FLASH&FLEX

DEVELOPER'S MAGAZINE

Vol.3 No.6 Monthly Issue 6/2010 (14) July ISSN 1898 -9136

WHAT HTML5 MEANS FOR FLASH

Flash Player 3D is Coming Soon!

MONETIZING YOUR WEB GAME

SMARTFOXSERVER FOR MULTIPLAYER GAMES

LIVECYCLE DEVELOPMENT

ENTITY DATA INTERCHANGE FORMAT

SOURCEMATE:AN ACTIONSCRIