used to pull or push path sections in Flash, but it isn't in FreeHand.

SOLUTION: Get used to it. If you want to smooch, squeeze, or scooch sections of a path in FreeHand, use the Freeform tool. This tool will push, pull, or reshape a path, changing, adding, or deleting points as you work.

In Push/Pull mode you can push or pull the path. Double-click the tool in the toolbox (it may be hidden behind the Roughen or Bend tools – click and hold on either of them to be able to select it from the pop-up menu). As shown in Image I, with the Push/Pull option you can set the size of the tool in pixels from 1 to 1000. The Precision setting determines how many points may be added to the path – larger numbers will add more points and therefore higher precision. The Pull setting allows you to pick Between Points or By Length. If you choose Between Points, the entire path segment will be modified. A By Length setting will create a modification in the length you set in pixels in the Length window. All of the settings in this dialog box can be entered manually or by moving the sliders. Last, if you have a pressure-sensitive drawing tablet you can choose to have either or both size and length determined by the pressure you apply to the pen. You must click the OK button to close the window and use the tool. It doesn't take long to get the hang of using the tool. If you click on the path and start to drag (pull), you'll notice the little "s" next to the cursor as the path changes according to your settings and the amount you drag. To push the path, click next to the path. A circle is added to the cursor as you push a curve into the path.

In Reshape mode the settings are about the same, with the addition of a Strength value. This is indicated as a percentage and indicates how strong the distortion will be. The cursor changes to something you've never seen in FreeHand – three concentric circles (see Image II). The center circle is where the cursor tip is,



image IV

the middle circle indicates the strength of the distortion, and the outer circle marks where a point or points will be placed on the path. Additional points will be placed as required to make the new curve.

I have a Wacom Intuos2 on my lap whenever I'm working in a drawing or photo manipulation program, and I can't recommend it highly enough. It's important to note that if you are using a graphic tablet, the path distortion changes in direct relation to the settings you've opted for in the dialog box. You can start out with a wide round shape and, by releasing pressure on the tablet, reduce the distortion to a skinny dart. Without a tablet, pressing the keyboard arrows reduces or enlarges the size of all three Freeform tools on the fly (see Images III and IV).

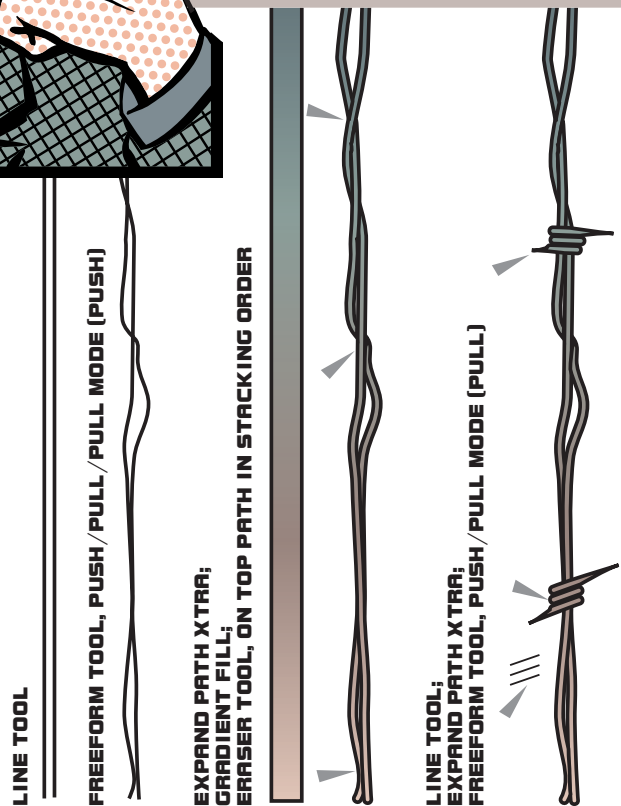


image II

image I

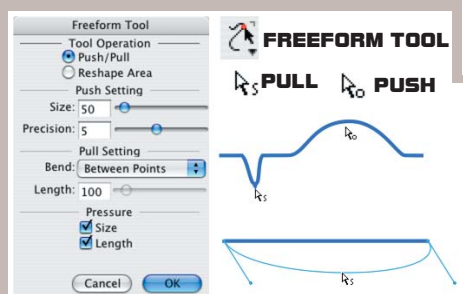
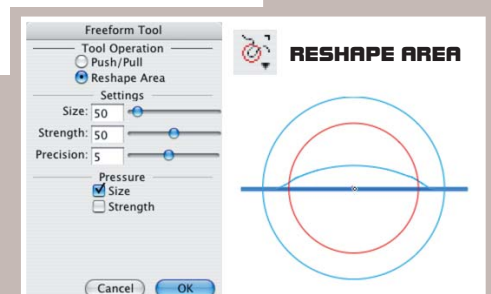


image III





PATHS

PROBLEM: The Pointer tool can be used to remove sections of a path in Flash, but selects entire paths in FreeHand.



SOLUTION: Like it. Compared to FreeHand, Flash uses a fairly haphazard method of selecting the sections. With the tools in FreeHand (see Image V), you can be extremely precise.

Knife Tool

Double-click the Knife tool to bring up its dialog box. The settings are pretty straightforward. The Freehand option lets you cut through objects as if you were drawing with the Pencil tool. Hold down the Option/Alt key to draw a straight line; use the Shift key to constrain the slice to 45° increments. The Straight option draws straight lines without the Option/Alt key. The width can be set from 0 to 72 points (1 inch). The path will be cut in two places separated by the distance you put in this field. When you make the knife cut, the path stays selected and you see the new points. Deselect the path by pressing the Tab key, or Command/Control, clicking the cursor away from the path. Now you can select the section that you cut and delete it or use it in some other way. The last two options in the Knife dialog box are Close Cut Paths and Tight Fit. The first adds two points, one on top of the other in the path, and allows a closed path with a fill to

KNIFE TOOL
SPLIT TOOL
ERASER TOOL

Image V

remain intact. Deselecting Close Cut Paths will turn the path into an open path, and unless you have Show Fill for New Open Paths set in your FreeHand Preferences, the fill will disappear. With that option in play, the fill will show as it did originally. The Tight Fit option adds precision to the course your cursor takes while you're cutting with the knife. To use the Knife tool, simply drag it across a selected path.

Split Command

The Split command is extremely powerful because it can be accessed in so many ways. The concept is to select a point or several points on a single path or many paths. Then do one of the following:

- Choose Modify>Split from the menu
- Use a custom keyboard shortcut
- Click the Split icon that you've placed in your main toolbar

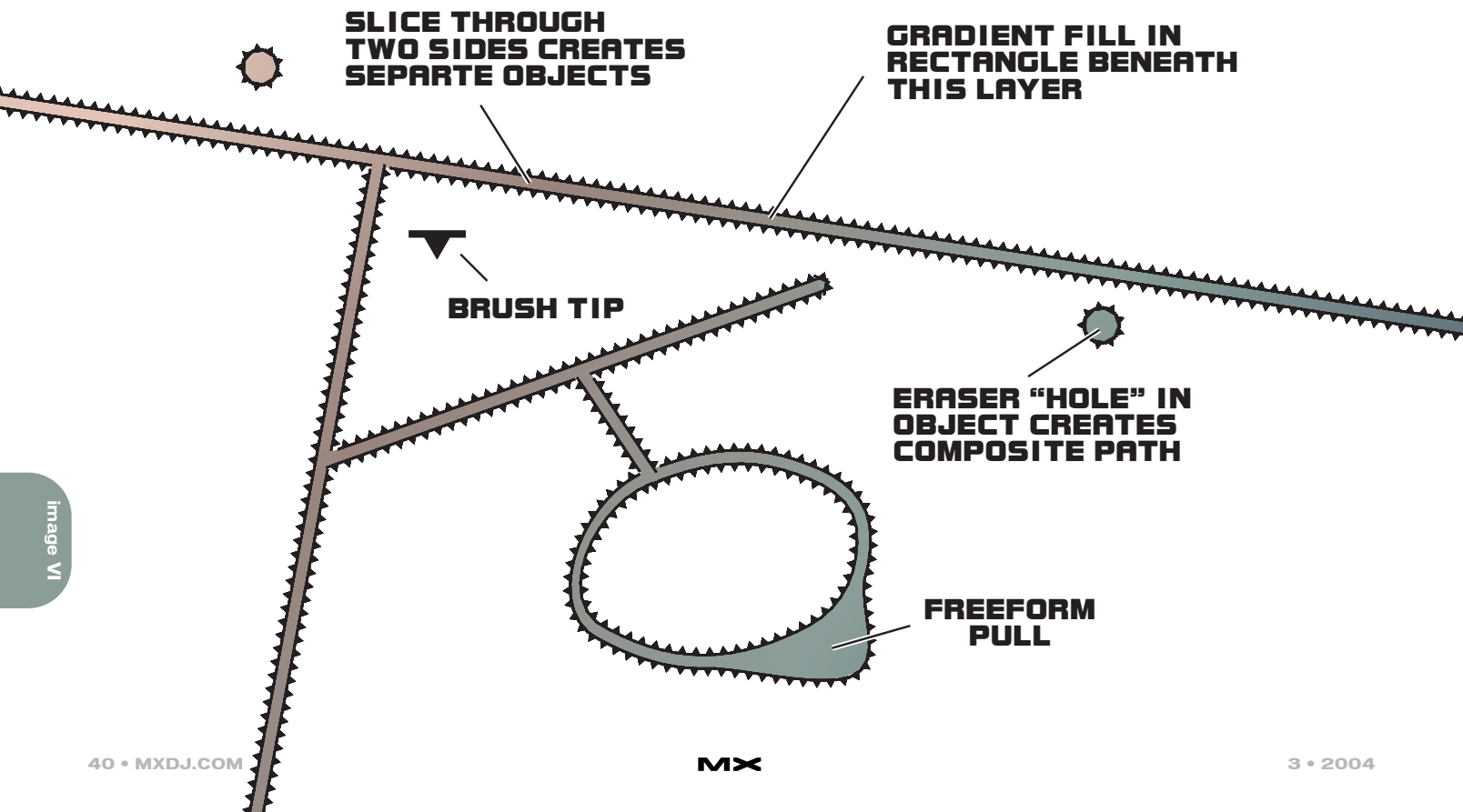


Image VI

Bézier Curves in FreeHand and Flash

image VII



No matter which method you use, the paths will be separated at every selected point. To reconnect the path sections, use the same methods with the Join command.

Eraser Tool

This tool is new in FreeHand MX. It works similar to the Knife tool, only you get a lot more bang for your buck. Double-clicking the Eraser tool icon brings up a dialog box where you can set minimum and maximum widths for the eraser tip. The numbers relate to the units of measure you have the document set to. Here's where the Wacom tablet comes into play again, because now you can draw freehand inside objects. Tip size can also be changed as you're working by pressing the keyboard arrow keys. Image VI shows a little of what can be done on a map. A rectangle was given a gradient fill, and another rectangle with a solid color was placed directly above it. The Eraser tool was given identical Min and Max widths to create a consistent line width; then, holding down the Option/Alt key to constrain the tool to a straight path, lines were drawn into the rectangle. Draw in freehand (Pencil tool) style without the modifier keys. If the path cuts through two sides of the object on the first cut, the object will become separate objects. That means that every section you want to reapply the Eraser to will have to be selected. However, punching a small hole prior to the "real" editing with the Eraser creates a compound path that will remain selected as you draw with the tool. Another bonus of this method of drawing is that any erasures take on the stroke attributes of the object it's erasing. In Image VI, a custom brush tip was applied in a spray pattern to add texture to the strokes. The Eraser tool does not work on live text.

Image VII goes a little further with the usefulness of the Eraser tool. The

A path drawn in FreeHand and imported to Flash sometimes loses its original shape, and a path drawn in Flash may be different when the file is opened later. What in the world is going on?

The reason the rules are so flexible is mainly because FreeHand's PostScript format utilizes Cubic Bézier curves, and Flash's SWF files are made of Quadratic Bézier curves – I know, your eyelids just got very heavy. The short definition of a Bézier curve: points on a page are connected by a path and described in computer code as a mathematical equation. For a more detailed description you can read all about Bézier curves at www.moshplant.com/direct-or/bezier/. FreeHand's Bézier curves utilize a control handle for each end segment of the curve, whereas Flash's curves have only one control handle for the curve.

That means when you draw something in FreeHand or Flash, both Pen tools use control handles at each end of a path, and the initial path is drawn the same way. But when it comes to editing the curvature of the path different methods must be employed. In Flash, after a path is drawn, use the Pointer tool to modify the path by clicking and holding on the path, then drag to change the curve. Flash will add points and control handles to the path as you drag, but you won't see them until you select the path with the Subselect tool. Clicking individual points will cause the control handles for the point to appear. If you click the Pointer on a path, a section of the path will be selected. You can move that section in its entirety, resize it, change its color or thickness, or delete it if you wish.

However, in FreeHand it's a different story. You usually modify the curvature of a path by using the Pointer tool or Subselect tool to adjust connector handles. The path remains contiguous unless you use one of several methods to break it. For this discussion, we'll consider that

you want the path to remain whole.

Any FreeHand user will tell you that the easiest way to modify a path is to adjust the control handles. There are two schools of thought as to when that adjustment is done. One school plots points in a semi-accurate array without regard to complete curve conformity. When this artist is done placing all the points with the Pen or Bezigon tool, he or she goes back around the path and adjusts control handles and/or points to make the path fit the shape accurately. The other school – which I believe has the



largest enrollment – places a point and adjusts the control handle immediately. When the last point is placed, the path is complete. As you place a point, keep the mouse button down and drag away from the previous point. This pulls out a pair of control handles that happen to extend equally on either side of the point you just placed. The control handle you dragged out actually affects the shape of the path to the next point you place. With Show Pen Preview selected (double-click the Pen or Bezigon tool in the toolbar), you can see what the path will look like after the next mouse click. If you don't want a curve on the next path section, click the tool on the last point. The control handle will disappear and the next point you place will give you a straight path from the last point. Dragging as you place that point will create a new set of control handles that you can manipulate to create a curved path.

color NASA image from Mars is overlaid with a rectangle with a black Lens fill set to darken by 50%. Using the eraser tool, the rock, was outlined. Then the center dark area of the rock isolated by

the erasing was selected with the Subselect tool and deleted, leaving a full-color image popping out of a severely desaturated background – without Photoshop.



CLICK

PROBLEM: Clicking anywhere on an object in FreeHand selects the fill and stroke of the object; single-clicking on an object in Flash will select either the stroke or the fill – double-clicking selects both.

SOLUTION: You'll learn to like it. You can change the width of a stroke, and the color of the fill or stroke – or both – quickly.

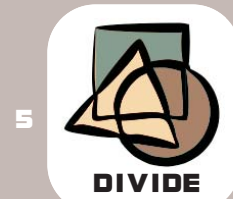


OVERLAP

PROBLEM: Why doesn't FreeHand allow overlapping objects to be removed as they can in Flash?

SOLUTION: You'll learn to like it. In Flash, if you draw objects that overlap all contiguous shapes are modifiable separately. Every new object creates a dividing operation, effectively removing the overlapped portions. In FreeHand, overlapping objects remain intact unless you use some of the path operation tools. Refer to Image VIII for examples. The original art consists of a square, a circle, and a triangle with solid fills and a brush stroke.

- **Intersect:** This command results in the lowest common denominator. Any portion of any object that does not overlap all the other objects will be deleted, with the fill and stroke attributes of the bottom-most object taking priority in the final intersected object.
- **Punch:** The top-most object punches a hole through everything beneath it.
- **Divide:** This is the Flash method. All overlapping paths are cut and closed. The result is a jigsaw puzzle of pieces that can be manipulated as you wish.
- **Union:** All the objects become united in one contiguous mass. Overlaps are deleted and the object retains the attributes of the bottom-most object in the stack.
- **Crop:** The top-most object acts as a cookie cutter and removes everything outside its shape. The action is the direct opposite of the Punch function.



↑ There's no getting around the fact that both Flash and FreeHand have some drawing tools and tricks that could be utilized in each other's program. Until that happens, take the time to experiment with the FreeHand toolset and save yourself a lot of frustration. I think you'll grow to really like FreeHand's drawing capabilities.

Acknowledgments

Many thanks to Delores Highsmith, David Spells, Peter Moody, and other engineers at Macromedia for the technical editing they provide.

Illustrator, designer, author, and Team Macromedia member Ron Rockwell lives and works with his wife, Yvonne, in the Pocono Mountains of Pennsylvania. Ron is MXDJ's FreeHand editor and the author of FreeHand 10 f/x & Design, and he co-authored Studio MX Bible and the Digital Photography Bible. He has Web sites at www.nidus-corp.com and www.brainstormer.org. guru@brainstormer.org

MX HOSTING



**FREE TRIAL on
VIRTUAL DEDICATED HOSTING!***

No obligation to buy!

▶ **866-CFX-HOST**

www.cfxhosting.com/vpstrial



FEATURES

- ▶ **Remote Management of Server**
Use Terminal Server to manage your server.
- ▶ **Support for Multiple Websites**
Host up to 50 domains. The first 10 are free.
- ▶ **Unlimited Database Support**
Includes 100MB SQL space and unlimited DSNs.
- ▶ **ColdFusion MX**
Offers access to the ColdFusion Administrator and custom tag support.
- ▶ **Custom Software**
Supports custom software installs, including:

Macromedia Dreamweaver

Macromedia Contribute

Macromedia Flash

Macromedia Studio

Macromedia Fireworks

Macromedia Flash Player

Macromedia Freehand



macromedia
ALLIANCE PARTNER

Our servers are housed in a fully-redundant data center, monitored 24/7 by experienced technicians.



APPLICATION HOSTING



macromedia
**FLASH™ COMMUNICATION
SERVER MX**



macromedia
COLDFUSION MX



macromedia
JRUN4



macromedia
BREEZE



www.cfxhosting.com **866-CFX-HOST**

* Free setup & first month free. Just \$189/month thereafter.



ColdFusion MX: A Web Services Example

Verify e-mail addresses at time-of-entry

by richard gorremans

from the first day the Internet was conceived, its primary goal was to allow people to access information stored on remote computers. Over the last couple of years, the technology of Web services has evolved not only to enhance accessing this information, but to share it as well.

Web services are in action everywhere. When you see 20-minute delayed stock quotes on a Web site, or you track eBay auctions on another, you are most likely seeing Web services in action. Look a little further and you'll find Web services that can provide these functions as well as spell checking, address verification, ZIP code to city search, and even validation of e-mail addresses. In this article, you'll see how to access one of these Web services and display the results on your own Web site.

Sending e-mail responses to users is one of the most important services you can provide to a customer visiting your site. E-mail is used for sending response messages, reports, and personal messages, and almost without exception, any site that has an online form has a field for entering an e-mail address. This information is, or at least was, one of the hardest pieces of information to verify. This article will outline a very simple application of Web services that will have you verifying

e-mail addresses at time-of-entry in no time.

Web Services

Four main components make up a Web service:

- **XML:** eXtensible Markup Language provides a language-neutral format for exchanging information.
- **WSDL:** Web Service Definition Language file in XML format that describes the Web service.
- **SOAP:** XML-based messaging framework used for Web services.
- **UDDI:** standardized directory service for registering and querying Web service meta data.

For this article, I will be concerned only with the URL location of a single Web service WSDL file.

WSDL

A WSDL file is an XML file with the following elements:

- **<definitions>:** Root element specifying namespace definitions for the Web service.
- **<types>:** Specifies data type definitions for the messages being exchanged.
- **<message>:** Defines the data being exchanged (input/output).

- **<part>:** Describes the content of the message being exchanged, typically used to name parameters being passed to the WSDL file.
- **<portType>:** Defines the operations the Web service can be called to perform.
- **<operation>:** Function or operation that can be performed with the Web service.
- **<input>:** Input parameters for the parent <operation>.
- **<output>:** Output parameters for the parent <operation>.
- **<fault>:** Message returned to the parent <operation> in the event of an error.
- **<binding>:** Protocol for accessing the operation described in the <portType>.
- **<service>:** Related port definitions.
- **<documentation>:** Information about the Web service operations.
- **<port>:** Endpoint definition for a <binding> element.

ColdFusion MX has incorporated three ways to access these WSDL files (referred to as "consuming a Web service"): the <CFINVOKE> tag, the <CFOBJECT> tag, and the createObject() function. This article will focus on using the <CFINVOKE> tag. Dreamweaver MX also has a components panel (see Image 1) where you can access various Web services, view tree diagrams of the WSDL files, and create code snippets by drag and drop.

Since there are numerous documents describing how to add a Web service to the component panel, this article will provide WSDL output file information directly from the Web service.

"In this article, you'll see how to access one of these Web services and display the results on your own Web site"

<CFINVOKE>

The <CFINVOKE> tag provides access to a registered WSDL component on a server. This WSDL component can be written as a ColdFusion Component (CFC), ASP.NET, SOAP, or in other languages that are capable of outputting a WSDL file.

The first attribute of the <CFINVOKE> tag to be populated is WEBSERVICE. The value of this attribute is the literal URL of the WSDL file for the Web service component being “consumed.” For this example, the Web service is located at: <http://soap.einsteinware.com/email/emailservices.asmx?WSDL>.

The next <CFINVOKE> attribute that will be populated is METHOD. The value of METHOD will correspond to the <part> element used for processing the request in the WSDL file. Image II shows that the <part> element is located in the s:element element and has an attribute of “ValidateEmailAddress”. The “ValidateEmailAddress” will be the value entered for METHOD.

The last attribute that will be populated for this example is the RETURNVARIABLE. This attribute specifies the name of the ColdFusion variable that will be populated with the result set returned from the Web service. Image III shows that the WSDL return values are located in the simpleType element named “CheckEmailResult”. The sub-element s:restriction specifies that the result set will be a string and the s:enumeration sub-elements show five possible results.

The possible values that can be returned are:

- Valid
- InvalidUser
- InvalidAddress
- InvalidServer
- Error

<CFINVOKEARGUMENT>

In some cases, as with the example shown, the Web service may require parameters to be supplied. Viewing the <http://soap.einsteinware.com/email/emailservices.asmx?WSDL> file shows that the <part> element named “ValidateEmailAddress” has a sub-element complexType.sequence.element with a name attribute of “emailAddress” with a type of string. This is the parameter for the e-mail address that’s to be verified.

When parameters are required to be passed to the Web service you can use the <CFINVOKEARGUMENT> tag. This tag has two attributes: “NAME” and “VALUE”.

The “NAME” attribute, for this example, is populated with the name attribute value from the complexType.sequence.element element – “emailAddress”.

The “VALUE” attribute is populated with the e-mail address to be verified.

You now have code that will “consume” a Web service located on the Einstein Technologies server that you can pass an e-mail address, and verify if the e-mail address:

- Is properly formatted
- Has a valid server
- Has a matching valid user on that server

<cfinvoke

webservice="http://soap.einstein-

ware.com/email/emailservices.asmx?WSDL" method="ValidateEmailAddress" returnvariable="aGetXMLValidEmail">

<cfinvokeargument name="emailAddress" value="#Trim(form.EmailAddress)#" />
>
</cfinvoke>

Now that you have the code for the <CFINVOKE> tag completed, the next step is adding code for getting the e-mail address (see Image IV for screen print) that will be sent as the email-Address parameter. The code used in this example is shown below:

```
<hr />
<h1>E-Mail Verification Form</h1>
<hr />
<form name="formEmailSubmit">
```

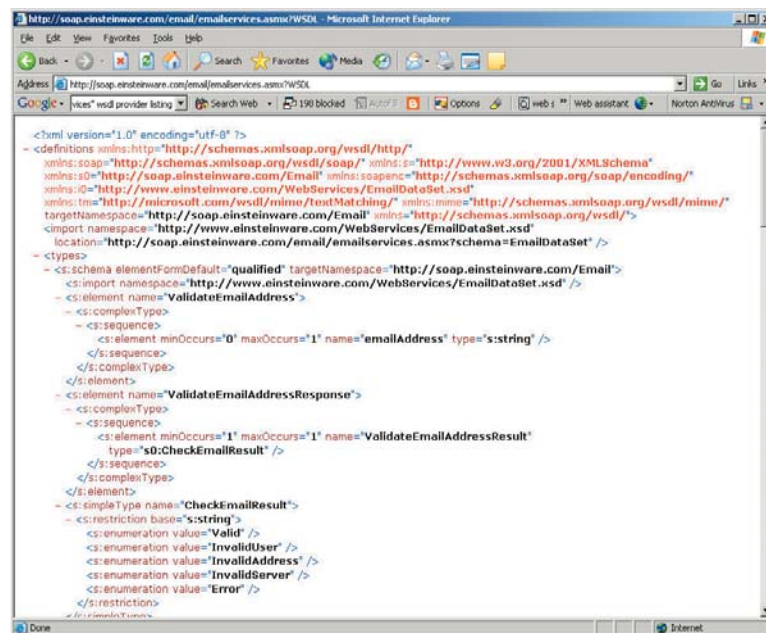


Image I

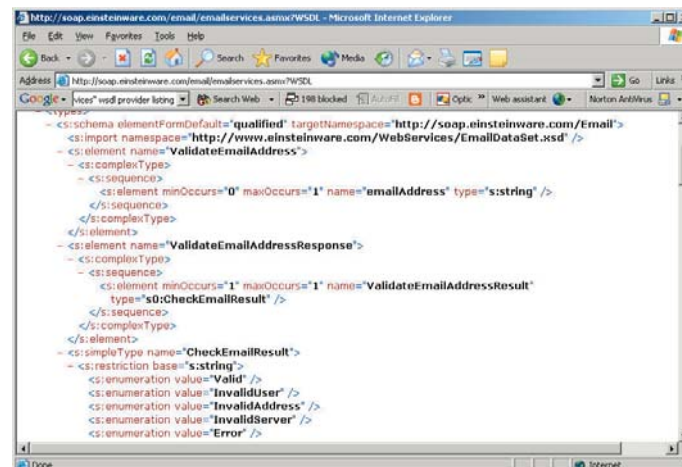
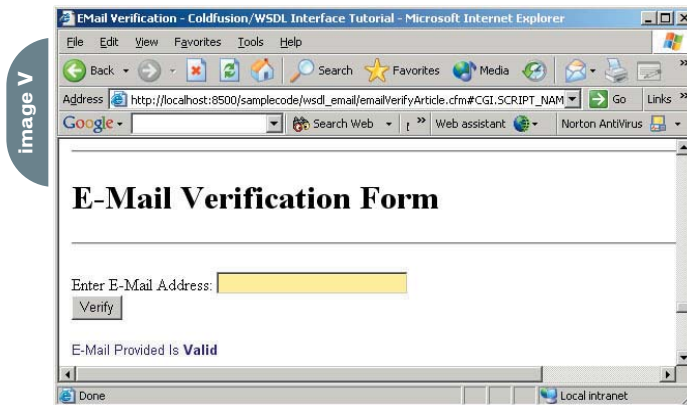
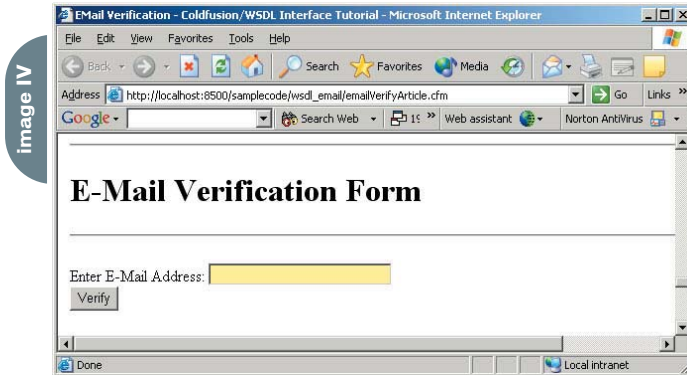
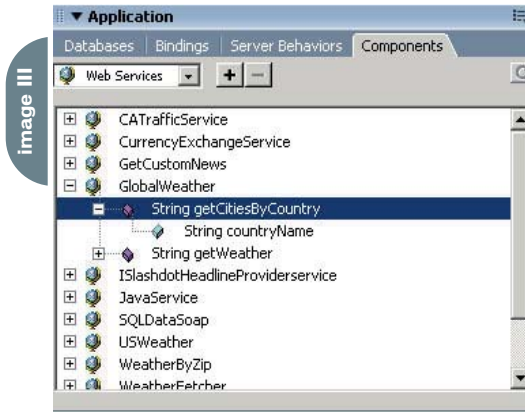


Image II



For the past four years Richard Gorremans has been working for EDFUND, the non-profit side of the Student Aid Commission, located in Rancho Cordova, California. As a senior software engineer, with over 13 years in the business, he has worked as a technical lead producing Web-based products that enable borrowers, lenders, and schools to view and maintain student loan information via the Web. xbase@volcano.net

```

action="#CGI.SCRIPT_NAME#"
method="post">
  Enter E-Mail Address:
  <input type="text" value=""
name="formEmailAddress"
maxlength="254" width="50">
  <input type="submit"
name="formSubmit" value="Verify">
</form>

```

Displaying the Results

The final step is processing the result set returned from the Web service that will be stored in the `aGetXMLValidEmail` ColdFusion variable created by the `<CFINVOKE>` tag. For this example `<CFSWITCH>` will be used to display,

dynamically, a "user friendly" message based on the value of the result set.

```

<cfswitch
expression="#aGetXMLValidEmail#">
  <cfcase value="Valid">
    E-Mail Provided Is Valid
  </cfcase>
  <cfcase value="InvalidUser">
    E-Mail Server is Valid but the User
    does not exist.
  </cfcase>
  <cfcase value="InvalidServer">
    E-Mail Server is not valid, cannot
    verify E-Mail address
    provided. Please re-enter and try
    again.

```

```

</cfcase>
<cfcase value="InvalidAddress">
  E-Mail provided is not properly for-
  matted. Please re-enter and try again.
</cfcase>
<cfcase value="Error">
  An error was encountered with the
  web service verification.
  Please try again later.
</cfcase>
</cfswitch>

```

Using the returned result set, a custom message can be displayed to the user (see Image V).

Conclusion

Sometime near the start of the Internet the term "Information Highway" was coined. Except for a very few, most drivers on this "Information Highway" got lost or missed a lot of the wondrous sights. The biggest problem, in my opinion, was that there were not enough road signs to indicate what services were available at the millions of off-ramps.

My sixth grade teacher, my mentor even now that she has been gone for many years, once told me, "The smartest person in the world is not the one who knows the most, but the one who knows where to find the most." She told me this the day she took me to the school library, gave me my first library card, and showed me how to use the card catalog to locate a book.

With this article you now have a road sign that spells out some of the services that are available on the off-ramp called Web services. Take the exit, explore the countryside, and enjoy the wonders and advantages that this article barely touches on.

Acknowledgments

A special thanks to Josh Einstein of Einstein Technologies for allowing the use of their Web service for the examples in this article.

Listings and information on hundreds of Web services can be found at the following URLs:

- www.xmethods.com
- www.salcentral.com
- www-306.ibm.com/software/solutions/webservices/uddi/
- www.webservices.org/
- www.macromedia.com/devnet/mx/coldfusion/articles/creating_cfcs.html



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, IIS and ASP problems in 60 seconds or less with no human interaction. Plus, our multi-user support system allows you to track support requests for each of your engineers individually, lookup server availability, receive a copy of all errors on your site in real time, and even monitor intrusion attempts on your site in real time. **For a new kind of easy listening, talk to EdgeWebHosting.net**

By the Numbers:

- 2 Rings or less, live support
- 100% Guarantee
- 99.998% Uptime
- 150 MBPS Fiber Connectivity
- 24 x 7 Emergency support
- 24 Hour free backup



EDGE
WEB HOSTING

What are you WAITING for?

www.edgewebhosting.net

Shared Hosting ¥ Managed Dedicated Servers ¥ Semi-Private Servers
ColdFusion ¥ SQL Server ¥ .NET ¥ Self-Healing Servers ¥ Value Priced

Win
a year
of free
hosting*



*On Shared Hosting or the equivalent value
See <http://edgewebhosting.net/cfdj> for details

customize



ColdFusion

Three years ago, I wrote an article in ColdFusion Developer's Journal discussing how to create customized roles-based ColdFusion (CF) authentication (CDJ Vol. 2, issue 3: "Customizing ColdFusion Authentication"). The article focused on showing how to implement page-level security within CF without the pains of setting up Advanced Security in ColdFusion 4.5.1.

by sarge sargent

d

Do you remember `<CFAUTHENTICATE>`, `isAuthorized()`, and `isProtected()`, and how hard it was to properly configure a userDirectory? It didn't even work with certificates! Well, since Macromedia completely removed Netegrity's SiteMinder and replaced it with the JAAS (Java Authorization and Authentication Service) in ColdFusion MX (CFMX), I figured it's time to do an update.

If you followed the example in the first article, or if you just happened to already roll your own security paradigm in CF, you were in good shape for the security changes in CFMX. This is because you already wrote your own code to perform user authentication against some back-end user database (LDAP, Active Directory, RDBMS, etc.) and assigned user roles based on group memberships within the user database. (Code examples for this article can be downloaded from www.sys-con.com/mx/sourcecf/cfm).

Additionally, you should have already coded a mechanism for authorizing logged-in user access to features and functionality within your applications. If you were diligent, you even included logic to ensure your authentication code only ran once per user session. CFMX provides this framework for you with the CFLOGIN scope and ColdFusion Components (CFCs).

Keeping in line with the first article, the scope of this article also focuses on user authentication and access at the page level within a given CFMX application. Naturally, this begs questions about other security concerns such as protecting CF templates with OS-level permissions and integrating CFMX with Web server access controls (e.g., digest security). These are all beyond the scope of this article but if there's interest, I will



cover these and other topics in future articles. Again, this article will focus on the basics of using the CFLOGIN scope to authenticate and authorize users via CFCs.

User Security Basics

Let's talk basics. Authentication is the process of identifying users – that they are who they say they are. This is typically performed with a username/password challenge, but can be as seamless as using X.509 certificates, or as conclusive as biometric devices. When answering the challenge, a query validates the user's input against some form of user store: RDBMS, LDAP, Active Directory, etc.

Authorization is the process of identifying the rights and permissions of the authenticated user. The user store is also the deposit for these rights or group memberships.

CFMX provides new tags and functions to control user security. <CFLOGIN> provides a container for performing user authentication and authorization and instantiates the CFLOGIN structure. Code inside the CFLOGIN tags executes only for unauthenticated users. The CFLOGIN structure contains two variables: CFLOGIN.name and CFLOGIN.password. Populate these variables with one of the following:

- j_username and j_password form fields
- username and password values (or hashes) passed in the Authorization header in Web Server Authentication (Basic, NTLM, Digest, etc.)
- setCredentials method of a Flash Remoting Call

<CFLOGINUSER> identifies (or logs in) the authenticated user to CFMX. It requires three parameters: name, password, and roles. Typically, you pass the CFLOGIN.name and CFLOGIN.password values to CFLOGINUSER. The roles attribute is a comma-separated list of authorized application roles. The getAuthUser function retrieves the CFLOGINUSER name attribute; the isUserRole function confirms whether the user is a member of the specified role. Finally, <CFLOGOUT> logs the user out of the CFMX application, and destroys all traces of the user's

ID, password, and roles.

In order to implement page-level security in CFMX, we need to authenticate the user, figure out the user's permissions, and then check those permissions on protected pages/sections of the application. We will use a simple login form to present the authorization challenge (see Code I). We will code a CFC to handle the authentication and authorization, instantiate the CFC inside the CFLOGIN section of the Application.cfm, and store the authenticated user's group memberships (or access levels) within a session-level structure.

The Auth CFC

The heart of our security paradigm is the CFC, so let's start there. As I said, authentication and authorization happen against some sort of user directory. You pick your poison: LDAP, Active Directory, RDBMS, NT SAM, etc. To continue in the spirit of the first article I will use an LDAP as my user directory. However, this time I will be using a username/password combination instead of X.509 certificates.

The Auth CFC will have four methods: init, authenticate, authorize, and setAuthUser. The init method is a "default" constructor that returns an instance of the CFC. It sets some default LDAP connection values for use by other methods in the CFC. You will also notice the <cfset init()> in the *pseudo* constructor area – the area after the opening <CFCOMPONENT> and before the first <CFFUNCTION>. Since the code in this area automatically runs upon CFC instantiation, the Init function will fire even without invocation (see Code II).

For more information on using the init method as a constructor, see Rob Brooks-Bilson's "Top Ten Tips for Developing ColdFusion Components" at www.oreillynet.com/pub/a/javascript/2003/09/24/coldfusion_tips.html.

Next is the authenticate method. This method requires two arguments: username and password. Pass these arguments to the Username and Password <CFLDAP> attributes, providing authentication to the LDAP. Upon successful authentication (i.e., a valid username/password pair in the LDAP), the method returns true; otherwise it

returns false. The Boolean values allow for efficient CFIF evaluations, such as <CFIF auth.authenticate(#Form.j_username#, #FORM.j_password#)>. (See Code III.)

The authorize method also requires the username and password arguments. Because the username and password are passed directly to the LDAP, the authorize method also provides a means of authentication. However, this method returns a comma-separated list of groups to which the user belongs (see Code IV). Within your code – or even at the Web server level – you can create access control lists (ACLs) to assign permissions to your LDAP groups. The application/system rights and permissions assigned to the groups extend to its members. These group memberships are roles within the CFMX security paradigm.

We now have simple methods for authentication and authorization within our application. However, it is not very efficient to authenticate the user without retrieving any of his or her attributes from the LDAP; nor is it efficient to make two separate CFC calls to build a user object. The setAuthUser method provides this functionality. The LDAP query in setAuthUser provides the same authentication as the authenticate method, but it also retrieves useful user attributes and stores them in a local structure. Next setAuthUser calls the authorize method to retrieve the user's group memberships, and then adds the returned list to the local user structure. Finally, setAuthUser returns the user structure (see Code V).

Notice that I wrap the body of the methods in Try-Catch blocks. This standard error-trapping technique provides valuable debugging information. <CFLDAP> now retrieves the actual error returned by the LDAP server instead of throwing a generic error message. Log the errors to a customized log file with <CFLOG>. The <CFTHROW> enables all <CFCATCH> in calling templates to display the LDAP error caught in the CFC method.

The Login Structure

Let's now turn our attention to the Application.cfm (see Code VI). First, create the CFMX Application framework with a

<CFAPPLICATION> tag. Enable SESSION-MANAGEMENT and a SESSIONTIMEOUT (or use the default timeout specified in the CFMX Administrator). CFMX 6.1 adds the LOGINSTORAGE attribute to <CFAPPLICATION>, which specifies whether to store the authentication information in a non-persistent cookie (default) or the Session scope. Use the default LOGINSTORAGE=Cookie for this example. Next, code some conditional logic to control user logout. You can do this nicely by wrapping <CFLOGOUT> in a <CFIF> block that checks for a URL or FORM variable. You can strengthen the logout by explicitly clearing the SESSION scope.

The <CFLOGIN>...</CFLOGIN> provides the security container for processing the authentication code for every unauthenticated user request. Performing this authentication only once during the user's session is optimal. The conditional logic forces the user to the login form if the CFLOGIN scope is not instantiated. The j_username and j_password submitted from the loginform.cfm instantiate the CFLOGIN scope.

Begin the authentication process by instantiating the auth.cfc into a shared scope. The Application scope makes the most sense because you want the CFC to persist across user sessions. Use <CFOBJECT> or createObject() to instantiate the CFC instead of <CFINVOKE>. <CFINVOKE> transiently invokes the CFC but will not persist the instance. Call the setAuthUser method to authenticate the user, retrieve the user's group memberships, and build an object containing the user's memberships and other LDAP attributes. Return this object into a Session-level container for the current user.

If setAuthUser() is successful, the Session.User variable will contain the authenticated object structure, providing a simple user profile structure that can be used throughout the site. Now we can log the user into CFMX with <CFLOGINUSER>. Use the CFLOGIN.name and CFLOGIN.password variables for the username and password attribute values. Pass the Session.User.Group value to the CFLOGINUSER to authorize the user for those roles. Then set a Session-level variable that indicates the login status.

The final construct in the CFLOGIN container is a CFCATCH block that handles any errors in the login process. It is important to wrap all of the security code in the Try-CATCH syntax for error control. The code in this block sets the login status indicator to false, creates a login failure message, and returns the user to the login form. The CFCATCH.Message will return any error messages thrown from the CFC methods.

User Authorization

Now that CFMX has authenticated the user, and his/her profile information and group memberships are stored in a Session-level object, use that information to provide authorization throughout the application. Use getAuthUser() to retrieve the username of the logged-in user. Implement access control within your templates by wrapping objects, code, or even the entire template itself in an IF-Else block that uses isUserInRole() to authorize the logged-in user for the protected functionality. The index.cfm provides an example of retrieving the logged in username, displaying the user profile, and using roles-based authorization to display links and/or text (see Code VII).

Conclusion

Leveraging ColdFusion Components and the security tags and functions in ColdFusion MX provides a robust system for roles-based authentication. Persistent CFCs provide a layer of abstraction and code reuse. The CFLOGIN container runs security code for any unauthenticated user request, ensuring that the challenge and authentication of the user happens only once. The built-in functions allow granular page-level access control. These features can augment any existing custom security logic for your CFMX applications.

Notes

- **CFLOGIN LoginStorage attribute:** At the time of this writing, the CFLOGIN LoginStorage attribute contains a bug (53320) when specifying SESSION. This bug does not completely clear the internal security scope when logging out of CFMX via <CFLOGOUT>.

Although the session data is tied to the logged-in user, the bug results in a cached user login exposable to the next user login. This bug does not exist when using cookies to store the authentication information.

CFMX uses a base64-encoded string containing the application name, username, and password as the in-memory cookie value (CFAUTHORIZATION_applicationname) sent with every page request. You may want to consider using SSL and domain-level cookies to protect this information, or utilizing the Session scope to persist this authentication information. You can read more about the LOGINSTORAGE attribute in the CFMX LiveDocs at <http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/appsec10.htm>.

- **CFC Instantiation versus invocation:** The code accompanying this article uses createObject() to instantiate the Auth CFC (auth.cfc). It is common practice for developers to use multiple CFINVOKE tags to invoke CFC methods. Accessing CFCs in this manner does not create a persistent instance of the CFC, and therefore does not run the instance data in the CFC pseudo-constructor. Although this constructor code is not required to execute the CFC, it is a best practice to utilize this area. This is why I advocate the use of <CFINVOKE> for method invocation of persistent CFCs instantiated with createObject() and <CFOBJECT>. Consult the CFMX LiveDocs for more details of CFC instantiation and invocation at <http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/buildi10.htm>.
- **Access control:** Most Web servers provide site and directory-level access control, requiring user authentication to access folders or directories containing the Web site files. I am not privy to native methods of extending this functionality beyond the template and/or page level to the elements within the code (e.g., links, text, variables, etc). This custom design does extend the Web server's own access control to the element level via CF, utilizing the same user database for authentication. ☺

Sarge Sargent is the technical editor of the CFMX section of MX Developer's Journal, and a senior product support engineer with Macromedia's MX Professional Services. He is coauthor of Advanced ColdFusion MX Application Development and a contributing author for Inside ColdFusion MX. ssargent@macromedia.com



code I

```

<html>
<head>
<title>Customized CFMX Authentication</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
</head>
<body>
<H2>Please Log In</H2>
<cfoutput><!--#### Display login error message on failure.
####-->
<cfif isDefined('Session.goodLogin') AND NOT
Session.goodLogin><span style="color:#FF0000">#loginmes-
sage#</span></cfif>
<cfform name="LoginForm" action="#CGI.script_name#"
method="post">
  <table>
    <tr>
      <td>username:</td>
      <td><cfinput type="text" name="j_username"
required="yes" message="You must enter a username!"></td>
    </tr>
    <tr>
      <td>password:</td>
      <td><cfinput type="password" name="j_password"
required="yes" message="You must enter a password!"></td>
    </tr>
  </table><br>
  <input type="submit" value="Log In">
</cfform>
</cfoutput>
<p>&nbsp;</p>
</body>
</html>

```

code II

```

<cfcomponent displayname="Auth" hint="Authenticate and
Authorize LDAP users">
<!--#### Psuedo constructor area automatically runs when
component is instantiated. ####-->
<cfset init()>

<!--#### Init function returns an instance of the compo-
nent ####-->
<cffunction name="init" access="public" output="yes"
hint="Initialize this CFC">
  <cfargument name="Start" default="dc=macromedia, dc=com"
required="Yes" type="string">
  <cfargument name="Server" default="localhost"
required="Yes" type="string">
  <cfargument name="Filter" default="uid="*" required="Yes"
type="string">
  <cfargument name="Scope" default="SUBTREE" required="yes"
type="string">
  <cfargument name="Attributes" default="*" required="no"
type="string">
  <cfargument name="Port" default=389 required="no"
type="numeric">
  <cfargument name="Sort" default="cn" required="no"
type="string">
  <cfargument name="SortCtrl" default="ASC" required="no"
type="string">
  <cfargument name="Sep" default="" required="no"
type="string">
  <cfargument name="Delimit" default="" required="no"
type="string">
  <cfset variables = duplicate(Arguments)>

```

```

<cfreturn this />
</cffunction>

```

code III

```

<!--#### Authenticates the user and returns true on suc-
cess, false on failure. ####-->
<cffunction name="authenticate" access="remote" return-
type="boolean" output="No" hint="Performs simple LDAP User
Authentication">
  <cfargument name="Username" type="string" required="yes"
default="" hint="Username">
  <cfargument name="Password" type="string" required="yes"
default="" hint="User password">
  <cftry>
    <cfldap action="QUERY"
name="getLdapUser"
attributes="uid"
start="#variables.Start#"
scope="#variables.Scope#"
filter="uid=#Username#"
server="#variables.Server#"
port="#variables.Port#"

username="uid=#lUsername#,ou=Employees,#variables.Start#"
password="#lPassword#">
    <cfif getLdapUser.RecordCount>
      <cfreturn true>
    <cfelse>
      <cfreturn false>
    </cfif>
  <!--#### Error Handling: Logs and throws error returned
from LDAP. ####-->
  <cfcatch type="any">
    <cflog text="Error in auth.cfc method authenticate() -
Error: #cfcatch.message#" type="Error" file="authentication
application="yes">
    <cfthrow message="#CFCATCH.Message#">
  </cfcatch>
</cftry>
</cffunction>

```

code IV

```

<!--#### Queries LDAP for user group memberships and
returns them as a string if successful. ####-->
<cffunction name="authorize" access="remote"
returntype="string" output="no" hint="Performs simple LDAP
Group Membership lookup.">
  <cfargument name="Username" type="string" required="yes"
default="" hint="Username">
  <cfargument name="Password" type="string" required="yes"
default="" hint="User password">
  <cftry>
    <cfldap action="QUERY"
name="getLdapGroups"
attributes="cn"
start="#variables.Start#"
scope="#variables.Scope#"

filter="(&(objectClass=groupofuniquenames)(uniquemember=uid=
#arguments.Username*))">
    sort="#variables.Sort#"
    sortcontrol="#variables.SortCtrl#"
    server="#variables.Server#"
    port="#variables.Port#"
    username="uid=#arguments.Username#, ou=Employees,#vari-
ables.Start#"
    password="#arguments.Password#">

```



```

<cfif getLdapGroups.RecordCount>
  <cfset this.roles = valueList(getLdapGroups.cn)>
  <cfreturn this.roles>

<cfelse>
  <cfreturn>
</cfif>
<!--##### Error Handling: Logs and throws error returned
from LDAP. ####-->
<cfcatch type="any">
  <cflog text="Error in auth.cfc method authorize() -
Error: #cfcatch.message# type="Error" file="authorization"
application="yes">
  <cfthrow message="#CFCATCH.Message#">
</cfcatch>
</cftry>
</cffunction>

```

```

<!--##### Authenticates and Authorizes the user, and
returns a structure containing the user's attributes and
roles. ####-->
<cffunction name="setAuthUser" access="remote"
returntype="struct" output="No" hint="Returns an authenti-
cated LDAP user object.">
  <cfargument name="Username" type="string" required="yes"
default="" hint="Username">
  <cfargument name="Password" type="string" required="yes"
default="" hint="User password">
  <cftry>
    <!--##### Authenticates the user and retrieves their
attributes. ####-->
    <cfldap action="QUERY"
      name="getAuthUser"
      attributes="uid, cn, dn, mail, givenName, sn,
telephoneNumber, pager, mobile"
      start="#variables.Start#"
      scope="#variables.Scope#"
      filter="uid=#Username#"
      server="#variables.Server#"
      port="#variables.Port#"
      username="uid=#arguments.username#,ou=Employees,#vari-
ables.Start#"
      password="#arguments.password#">

    <cfscript>
      if (getAuthUser.RecordCount) {
        this.siteUser = StructNew();
        //Concatenate the User's name from the LDAP attributes
for convenience
        StructInsert(this.siteUser, 'Name',
getAuthUser.givenName & ' ' & getAuthUser.sn, True);
        attribs = getAuthUser.ColumnList;
        //Loop the LDAP results and store them in a user
structure
        for (i=0; i LT ListLen(getAuthUser.ColumnList); i =
i+1) {
          col = ListFirst(attribs, ',');
          val = "getAuthUser."&col;
          StructInsert(this.siteUser, col, evaluate(val),
True);
          attribs = ListDeleteAt(attribs, 1, ',');
        } //END FOR
      } //END IF
      //Retrieve the user's group memberships and add them to
the siteUser structure

```

```

this.roles = authorize(argu-
ments.username, arguments.password);
  StructInsert(this.siteUser, 'Groups', this.roles);
  return this.siteUser;
</cfscript>
<!--##### Error Handling: Logs and throws error returned
from LDAP. ####-->
<cfcatch type="any">
  <cflog text="Error in auth.cfc method setAuthUser() -
Error: #cfcatch.message# type="Error" file="setAuthUser"
application="yes">
  <cfthrow message="#CFCATCH.Message#">
</cfcatch>
</cftry>
</cffunction>

```

```

<cfsilent><!--##### Remove the white space left by
Application.cfm ####-->
<cfapplication name="CFMXAUTH" sessionmanagement="Yes"
loginStorage="COOKIE">
<!--##### Logout Control: Clear user's Session information
and logout ####-->
<cfif isdefined("form.logout") or isdefined("url.logout")>
  <CFLOGOUT>
  <cfset StructClear(Session)>
  <cflocation url="index.cfm" addtoken="no">
</cfif>

<!--##### Security Container: Only runs if current user is
not logged into CFMX. ####-->
<CFLOGIN>
<!--##### If the CFLOGIN scope is not instantiated, force
user to login. ####-->
  <cfif not IsDefined("cflogin")>
    <cfinclude template="loginform.cfm">
    <cfabort>
  <cfelse> <!--##### CFLOGIN instantiated, begin authenti-
cation/authorization code. ####-->
    <cftry>
      <!--##### Instantiate the component into a shared scope
for reuse. ####-->
      <cfscript>
        if(not isDefined('Application.ldapLogin')) {
          // Create CFC in Application scope if it does not
exist.
          Application.ldapLogin = createObject("component",
"auth");
        }
        // Create Session-level object to hold returned authen-
ticated user object.
        Session.User = Application.ldapLogin.setAuthUser(CFLO-
GIN.name, CFLOGIN.password);
      </cfscript>
    <cfif isdefined('Session.User')>
      <!--##### Log user into CFMX and authorize for specified
roles. ####-->
      <CFLOGINUSER name = "#CFLOGIN.name#" password = "#CFLO-
GIN.password#" roles="#Session.User.Groups#">
      <!--##### Create Session-level login indicator. ####-->
      <cfset Session.goodLogin = true>
    </cfif>
    <!--##### Trap any errors, including those thrown by the
CFC methods. ####-->
    <cfcatch type="any">

```



code VII

```
<cfset Session.goodLogin = false>
<!--#### Login message to display on login form. ####--
->
<cfset loginmessage = "Invalid Login: " &
CFCATCH.Message>
<cfinclude template="loginform.cfm">
<cfabort>
</cfcatch>
</cftry>
</cfif>
</CFLOGIN>

<!--#### Additional CFML here. ####-->
<cfset Request.DSN = "test">
</cfsilent>
```

```
<html>
<head>
<title>Customized CFMX Authentication</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<style type="text/css">
.Normal {
font-family: Arial, Helvetica, sans-serif;
font-size: 11 pt; font-weight: normal; color: #000000;
}
.header {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 11 pt; font-weight: bold; color: #000000;
}

.Tasks {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 10 pt;
}
</style>
</head>

<body class="Normal">
<cftry><!--#### Display currently logged in Username and
Roles ####-->
<cfoutput><p>Welcome! You are currently logged in as <span
style="color:#00CC66;">#GetAuthUser()#</span> with the fol-
lowing roles: <span
style="color:#00CC66;">#SESSION.User.Groups#</span></p></cfoutput>

<table width="*" border="0" cellpadding="5" cellspacing="3">
<tr>
<td class="Tasks" width="25%" valign="top">
<span class="header">Task Menu</span><br>

<a href="index.cfm">Personal Page</a><br>
<!--#### Check Roles for permission to view ####-->
<cfif isUserInRole("Engineers")>
<a href="index.cfm">Basic User Tasks</a><br>
</cfif>
<cfif isUserInRole("ColdFusion")>
<a href="index.cfm">ColdFusion Product Support</a><br>
</cfif>
```

```
<cfif isUserInRole("Professional Services")>
<a href="index.cfm">Professional Services</a><br>
</cfif>
<cfif isUserInRole("Managers")>
<a href="index.cfm">Manager's Task List</a><br>
</cfif>
<p><a href="index.cfm?logout=true">Log out</a></p>
</td>
<td width="*">
<cfoutput><table width="*" border="1" summary="User
Info">
<caption class="Tasks">#SESSION.User.Name# <span
style="font-size: 8 px">(#Session.User.DN#)</span></caption>
<tr>
<th scope="row">Firstname</th>
<td>#Session.User.Givenname#</td>
</tr>
<tr>
<th scope="row">Lastname</th>
<td>#Session.User.sn#</td>
</tr>
<tr>
<th scope="row">UserID</th>
<td>#Session.User.uid#</td>
</tr>
<tr>
<th scope="row">Dept.</th>
<td>#ReplaceNoCase(Session.User.Groups, ",", " ",
"ALL")#</td>
</tr>
<tr>
<th scope="row">Email</th>
<td>#Session.User.mail#</td>
</tr>
<tr>
<th scope="row">Phone</th>
<td>#Session.User.telephonenumber#&nbsp;</td>
</tr>
<tr>
<th scope="row">Cell</th>
<td>#Session.User.mobile#&nbsp;</td>
</tr>
<tr>
<th scope="row">Pager</th>
<td>#Session.User.pager#&nbsp;</td>
</tr>
</table></cfoutput>
</td>
</tr>
</table>

<cfcatch>
<cfdump var="#CFCATCH.Message#">
<p><a href="index.cfm?logout=true">Log out</a><br></p>
</cfcatch>
</cftry>
<cfdump var="#Session.User#">
</body>
</html>
```

The bible for
<CF_Developers>



ADVERTISE

Contact: Robyn Forma
robyn@sys-con.com
(201) 802-3022
for details on rates
and programs

SUBSCRIBE

www.sys-con.com/
cfdj/subscription.cfm
1 (888) 303-5282



COLD FUSION Developer's
Journal





Architecting with Director

Writing Xtras

by tab julius

In my last article (*MXDJ*, Vol. 2, issue 2), we looked at how Macromedia Director is extensible, primarily through Xtras (plug-ins); and that there are four major types of Xtras – Scripting/Lingo Xtras, Sprite Xtras, Transition Xtras, and Tool Xtras.

Now it's time to look specifically at what's involved in rolling your own Xtra. A full treatment of all the ins and outs, gotchas and nuances of writing Xtras is unfortunately beyond the scope of this article (indeed, it would fill a small book), but we do have enough space to start to show you how to write a basic Scripting Xtra.

A Scripting Xtra (they used to be called Lingo Xtras) allows you to call C/C++ functions from within Lingo. You commonly call them in one of two ways – the first by instantiating the Xtra, as in:

```
infile =new(xtra "fileio")
```

which returns an instance you can work with. Alternatively, you can create a global command that you can just invoke from Lingo, no instantiating involved, as in:

```
printMyFile("filename")
```

If such a command existed, you could just use it directly, no objects to fiddle with, no muss, no fuss. Although the global commands might seem less troublesome to use, and are easier for beginners, the instantiation method is actually more powerful, because each instance can carry along its own set of internal variables. Thus, in our FileIO usage example above (FileIO is an Xtra that comes with Director for reading and writing text files), you can open up multiple instances, one for an input file, one for an output file, and so on, and each instance

will keep track of its own internal data.

On Windows, Xtras are really .DLLs with a special entry point; you compile them in Visual C. The Macintosh is similar, but on the Mac they're code fragments. CodeWarrior is used for compiling on the Mac.

To write an Xtra, you should really be familiar with C/C++. Some people have experimented with writing Xtras in other languages like Delphi and Visual Basic, but the developer's kit assumes you're using C/C++ and adapting it to other languages isn't trivial. Speaking of the developer's kit, you will need one. It's called the XDK (Xtras Developer's Kit) and it's free. You get it from Macromedia at their website – just search for XDK. There are XDKs for various products, such as Authorware, but in this article we will focus on writing an Xtra for Director.

The XDK for Windows has no binaries, just header files. The XDK for Macintosh now has some special binaries for Carbon development that will need to be included in your projects. The XDK includes both examples and template projects. For reasons of space, we'll simply focus on Windows in this article.

You should unpack the XDK onto your hard drive. I usually make the version of the XDK part of the filepath, as XDKs differ with successive releases. The current XDK is for Director 8.5 (yes, it lags behind the product release) and I use a folder named \XDK_D85. Within that folder will be a folder named INCLUDE, which will need to be in your compiler's include path; a folder called DOCS made up of .HTM files documenting the API; and a folder called EXAMPLES.

The Examples folder is subdivided into sections relating to each of the different common Xtra types. We'll be looking in the Examples\Script folder. That folder is further divided into DrAccess,

Skeleton, Skeleton2, and valueChecker. Of these, DrAccess and valueChecker are actual examples and Skeleton and Skeleton2 are the templates that you can base your project on.

Of the skeletons, Skeleton is the older style template, with multiple files. In Skeleton2, the template has been reduced from five files to two (it's the second version of the skeleton). In fact, Skeleton itself has undergone significant changes since the early releases, but it is only with Director 8.5 that the slimmer Skeleton2 has been introduced. In this article, though, we will use the original Skeleton; that way if you're writing an Xtra for Director 8, which only supports the original Skeleton, you won't be lost.

To begin, copy the files from the skeleton to wherever you want to be working. For example, C:\XTRASDEV\MYXTRA\, copying the winproj and source subfolders. You will have five source files:

- CREGSTER.CPP
- CREGSTER.H
- CSCRIPT.CPP
- CSCRIPT.H
- XTRA.CPP

If you choose to open up the project SCRIPT.DSW (on Windows) you will want to remove the listed source files from the workspace and reimport them. The reason for this is that they won't point to your source files in your new location, and it's just easier to reimport them. Make sure to import the .DEF file as found in your WinProj folder. Failure to import this file means that the Xtra will compile but will be ignored by Director when you go to run Director. It will seem as if it didn't exist.

Now is also a good time to correct your project settings. In the Link settings I like to compile the Xtra directly into the

Director Xtras folder (it makes debugging much easier than compiling to one place and copying it over). In the Preprocessor settings for the C/C++ language I usually correct the search path to be ..\source because the project folder is one folder over from the source folder; and I correct the path to the XDK Include folder because I don't compile from within the Xtras folder and the path is relative to the examples.

Now you can work on the files themselves. As I mentioned, there are five files in the original version of the skeleton. The first, Xtra.cpp, mainly concerns itself with versioning. You can put your version numbers here, but for now we'll leave it alone. You won't have to touch it for this example.

The CREGSTER files control the registration aspect of loading an Xtra. When Director starts up, the runtime Xtra looks in the Xtras folder for any Xtras and queries them to see what kind of Xtra they are and what capabilities they possess. This is the point at which the Xtras register themselves, and is where the CREGSTER files come into play.

For a scripting Xtra, there are three things you need to do with the CREGSTER files. The first is in CREGSTER.H. You must generate a GUID for the registration class. Each class has a GUID, or unique identifier. You need to have unique identifiers for each class, and to generate a new set for every Xtra you write; failure to do so will cause Director to complain that it has duplicate Xtras in its folder. GUIDs are generally a combination of your Ethernet address from your network card and a date/time stamp. Visual C comes with a utility in its BIN folder called GUIDGEN.EXE; you can run this program, make a GUID, and copy it into your source file.

CSCRIPT.H and CREGSTER.H both have sections where they want a GUID, and you have to give them different ones. Look in each .H file for a line that says #error PLEASE DEFINE A NEW CLSID

Although we haven't gotten to CSCRIPT.H in this discussion yet, I usually create both my GUIDs at the same time just because it's quicker.

To make a GUID, run GUIDGEN.EXE, choose the format of GUID that says DEFINE_GUID(...), and then press the Copy button to copy the GUID to the

clipboard. You can then paste that into the .H file. To make a second one, press New GUID and then Copy again. When you paste, the number will come in like:

```
// {A53645A1-557D-11d8-9F02-0010B53FC39F}
DEFINE_GUID(<<name>>,
0xa53645a1, 0x557d, 0x11d8, 0x9f, 0x2,
0x0, 0x10, 0xb5, 0x3f, 0xc3, 0x9f);
```

You will need to replace <<name>> with CLSID(CRegister) in CREGSTER.H and CLSID(CScript) in CSCRIPT.H. Then your GUIDs will be defined for the CScript class and the CRegister class.

This is a good time to point out that all of the example files have a comment at the top that says "How to Customize This File" and tells you to rename the file, adjust the included filename references accordingly, rename all the classes to be the name of your class, and so on. I can tell you that the easiest thing to do is to not bother with any of that. I used to spend all sorts of time renaming and customizing and realized it was not only a huge waste but prone to error as well. Now I just use the default file names and classes and it works just fine. As long as you keep your Xtras in separate folders, which is a good idea anyway, you won't run into trouble.

There isn't anything else in CREGSTER.H that you have to bother with in a scripting Xtra. There are no class instance variables here that you would normally set up. So, the next step is CREGISTER.CPP.

There are only two things you'll probably need to do in CREGISTER.CPP, at least for a scripting Xtra. One is that if you intend for your Xtra to run in Shockwave, you will have to declare it as "Safe for Shockwave". Declaring it as safe doesn't make it so, it just tells Shockwave that you say it is so, and Shockwave will allow the Xtra to load. It is your responsibility to make sure that the Xtra does not allow Shockwave access to the user's machine without their knowledge in any way that could violate their

security, data integrity, or privacy.

If you do want to declare it as Safe for Shockwave you'll need to incorporate Code I into the CRegister_IMoaRegister::Register function, just after the section that registers the method/message table – generally the last block of code before the return code.

You may or may not need the #define's depending on what version of the XDK you have. If the compiler complains they're already defined, comment them out.

Defining Your Message Table

The message table is where the fun starts. This is where you get to declare the commands your Xtra will support. In the original skeleton the method table appears about halfway through the file (you can't miss it). The format of the message table is something like Code II.

You can see many examples of message tables by using PUT in the message window to put the interface for other scripting Xtras (in Director MX you can use the third-party scripting Xtras button

"We don't have a lot of time to write down documentation...but with Camtasia Studio, we can document our software with video tutorials."

—Fred Shepardson, PhD, Mathematician, Management Consultant

Full-motion video tutorials of any application.



www.techsmith.com/mxdev

TechSmith
CAMTASIA
STUDIO
SHOW THE WORLD



on the message window to do the same thing).

If you look at the second line of the sample table above, you'll see where it says xtra MyXtra. That declares the internal name of the Xtra to be "MyXtra". Thus, when you instantiate it, or put the interface out to the message window for it, you'll refer to it as xtra "MyXtra", as in:

```
put interface(xtra "MyXtra")
```

Note that in the message table there are no quotes around the name itself, but there are when the command is issued from the message window.

After the declaration of the name of the Xtra you can have comments – multiple comments if you wish – then you define the commands. There is always a "new" command, but after that you can have one or more of your own defined commands. This example defines two. You can follow each command with a comment describing the command, although I don't often do this on Xtras with lots of functions lest I run past the maximum string space (remember, the message table is a string, just a big one).

Commands preceded by "*" are global commands, meaning you can just issue them from Lingo without instantiating anything. Commands without the asterisk (like the "new" command shown) are child commands and must be tied to an instance, meaning that you must instantiate the Xtra before you issue a call to that function.

After the command name, you can

optionally allow parameters to be passed in. Specify the type that is required and Lingo will do basic validation during a call to make sure that the right type is passed in. For instance, you could specify integer, integer, string, and Lingo would require the user to pass in two integers and a string; otherwise, it would throw an error. The argument types are integer, float, string, symbol, object, any, and *; the asterisk is a wild card, allowing any number of arguments, of any type. If used, it should be the last entry on the line.

Optionally, you can have names with each arg, to make it easier for the user, as shown in the FixCertainBug example; or you can omit the names and just have the arg types.

Type Any is used to allow any type – not restricted – but it's also used to allow types that aren't listed. For instance, if you wanted to pass down a member spec (e.g., member 8 of castlib 1), it won't officially be an integer, float, string, etc. For a parameter like that you would use Any to allow it in.

If you required a minimum of four parameters, the third of which can be of any type, you could do:

```
FixCertainBug integer, string, any, string, *
```

Lingo will only allow that call to go through if the user supplies at least four parameters, the first of which is an integer, and the second and fourth of which are strings. Note that Lingo will do as much validation as it can for you, but if

you use Any or * you will be responsible for validating those parts when called, to make sure that you got what you wanted from the user.

Child functions (that must be instantiated) must always have an object type as the first parameter, as in:

```
"myChild object me, string newName\n"
```

The Enum Table and CSCRIPT.H

When your Xtra is invoked from Lingo you will be called through the function ::Call, and passed in a number that tells you which command from your message table was invoked. New (the first command) is always #0; in the example message table, FixAllBugs would be #1 and FixCertainBug would be #2, etc.

It's easy to introduce bugs if you're just checking by number. If you were to move commands around in the table you'd have to reorder all your code. To keep problems to a minimum, the Xtras use an enum table, found in CSCRIPT.H, that looks like:

```
enum
{
    m_new =0,

    m_fixAllBugs,
    m_fixCertainBug,

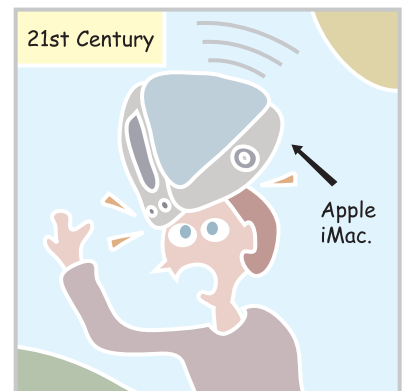
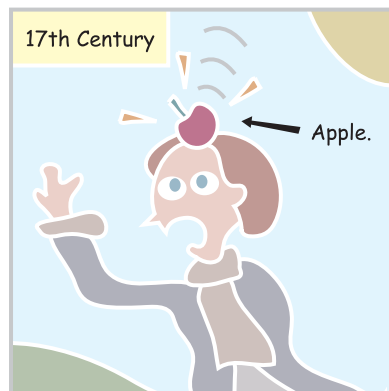
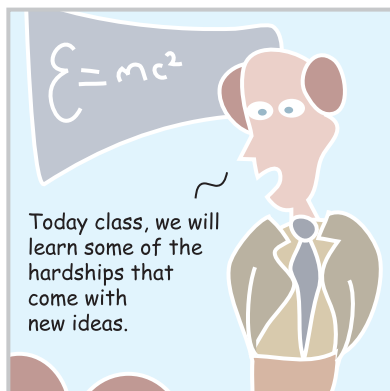
    m_XXXX
};
```

With such a table, all you have to do

Tab Julius has been writing software since the mid-70's, and now works for a software firm developing medical imaging applications, although he still does limited consulting on the side. tab@penworks.com

xile

written & illustrated by louis f. cuffari 4



is make sure the order of enums in the enum table matches the order of commands in the message table. If you move them around in the message table, you must do so in the enum table. If you delete, or comment out, entries in the message table, you must make the corresponding changes in the enum table. As long as they're in sync, things will work. If you forget to make an entry, or change an entry, then you'll find your program executing one command when you intended for it to execute another. The `m_XXX` entry at the end serves no purpose other than to let you keep a comma at the end of all of your entries, without having to remember to add or remove commas when moving enums around.

Class Instance Variables

Any variables your Xtra wants to keep around between calls should be kept in the Class Instance Variable section in `CSCRIPT.H`, rather than using globals. If your Xtra is instantiated, then the variables here are specific to each instance. If you don't instantiate your Xtra and just use global Lingo commands, they'll still work – they'll just apply to the one “global instance” of the Xtra. You only need to use true global, non-class instance variables if there's something you need to share between instances, although this practice is discouraged.

Here's Where It All Happens

Let's turn our attention to `CSCRIPT.CPP`, which is where the guts of the Xtra will be. There are three parts you need to concern yourself with:

1. The `::Call` function, which is the main entry point for the Xtra, where you will test for how your Xtra was called and react accordingly.
2. Any individual functions you wish to set up. Although you could do all your work in the `::Call` function, typically developers use `::Call` simply as a dispatcher and make a corresponding function for each command supported by the Xtra.
3. The Create/Destroy method, which is where you will typically acquire (or free) any interfaces you need – and you will typically need a few – as well as where you will initialize your instance variables.

```
// SAFE FOR SHOCKWAVE.
#define kMoaMmDictKey_SafeForShockwave "safeForShockwave"
#define kMoaMmDictType_SafeForShockwave kMoaDictType_Bool
MoaBool bItsSafe = TRUE;
/* Mark the xtra as safe for Shockwave */
if (err == kMoaErr_NoErr)
{
    err = pRegDict->Put(kMoaMmDictType_SafeForShockwave,
        &bItsSafe, sizeof(bItsSafe), kMoaMmDictKey_SafeForShockwave);
}
}
```

code I

```
static char msgTable[] = {
    "xtra MyXtra\n" \
    "-- www.penworks.com\n"
    "-- by Tab Julius <tab@penworks.com>\n"
    "--\n"
    "new object me\n"
    "* FixAllBugs      -- Magically fix all bugs\n"
    "* FixCertainBug integer bugNum, string fixName\n"
};
```

code II

```
STDMETHODIMP CScript_IMoaMmXScript::Call(PMoaMmCallInfo callPtr)
{
    /* variable declarations */
    MoaError err = kMoaErr_NoErr;

    switch( callPtr->methodSelector )
    {
        case m_new:
            /*
             * --> insert additional code -->
             */
            break;

        case m_fixAllBugs:
            err = XScrpFixAllBugs(callPtr);
            break;

        case m_fixCertainBug:
            err = XScrpFixCertainBug(callPtr);
            break;
            break;
    }

    return(err);
}
```

code III

```
MoaError CScript_IMoaMmXScript::XScrpFixAllBugs(PMoaDrCallInfo callPtr)
{
    UNUSED(callPtr);

    MoaError err = kMoaErr_NoErr;

    MessageBox(NULL, "Hello, World", "", MB_OK);

    return(err);
}
```

code IV



Each function you make will have a corresponding prototype; on some versions of the XDK the prototypes are kept in CSCRIPT.H, in other versions they are at the top of CSCRIPT.CPP. In the D8.5 XDK they are in CSCRIPT.H – look for the section `EXTERN_BEGIN_DEFINE_CLASS_INTERFACE` for CScript, which declares Call as a public interface and then lists XScriptGlobalHandler, XScriptParentHandler, and XScriptChildHandler prototypes. These are examples only of Global/Parent/Child handlers, and the names are not a requirement. In fact, I would delete them and replace them with the equivalent XScriptFixAllBugs, XScriptFixCertainBug, etc.

In CSCRIPT.CPP you will find the corresponding functions near the top of the file – again, replace the names with the names of *your* functions and delete the ones you're not using. Make them match your prototypes.

At this point, the only major thing left to do is to tweak the `::Call` function so that it invokes the functions properly. It will still be necessary to access the args passed in, and learn how to

return values, and so forth, but we'll have to defer that part until next time. For the moment we'll get it up and limping, and get it to say "Hello World", which is enough to claim victory for now.

As I mentioned, when someone calls a function in your Xtra, the `::Call` function is invoked. For our example, the `::Call` function should end up looking like Code III.

The whole flow of events is now in place. The message table lists two commands besides New – `fixAllBugs` and `fixCertainBug`. `fixAllBugs` takes no parameters. The enum table lists three enums, one for New and one each for `m_fixAllBugs` and `m_fixCertainBug`. When the user invokes `fixAllBugs()`, that will be found as the second entry (0-based) in the message table, or rather the first one after New (0). Since the entries in the enum table are in the same order as those in the message table, `m_fixAllBugs` has an internal value of 1. This is how it's supposed to work, because when `::Call` is invoked it is passed `callPtr`, which has a field, `methodSelector`, that has a value of 0..n,

up to the number of entries in your message table less one. The net effect is that by syncing an enum table to your message table, you can respond with mnemonic enum names, not by keeping track of index numbers into the table.

The code in `::Call` will switch to the case statement for `m_fixAllBugs`, which will invoke `XScrpFixAllBugs`. If you then modify the function `XScrpFixAllBugs` you can complete the cycle (see Code IV).

Compile it, and you now can restart Director. If everything is in your favor, the Xtra will be loaded, you can issue the command `fixAllBugs()` from either the message window or from a script, and a "Hello World" message box should pop up!

Conclusion

Next time we'll take a closer look at how parameter passing works, how to return data back to Director, and what gotchas to look out for; and we'll take a peek at some of the MOA (Macromedia Open Architecture) classes. Until then, enjoy! ☺

–continued from page 31

The History panel makes it really easy to build commands that automate particularly tedious or common tasks. The process of creating a Command file with the History panel is fairly straightforward: clear the History panel so you can start fresh.

Now perform a sequence of actions in the Flash MX 2004 IDE that you want to use regularly. Select the group of actions you performed in the History panel and then click the disk icon in the bottom right-hand corner. Enter a name for your command and then click the OK button.

You now have a new command in your Commands menu that when selected will execute and perform that particular sequence of actions. The `.jsfl` file has been created for you and saved into the correct directory, such that it is displayed in the Commands menu.

There are technical limitations with the History panel; some of the interactions you make with the Flash MX 2004 IDE cannot be replayed. As you play around, you will notice that certain

actions contain a red cross. These are the actions that cannot be replayed.

But it doesn't stop there; the History panel can also display the JSFL actions required to perform a particular interaction within the Flash MX 2004 IDE. This comes in handy when trying to learn how to make Flash do something with the JSAPI. So, for example, if you want to know which function to use to add a new layer to the current timeline using JSFL, you can tell the History panel to expose the function calls, by changing the view settings (see Image VII).

Then you can manually add a new layer to the timeline, which will add a new action to the list in the History panel (see Image VIII).

You can then copy that function call by right-clicking the selected actions and selecting to Copy Steps (see Image IX).

You now have the JSFL code in your clipboard, which you can then paste directly into any text editor.

Flash Panels

As I mentioned in the introduction, it is possible to add new panels to the Flash

MX 2004 IDE. You can create your own panel by simply creating a Flash movie and then saving it into the correct directory.

The panel can be opened by selecting it from the Window>Other Panels menu. The name associated with the panel you have created and thus displayed in the Other Panels menu is simply the name of the `.swf` file you placed in the WindowSWF directory. When you select your panel from the menu, your Flash movie is automatically displayed inside a panel, and the size of the panel is automatically determined by the size of your Flash movie.

I've created a Flash movie and named it `Auto Save.swf`. I then save that Flash movie into the WindowSWF directory. The next time I restart Flash MX 2004, a new option has appeared in the Other Panels menu.

When I select the new Auto Save option, a new panel opens that contains my Flash movie.

ActionScript and JSFL

Flash movies that run in the Flash MX 2004 IDE can execute a string of JSFL

using a special ActionScript function. This means that your Flash Panels and XML2UI dialog boxes that contain Flash movie controls can modify/update the Flash document from a Flash movie. The JSFL code you execute can also return a value back to ActionScript. The ActionScript function is called MMExecute, and it can be used anywhere in your Flash movie.

The syntax is simple:

```
MMExecute("yourStringOfJSFL");
```

I can create a new panel, as above, and in the Flash movie for that panel I can execute a string of JSFL that will delete a layer when a button is pressed. This would be achieved with the following ActionScript:

```
MyButton.addEventListener("click",this)
This.click=function()
{
MMExecute("flash.getDocumentDOM().getTimeline().deleteLayer();")
}
```

When the button is pressed, the MMExecute function passes the JSFL string passed as an argument to the Flash MX 2004-JSAPI Interpreter, which then runs the JSFL code in exactly the same way that a command is run.

It's also possible to access the value of a JSFL property due to the fact that the MMExecute function will return a value, so if I wanted to check that the computer that is running my Flash panel is currently connected to the Internet, I could use the following ActionScript:

```
HasInternetConnection=MMExecute("flash.isConnectedToInternet");
```

In the above code, the ActionScript variable "hasInternetConnection" will contain a Boolean value, which was received from the JSAPI Interpreter.

MMExecute will wait for a return value before it runs the next line of code in your ActionScript. It is recommended that if you want to execute a large JSFL script from ActionScript, you place the JSFL script in a .jsfl file and then use the "flash.runScript" function to execute the JSFL script within that file.

Here is an ActionScript function that

will execute the JSFL code within a .jsfl file:

```
function
executeLargeJSFLScript(fileURI)
{
return
MMExecute("fl.runScript('+fileURI+')");
}
```

And you can use it like this:

```
executeLargeJSFLScript("file:///C:/myscript.jsfl");
```


Notice that the fileURI in the above example doesn't contain back slashes, it contains forward slashes and a file:/// protocol prefix.

The following would be incorrect:

```
executeLargeJSFLScript("C:\myscript.jsfl");
```

This is a common mistake. All the functions exposed to us in our JSFL scripts that accept a fileURI as an argument, such as "flash.saveDocumentAs", require the string to contain a file:/// protocol prefix and to contain forward slashes instead of back slashes.

One really useful implementation of MMExecute would be if you wanted a Flash movie running in the Flash MX 2004 IDE to work with regular expressions. As ActionScript doesn't support regular expressions, it's possible to send them the JSAPI Interpreter using MMExecute, and the return value(s) will be returned to ActionScript.

That should be enough to get you started. Part 1 of this article covered a general introduction to the extensibility layer, and discussed the fundamental Document Object Model and the relationship between different parts of a Flash document and their associated objects. It introduced the new Flash JavaScript Editor in Flash MX 2004 along with the History panel, and showed you how to build your own Flash panels. When developing Flash extensions, it is sometimes necessary to provide a dialog box, which prompts the user to make a choice or customize settings of extensions in an intuitive and user-friendly way. In Part 2 we'll take an in-depth look at XML2UI - Flash Dialog Boxes. 

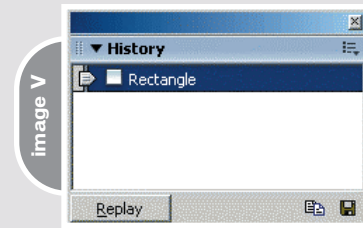


image V

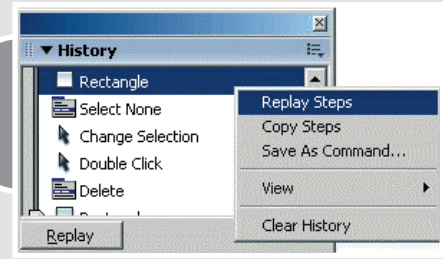


image VI

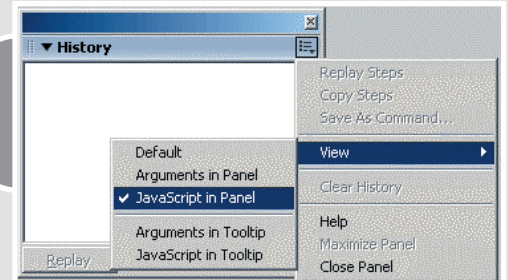


image VII

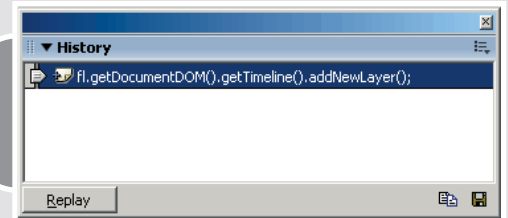


image VIII

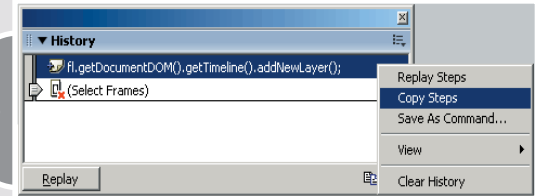
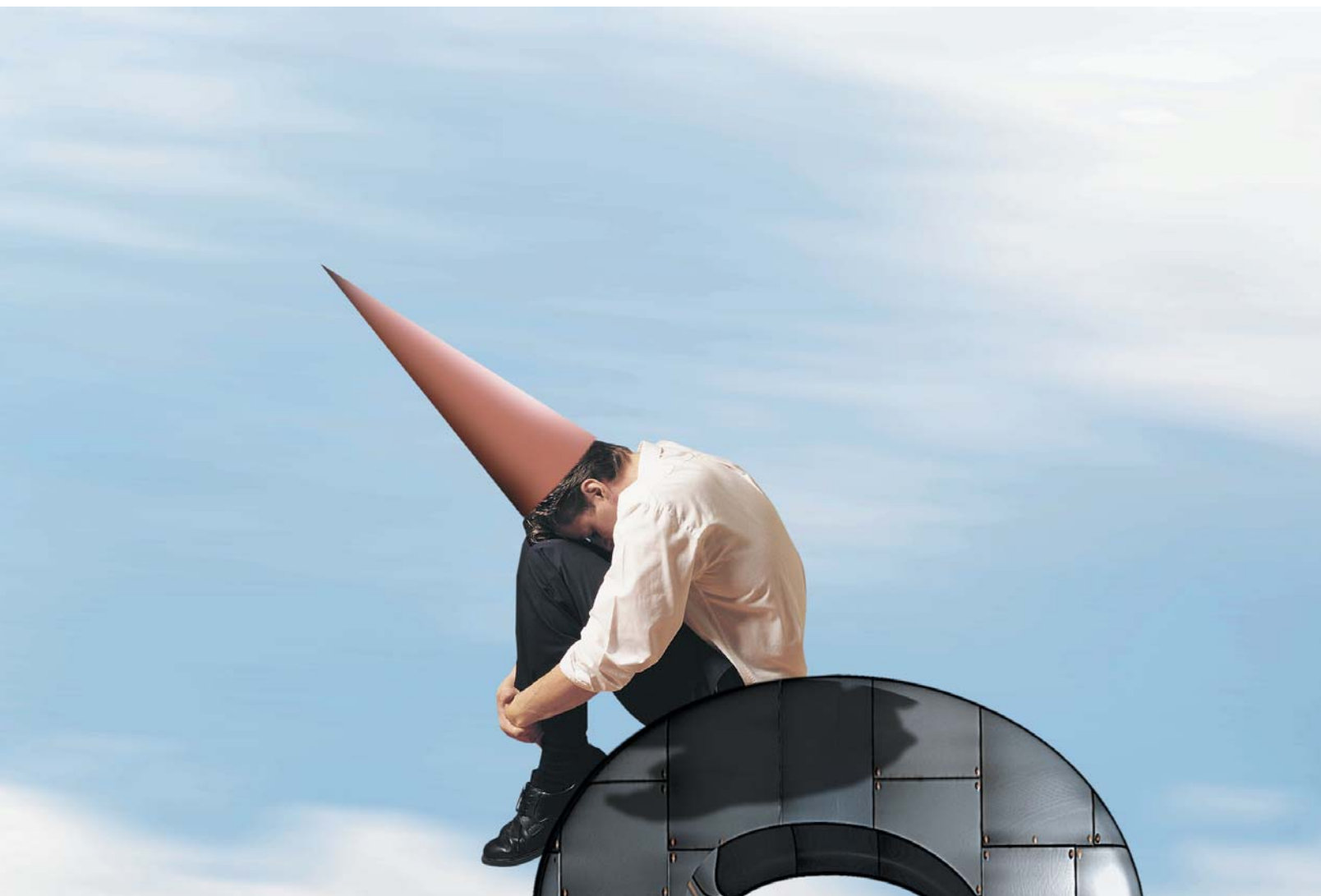


image IX

Guy Watson (aka FlashGuru) has been a well-recognized figure in the Flash community for around four years, supporting the community with tutorials and source files, moderating the large Flash community forums, and running his own Flash resource Web site - FlashGuru's MX 101. Guy was one of the two developers that created the award-winning zoom interface for Relevance and now works for Endemol UK, the creative force behind reality television, producing programs such as Big Brother and The Salon. Guy spends most of his time developing Flash games and applications for high-profile clients such as Channel 5 Television, Ladbrookes, and UK Style. guy@flashguru.co.uk

best



behavior

1

One of the great things about working with Director is that there are so many tools and functionalities, but only when you start combining these do you start to realize the real power of Director. by martin kloss



In this article we will combine three cool tools and use them to create another tool that we can use like any other in Director. Not only will it take a mere half hour to create, but it will save you hours of development time in the future (see Image 1) This probably sounds confusing so let me be more specific: I'm talking about creating a behavior.

Behaviors

Behaviors are one of the most powerful tools in Director because they allow you to create generic code snippets that can be utilized in many different situations and projects. To show you what a behavior is, let's create a simple one. Open the script window and set the script type to "behavior" in the property inspector. Now, add a property to the script:

```
property pName
```

Properties are variables that are accessible in every handler throughout the entire behavior script, but they

belong to that specific behavior so they are not global. That means you can have properties with the same name in different behaviors and they will not overwrite each other, as globals would do.

A *behavior* really consists only of properties and handlers. As with the properties, the handlers are not global but belong to the behavior script, so you cannot call a handler in a behavior like a movie script. Let's add a handler to the behavior we just created:

```
on mouseDown (me)  
    put "My name is:" && pName  
end mouseDown
```

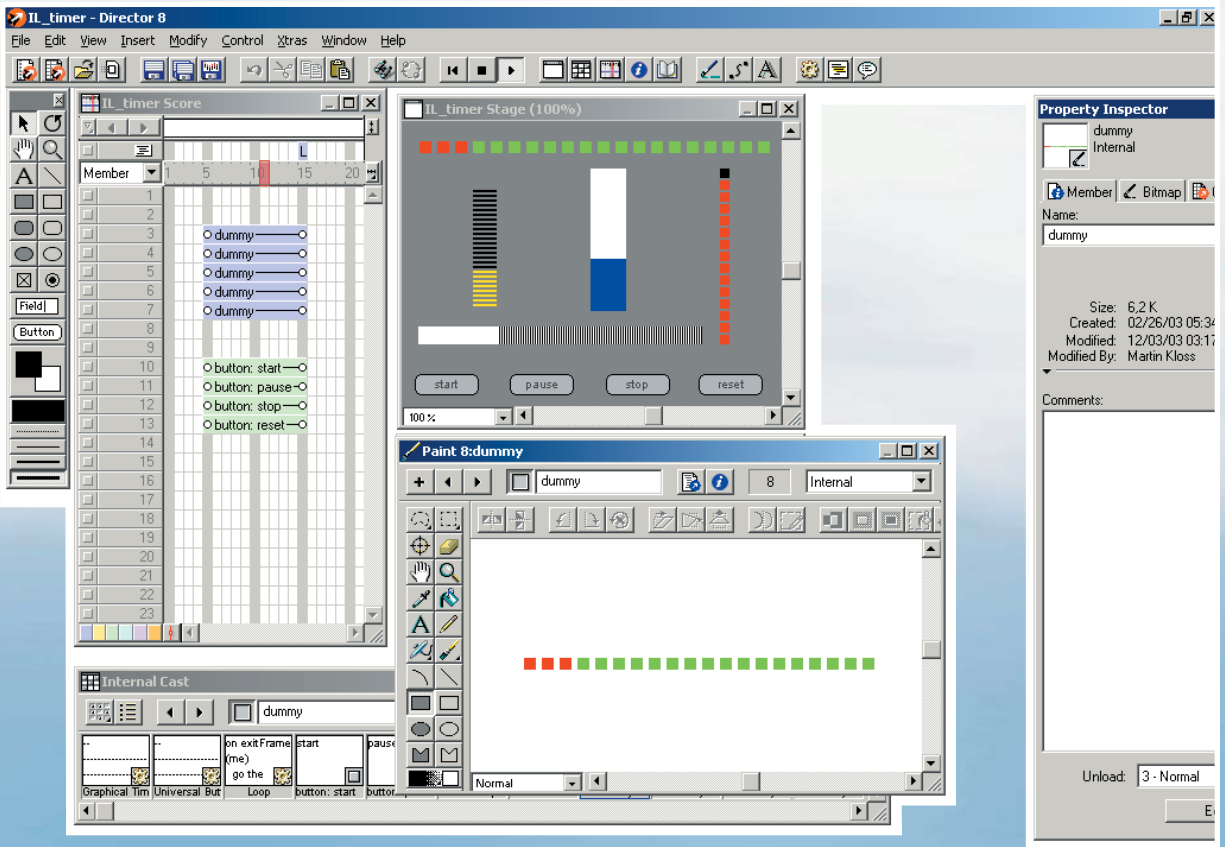
Now the behavior can react to a "mouseDown" event. But how does the behavior get this event? This is where sprites come into play. Create a button cast member (or bitmap, text, or any other type of sprite you like) and drag it onto the stage. Now drag your behavior script onto the sprite on the stage. Run the movie, click on the sprite, and watch the message window:

```
-- "My name is: "
```

The sprite gets the "mouseDown" event from Director because you clicked on it. Because it has a behavior with a handler that reacts to this event, the handler is executed and you see the output. The only problem is that the property pName does not have any value yet. That's where the cool part of behaviors needs to be explained – the getPropertyDescriptionList() handler. This handler is called when you drag the behavior onto the sprite or when you click the behavior "parameters" button in the property inspector. It's used to assign default property values to the behavior. In this case we want to give the property pName a value. In order to do that the handler needs to return a property list in a specific format that contains all of the needed information to generate a behavior dialog for the user to input the property values to be used by the behavior instance. Take a look at the handler for our simple example behavior:

```
on getPropertyDescriptionList ()
```

image 1



```

p = [#pName: [#comment: "Name",
#format: #string, #default: ""]]
return p
end getPropertyDescriptionList

```

The property pName has a description (#comment), a format (#string), and a default value in case the user does not type anything into the behavior dialog. To see the dialog simply drag the behavior onto the sprite and you will be prompted to input a value for the name property. Now type a name, hit enter, and run the movie again. When you click on the sprite again, the message window will output the value you just entered:

```
-- "My name is: Woody"
```

But the real beauty of behaviors will become obvious in a second. Create a second sprite on the stage and drag the same behavior onto the sprite. Now type another name into the behavior dialog field, run the movie again, and click on the first and then the second sprite:

```

-- "My name is: Woody"
-- "My name is: Allen"

```

The secret behind this is that you just created two totally independent instances of the same behavior from one behavior script, doing nothing but drag and drop. The behavior script is like a blueprint from which behavior instances are created when you drag the script onto a sprite, much like sprites are instances of cast members you drag onto the stage. The behavior instances don't know about each other, and they don't share any values or handlers since they only belong to the sprite they are assigned to and only receive messages from that sprite.

But what if you wanted to call a function of the behavior from somewhere in your movie, say from a movie script or from the message window? There are two commands in Lingo that let you do exactly that:

```

sendSprite(1, #reportName)
sendAllSprites(#reportName)

```

These commands send the message to either a single sprite or all sprites in the current frame. They don't call the handler of the sprite, they just send a message to the sprite. If the sprite has a handler with that name, it will automatically call that handler. If the sprite does not have a handler with that name or does not have a behavior at all, nothing will happen – the sprite will just ignore the message. So if you are not sure which channel your sprite is in or you want to send a message to all sprites, you can use sendAllSprites.

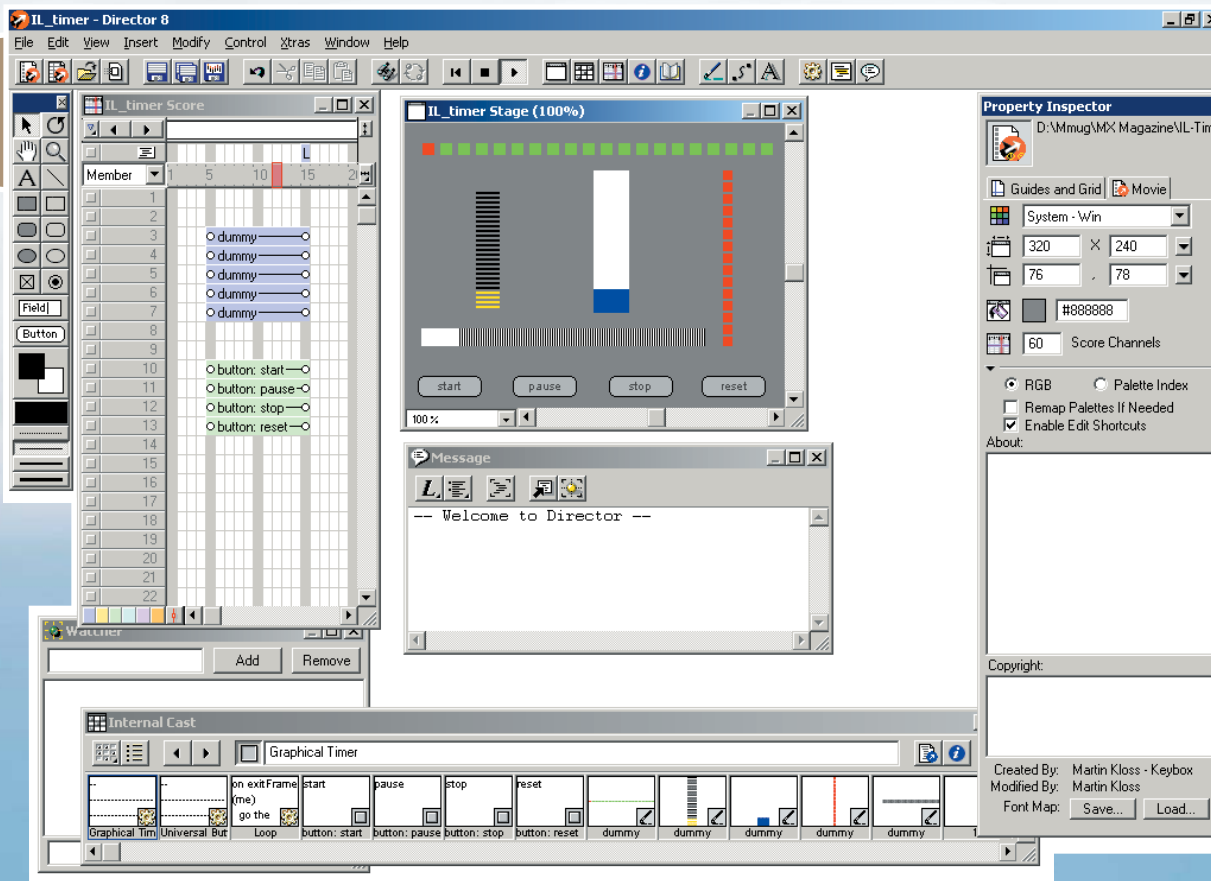
Let's add another handler to the example behavior to test this functionality:

```

on reportName (me)
    put "My name is:" && pName
end reportName

```

As in the mouseDown handler before, the "me" keyword after the handler name is a reference to the behavior instance and is always required in behavior scripts. (See the "me" entry in your Lingo dictionary for further explanations.)





Now you can call the handler by sending a message to all sprites from the message window:

```
sendAllSprites(#reportName)
-- "My name is: Woody"
-- "My name is: Allen"
```

Now you know almost everything there is to know about creating and working with behaviors. But if you're still not really sure what behaviors are, look at the behaviors that come with Director and check out some of the demo movies.

In this article, we're going to create a behavior that uses two other great functionalities in Director. The first one is called "Imaging Lingo". While this sounds a little scary at first, it's really nothing more than a set of relatively simple Lingo commands that give you access to Director's internal sprite-rendering engine. That means you can create new or manipulate existing bitmap images in Lingo.

Image Objects

When talking about images it's important to know that Director has an

internal object type called "image", which is pretty much what you'd expect it to be – an image. Cast members like bitmaps, but text members also have an "image" property that stores a pointer to the internal image object of that cast member. Create a new bitmap or text member and try to get its image into the message window:

```
put member("my picture").image
-- <image:23cdb4>
```

What you see is the reference to the image object in Director's memory. It looks complicated, but trust me, it really isn't.

You can manipulate that image, copy it, or create a new image from scratch using the "image()" function and assign the new image to the member's image property. Try the following in the message window:

```
myMember = new(#bitmap)
```

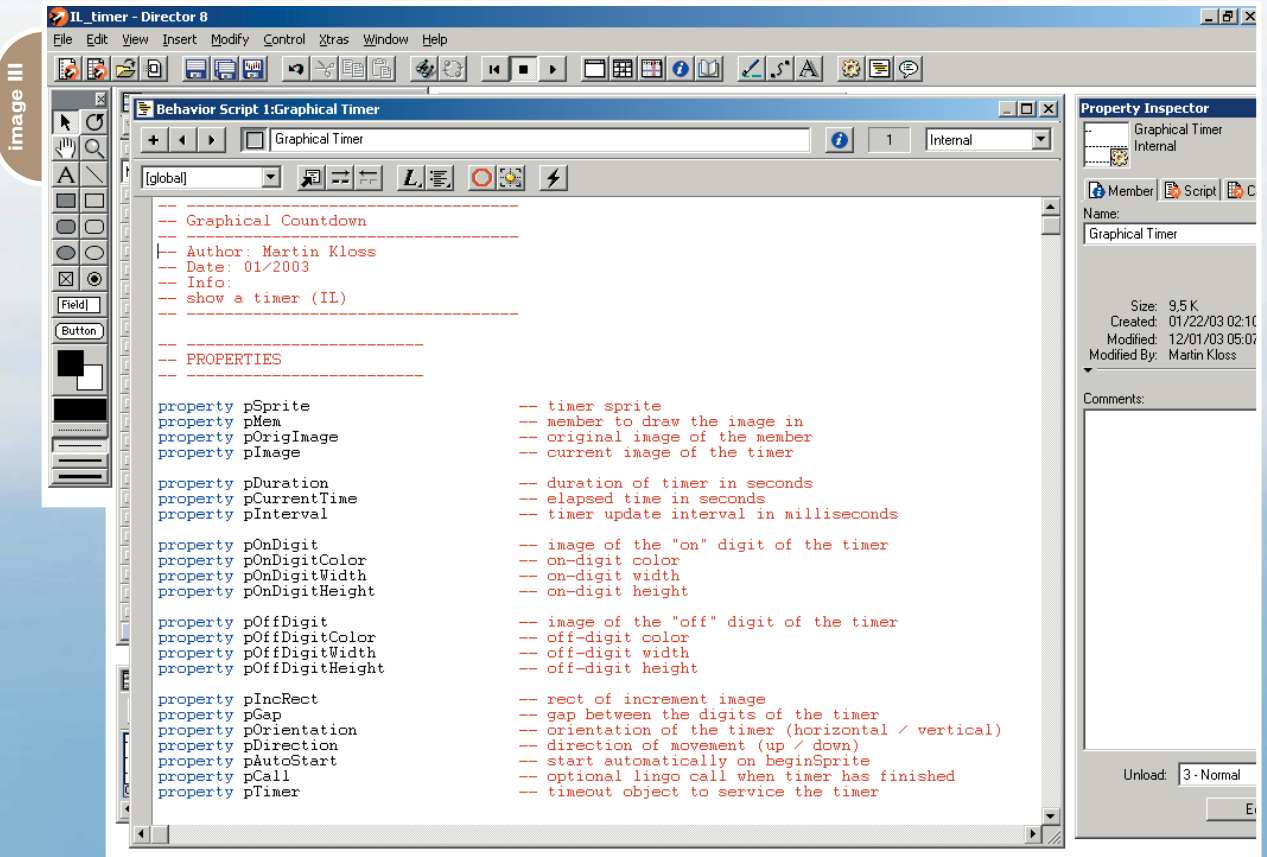
Director created a new bitmap cast member for you in the current cast

library. Open the new cast member in the paint window by double clicking it and type the following two commands in the message window:

```
myMember.image = image(100, 100, 16)
myMember.image.fill(0, 0, 100, 100,
rgb(255,0,0))
```

The paint window will now show a red square. The same thing that happens when you use the tool in the paint window to draw a square box just happened because of these two lines of Lingo code. You told Director to create a new 100x100 pixel, 16-bit image object, and assigned the new member's image property to it. In the next line you filled the whole rect of the image (0,0,100,100) with the color red (rgb(255, 0, 0)). See how easy it is to work with image objects?

Imaging Lingo really opens up a whole new world for the Director developer. If you haven't learned about this exciting functionality I recommend reading a good Director book or a few of the various online articles on the topic.



LOOK FOR YOUR **FREE...**

IT solutions >GUIDE

»Linux »Java »Web Services ».NET »XML »Wireless »Storage »Security

The Premier Resource for Today's Corporate & IT Decision Makers

VOL 1 ISSUE 1 SPRING 2004

WWW.SYS-CON.COM/IT

TECHNOLOGIES YOU NEED NOW!

How to Manage Your Ideas Using Today's i-Technologies

- » Delivering Software as Service
- » Leveraging Linux/Open Source
- » Moving to a Service-Oriented Architecture
- » Desktop Software: Migrating from Server to Client
- » Using Developer Tools to Drive Cost Out of Software
- » Application Integration
- » Storage & Security

Reaching 135,000 Corporate and IT Decision Makers

\$5.99US \$7.99CAN 12>

0 09281 01121 7

Coming this **SPRING!**



Timeout Objects

On to the next step. To complete our toolbox, we add another great functional-ity that Director has offered since version 8: the “timeout” object. Like image objects this is also an internal object, and behaves a little like a virtual metronome. It can send out messages on time intervals that are defined when you create the object.

I know this sounds a little confusing, but once you get to know timeout objects, you’ll realize that there’s really nothing complicated about them. It’s really easy to create one:

```
t = timeout("My Timeout").new(1000,
#helloWorld)
```

This creates a new timeout object with the name “My Timeout”, stored in the variable t. The first parameter sets the interval and the second parameter defines the name of the function to be called. After the timeout object has been created it starts to do what it’s told to do automatically. In this case it calls the function “helloWorld” every second (1000 milliseconds).

You can easily test it for yourself; just create a new movie script and write a simple “helloWorld” function:

```
on helloWorld
  put "Hello World"
end helloWorld
```

Start the movie and see how the timeout object calls your function by watching its output in the message window.

That’s pretty much all there is to know about working with timeout objects. Some of the functions and properties of timeout objects will be covered later.

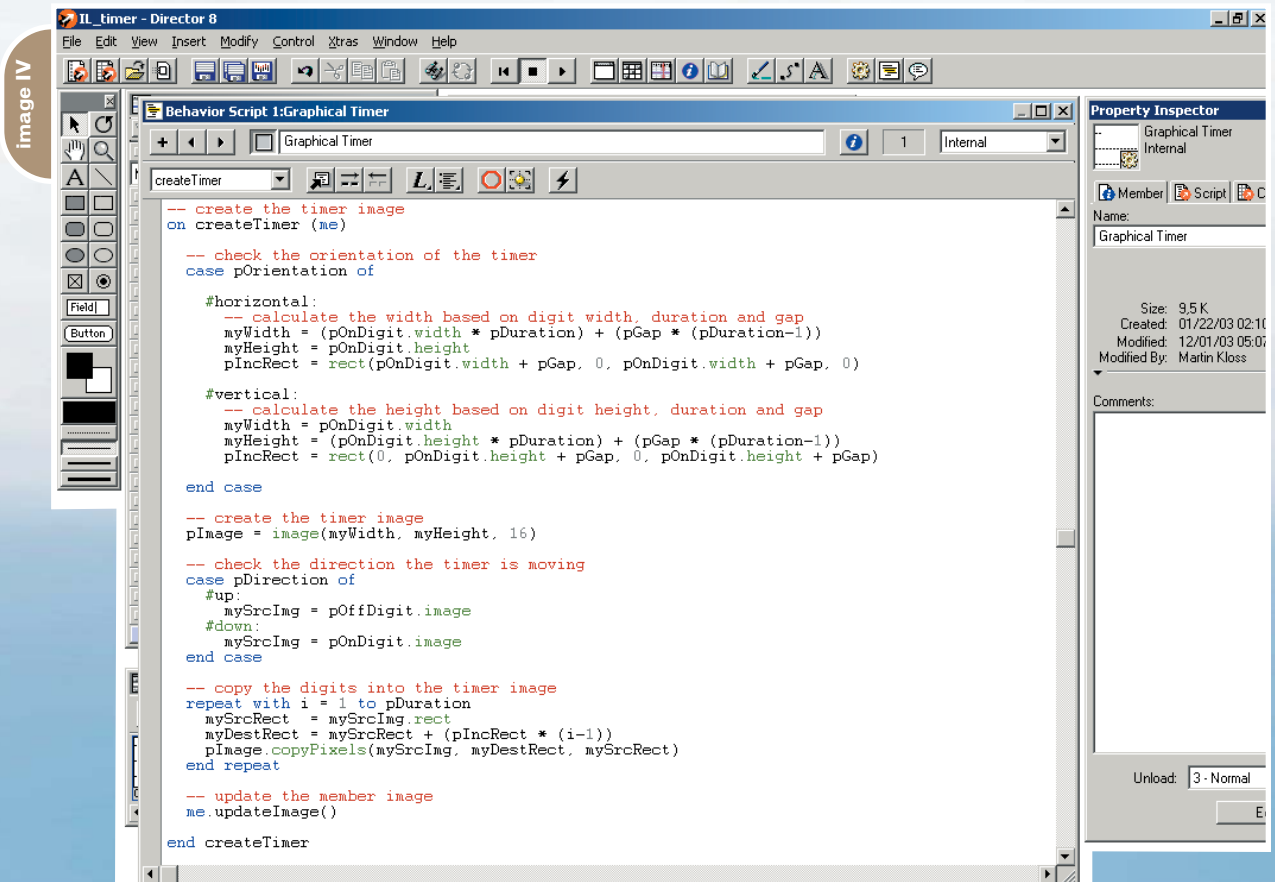
The Goal

Now that we have our tools, let’s define what we’ll build using them. In many projects, especially games, you can see some sort of a graphical timer that displays the amount of time passed or remaining. For example, in a game you might see a timer bar on top that shows the time left for you to shoot all of your enemies, or in an online learning application you might see the time passed since you started the session.

These time displays make a nice addition to many applications. They’re much nicer than simply displaying numbers and they can be used in a broad variety of colors or shapes. But as a requirement, since we’re talking about displaying something as constant as time, they need to indicate the passing of time independent of the movie’s frame tempo. While scientists could well argue that time indeed isn’t constant at all, we’ll just assume it is because we’re going to create a multimedia application, not the navigation system for the next Mars mission.

Timeout objects send their messages based on the interval they were given when they were created. The frame tempo of your movie does not influence their behavior at all, so you can have your movie running at 1 or 30 frames per second and the timeout object will send a message every second if you tell it to do so.

One important thing to know about timeout objects is that if you specify very short intervals, such as 1 or 10 milliseconds, Director in some cases might



not be able to send out a message at every interval because other tasks, like rendering the stage, might take too much time, so there's not enough processing time left to send out the message. In that case the message will be dropped and, hopefully, at the next interval a message will be sent. So if you know there are many things to be animated on the stage, set the timeout interval to something Director can manage. In most cases you will not need such very short intervals anyway, so don't worry about it.

The Behavior

We're going to create a behavior that uses Imaging Lingo and Timeout objects to display a graphical timer bar based on input defined in the behavior's property settings dialog. The cool thing about this behavior will be that by changing the behavior's properties you can create an unlimited amount of different timer displays. Curious? Let's start.

We'll begin by defining the behavior's properties. Take a look at the comments describing the properties; most of it should be pretty self-explanatory, and the rest will be made clear in the process.

Properties

- **property pSprite:** Timer sprite
- **property pMem:** Member to draw the image in
- **property pOrigImage:** Original image of the member
- **property pImage:** Current image of

the timer

- **property pDuration:** Duration of timer in seconds
- **property pCurrentTime:** Elapsed time in seconds
- **property pInterval:** Timer update interval
- **property pOnDigit:** Image "on" digit
- **property pOnDigitColor:** On-digit color
- **property pOnDigitWidth:** On-digit width
- **property pOnDigitHeight:** On-digit height
- **property pOffDigit:** Image "off" digit
- **property pOffDigitColor:** Off-digit color
- **property pOffDigitWidth:** Off-digit width
- **property pOffDigitHeight:** Off-digit height
- **property pIncRect:** Rect of increment image
- **property pGap:** Gap between the digits
- **property pOrientation:** Orientation (horizontal/vertical)
- **property pDirection:** Direction of movement (up/down)
- **property pAutoStart:** Auto start
- **property pCall:** Lingo call when finished
- **property pTimer:** Timeout object

Some of these properties will be set through the property setting dialog. (If you don't know what the getProperty-DescriptionList handler does, please read up on it in the Lingo dictionary and

Director manuals.)

Now that all of the properties are defined and we know how most of them are set, let's start with the internal events and how our sprite reacts to them (see Image III).

The beginSprite handler is always a great place to do all or most of the initializations of your behaviors since it is called automatically when the playback enters the frame where the sprite is started. The same goes for the endSprite handler, since this event is also automatically called when the playback head leaves the sprite.

The sprite initialization here is really simple. In order to save some typing work, we store a reference to the sprite object and its member in properties. In order to be able to restore the image of the sprite's member, we copy the current image using the duplicate() function. (Note: Using member(n).image always returns a reference to that image, so any manipulation of that image will be reflected in the original cast member.)

The timer bar will be constructed by two images, an "on" and an "off" digit. Each digit will have four properties: color, width, height, image. These properties are stored in a property list to make it easier to access them. The color, width, and height of each digit will be set through the property description dialog (see above); the image will be assigned in a minute, so for now we'll just set it to 0.

After the properties are set, the handler calls two custom initialization han-

"behaviors

are one of the most powerful

tools in Director because they allow you to create generic code snippets that can be utilized in many different situations and projects"



dlers. The first one, `initDigits()`, creates the images of the "on" and "off" digits; the second, `initTimer()` handler, creates the timer bar image and starts the display. Let's look at these two handlers:

```
-- init the digits
on initDigits (me)

    -- create "on" digit
    me.createDigit(pOnDigit)

    -- create "off" digit
    me.createDigit(pOffDigit)

end initDigits
```

This handler calls another function to create each digit (see Image IV). The function expects the property list with the information about color, width, and height of the timer digit; it creates a new image in the given dimensions, fills it with the desired color, and stores it in the "image" property of the list:

```
-- create a digit image
on createDigit (me, myProps)
```

```
-- create image and fill with given
color
    myImg = image(myProps.width,
myProps.height, 16)
    myImg.fill(myImg.rect,
myProps.color)
    myProps.image = myImg

end createDigit
```

The `initTimer()` handler is very similar. It calls a function to create the timer bar image, resets the `pCurrentTime` property, and if needed, starts the display of the timer (see Code II).

The creation of the timer is based on three things: the orientation of the timer (horizontal, vertical), the direction of the movement of the timer animation (up, down), and the images of the digits we just created (see Code III).

The timer orientation determines the total width and height of the display, based on the width and height of the digits and the amount of time set in the properties. Depending on the direction in which the animation moves, the "on" or "off" digits will then be copied into the

timer image. After the image is created we just need to update the member's image so the timer will be displayed in the sprite on the stage:

```
-- update the member image
on updateImage (me)

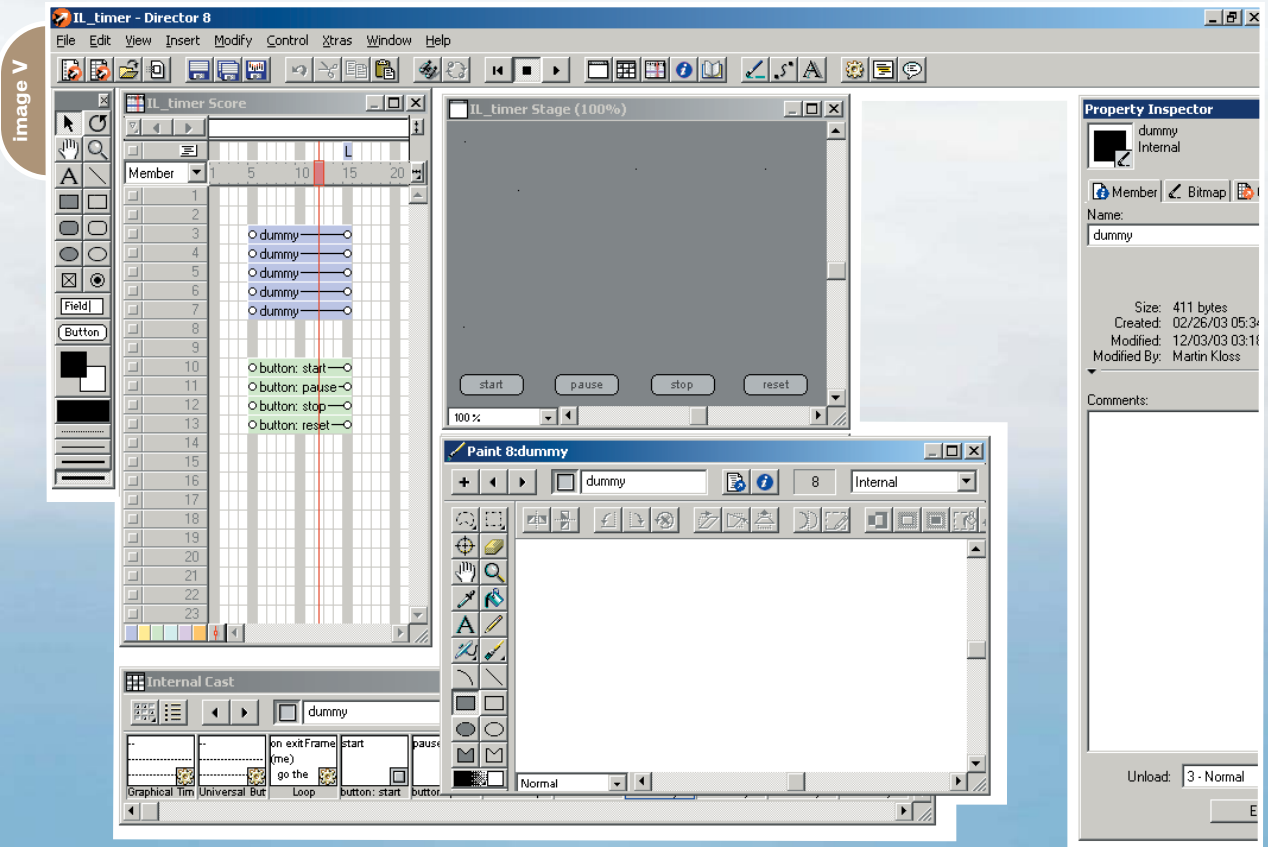
    pMem.image = pImage
    --pMem.centerRegPoint = TRUE
    pMem.regpoint = point(0,0)

end updateImage
```

Depending on how you want the timer to be aligned with other elements on the stage, you can set the `regpoint` to `point(0,0)` or center it. It's really a personal choice and something that can be changed by uncommenting a single line, so don't worry about it at this point.

Hard to believe, but that's almost it. We really have only one final handler left to build to manipulate the display of the timer (see Code IV).

This handler draws the image of a single digit ("on" or "off") into the timer



FREE* CD! (\$198.00 VALUE!)

Secrets of the ColdFusion Masters

Every *CFDJ* Article on One CD!



— The Complete Works —

CD is edited by *CFDJ* Editor-in-Chief Robert Diamond and organized into 23 chapters containing more than 450 exclusive *CFDJ* articles!

All in an easy-to-navigate HTML format! **BONUS: Full source code included!**

ORDER AT WWW.SYS-CON.COM/FREECD

**PLUS \$9.95 SHIPPING AND PROCESSING (U.S. ONLY)*

©COPYRIGHT 2004 SYS-CON MEDIA. WHILE SUPPLIES LAST. OFFER SUBJECT TO CHANGE WITHOUT NOTICE. ALL BRAND AND PRODUCT NAMES ARE TRADE NAMES, SERVICE MARKS OR TRADEMARKS OF THEIR RESPECTIVE COMPANIES.



Only from the World's Leading i-Technology Publisher



image, so it's basically our animation handler. The direction property defines which way the timer animates; the elapsed time property helps us find the location of the current digit in the timer image. After the image is altered, we just need to call the `updateImage ()` handler again and the changes will be reflected on the stage.

All that is left now is to write a couple of handlers to control the timer from the outside. That's where the `Timeout` object we mentioned earlier comes into play (see Code V).

Before we create a new timeout object, we check whether there already is a timeout object so we wouldn't have to create a new one. The timeout object we create will call the `mStepTimer ()` handler; using the "me" keyword we tell the timeout to look for the handler in this behavior instance. The calls interval was set through the behavior description dialog. If there already is a timeout object, we activate it by setting its "period" property manually.

Everything that has a start also has an

end, so we not only want to be able to start the timer, but also to pause or to stop it:

```
-- pause the timer
on mPauseTimer (me)
  if objectP(pTimer) then
    -- pause the timeout object
    pTimer.period = 0
  end if
```

end mPauseTimer

This handler simply sets the timeout object's "period" property to 0 and keeps it from calling the handler it's told to call. This is a neat trick to pause a timeout object without having to destroy the object altogether.

This is done in the `mStopTimer` handler. The timeout object's internal "forget()" method is called to deconstruct the timeout object and the property is set to `VOID` to ensure that no reference is left.

-- stop the timer

```
on mStopTimer (me)
  if objectP(pTimer) then
    -- forget the timeout and reset
    the property
    pTimer.forget()
    pTimer = VOID
  end if
end mStopTimer
```

In some cases you might want not only to start, pause, and stop the timer, but also to reset it, so a running timer is stopped and restarted. That can be done by combining two already existing functions:

```
-- reset the timer
on mResetTimer (me)

  -- make sure the timer is stopped
  me.mStopTimer()

  -- re-initialize the timer
  me.initTimer()

end mResetTimer
```

code I

```
on beginSprite (me)
  -- store sprite and member
  pSprite = sprite(me.spritenum)
  pMem = pSprite.member
  -- store original image of the member
  pOrigImage = pMem.image.duplicate()
  -- create property list with on-digit information
  pOnDigit = [#color: pOnDigitColor, #width: pOnDigitWidth,
#height: pOnDigitHeight, #image: 0]
  -- create property list with off-digit information
  pOffDigit = [#color: pOffDigitColor, #width:
pOffDigitWidth, #height: pOffDigitHeight, #image: 0]
  -- initialize the timer digits
  me.initDigits()
  -- initialize the timer
  me.initTimer()
end beginSprite

on endSprite (me)
  -- restore original image of the member
  pMem.image = pOrigImage
end endSprite
```

code II

```
-- init the timer display
on initTimer (me)

  -- create timer image
  me.createTimer()

  -- reset elapsed time
  pCurrentTime = 0

  -- start timer if autostart flag is true
  if pAutoStart then
    me.mStartTimer()
  end if

end initTimer
```

code III

```
-- create the timer image
on createTimer (me)

  -- check the orientation of the timer
  case pOrientation of

    #horizontal:
      -- calculate the width based on digit width, duration
      and gap
      myWidth = (pOnDigit.width * pDuration) + (pGap *
(pDuration-1))
      myHeight = pOnDigit.height
      pIncrRect = rect(pOnDigit.width + pGap, 0,
pOnDigit.width + pGap, 0)

    #vertical:
      -- calculate the height based on digit height, dura-
      tion and gap
      myWidth = pOnDigit.width
      myHeight = (pOnDigit.height * pDuration) + (pGap *
(pDuration-1))
      pIncrRect = rect(0, pOnDigit.height + pGap, 0,
pOnDigit.height + pGap)

  end case

  -- create the timer image
  pImage = image(myWidth, myHeight, 16)

  -- check the direction the timer is moving
  case pDirection of
    #up:
      mySrcImg = pOffDigit.image
    #down:
      mySrcImg = pOnDigit.image
  end case

  -- copy the digits into the timer image
  repeat with i = 1 to pDuration
    mySrcRect = mySrcImg.rect
```

See how nice it is to create reusable code? Not only does it save you a huge amount of work, it also enables you to build new functionalities with already existing code.

Okay, we're almost done now. If you're still with me, you probably already know what's missing. I'm talking about the mStepTimer handler we used in the creation of the timeout object to call the animation functions of our behavior (see Code VI).

The handler increments the internal counter pCurrentTime and calls the drawTimerStep () handler to draw the next digit into the timer image. If the timer has finished with its animation, the mStopTimer () handler is called to stop the timer and destroy the timeout object. If there is a Lingo function you want to call after the timer has done its job, it will be called using the "do" command.

Now you can control the behavior from anywhere in your movie by simply calling these handlers using sendAllSprites or sendSprite if you have

more than one timer display on the stage:

```
sendAllSprites(#mStartTimer)
sendAllSprites(#mPauseTimer)
sendAllSprites(#mStopTimer)
sendAllSprites(#mResetTimer)
```

Conclusion


That's it, you did it. It doesn't look like much and it really isn't (see Image V). The real beauty of these few lines of code will be revealed once you start using the behavior. Create a new bitmap cast member, insert a 1x1 pixel dot as a dummy image, and drag the cast member to the stage. Assign the behavior to the sprite, set the properties, and start the movie. After playing with the properties for a while, you will see how many variations of timer displays are possible with this simple behavior. You can create more cast members, have 10 or more different timers on the stage at the same time, and control them with a single call.

To give you a small idea of what you can do to extend the functionality of this

behavior, here's a handler to set the image of the "on" or "off" digits of the behavior from the outside:

```
-- set digit image
on mSetDigit (me, myDigit, myImage)
  case myDigit of
    #on:
      if ilk(myImage, #image) then
        pOnDigit.image = myImage
      end if
    #off:
      if ilk(myImage, #image) then
        pOffDigit.image = myImage
      end if
  end case
end mSetDigit
```

You can use existing images of your application or game for the timer display or even change the digits while the animation is running. The number of possible extensions is really up to your imagination.

You can view and download a demo movie, including the source files, at www.sys-con.com/mx/sourcecec.cfm. 

Martin Kloss is a freelance author, musician, programmer, and consultant based in Hamburg, Germany. He has been working in multimedia since 1994 and started his first company in 1996. He is also the founder and leader of the MMUG-D/LingoPark (Macromedia User Group Germany) at www.lingopark.de. In early 2003, he started his current company, selling|sound, producing royalty free music for multimedia productions (www.selling-sound.com) martin.kloss@gmx.de

code IV

```
myDestRect = mySrcRect + (pIncRect * (i-1))
pImage.copyPixels(mySrcImg, myDestRect, mySrcRect)
end repeat

-- update the member image
me.updateImage()

end createTimer

-- draw a single step of the timer
on drawTimerStep (me)

  -- check the direction the timer is moving
  case pDirection of
    #up:
      mySrcImg = pOnDigit.image
      mySrcRect = mySrcImg.rect
      myDestRect = mySrcRect + (pIncRect * (pCurrentTime - 1))
    #down:
      mySrcImg = pOffDigit.image
      mySrcRect = mySrcImg.rect
      myDestRect = mySrcRect + (pIncRect * (pDuration - pCurrentTime))
  end case

  -- copy the new digit into the timer image
  pImage.copyPixels(mySrcImg, myDestRect, mySrcRect)

  -- update the member image
  me.updateImage()

end drawTimerStep
```

code V

```
-- start the timer
on mStartTimer (me)
```

```
-- check if there is already a timeout object
if NOT(objectP(pTimer)) then
  -- create a new timeout object to animate the timer
  pTimer = timeout("Graphical Timer - Sprite " & me.spritenum).new(pInterval, #mStepTimer, me)
else
  -- start a paused timer
  pTimer.period = pInterval
end if

end mStartTimer
```

code VI

```
-- timer animation step
on mStepTimer (me)

  -- increment elapsed time
  pCurrentTime = pCurrentTime + 1

  -- draw current step
  me.drawTimerStep()

  -- check if animation has finished
  if pCurrentTime >= pDuration then

    -- stop the timer
    me.mStopTimer()

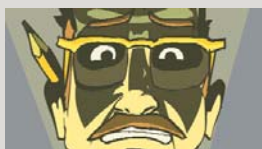
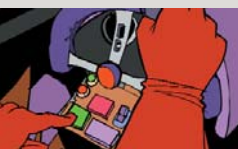
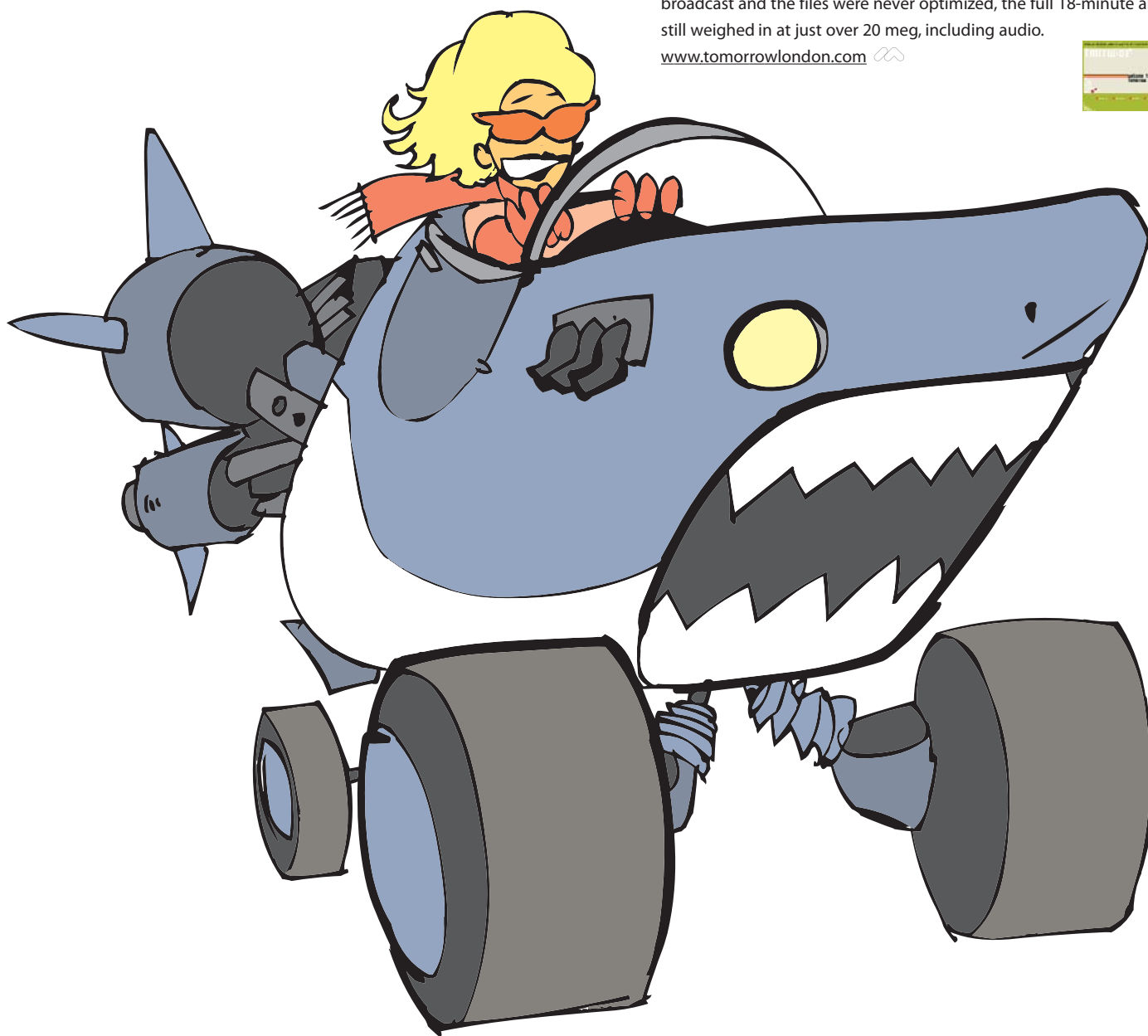
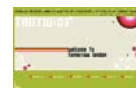
    -- call Lingo function
    if pCall <> "" then
      do pCall
    end if
  end if

end mStepTimer
```


Dangerville

dangerville, designed by Alec East and Julian Parry of Tomorrow London, is a live-action Children's TV series whose director wanted to have one episode in animation. 3D would take too long to build and render, as would Adobe After Effects. The only practical answer was to use Flash. All of the character design and key frame elements were done in pencil on layout pads, penned in, then scanned and imported into Flash MX and the "trace bitmap" applied. This retained the hand-drawn quality and made it easy to fill in the color areas in Flash. Although it was created for broadcast and the files were never optimized, the full 18-minute animation still weighed in at just over 20 meg, including audio.

www.tomorrowlondon.com



live wireless.

work wireless.

be wireless.



The most important
technology event
of the year!

CTIA WIRELESS 2004

is the one show where wireless standards are created and the technological direction of the industry is set. With the largest gathering of wireless engineers and technologists, this is where you will find the tools you need to help build and advance the wireless industry.

Look at all CTIA WIRELESS 2004 has to offer the wireless engineer and technologist:

- 6 CTIA educational sessions dedicated to exploring wireless technology
- IEEE Wireless Communications Network Conference (WCNC) 2004 – the industry's foremost conference for developing wireless standards and engineering
- WiFi Summit – the CTIA Smart Pass program, a cutting edge look at WiFi strategy and security
- A 400,000 square foot exhibit floor displaying the latest in wireless technology and applications



welcome the wireless generation.

March 22-24, 2004

Georgia World Congress Center

Atlanta, GA, USA

www.ctiashow.com

John T. Chambers
President & CEO
Cisco Systems



Scott McNealy
Chairman & CEO
Sun Microsystems, Inc.




Russell Simmons
Chairman & CEO, Rush Communications,
Co-founder & Chairman, Def Jam Records





Consumer
Products



Music



Fighting AIDS



Water

INTO

WHAT ARE YOU INTO?

At Macromedia, we're continually inspired by the passion of our customers. Visit "Into" to see their stories, or share your own. www.macromedia.com/into



Announcing Macromedia MX 2004:
New versions of Macromedia® Flash™, Dreamweaver®,
Fireworks® and Studio. [Run with it.](#)

Copyright © 2003 Macromedia, Inc. and its licensors. All rights reserved. Macromedia, the Macromedia logo, Dreamweaver, Flash, Fireworks, and Macromedia Flash are trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries. Other marks are the properties of their respective owners.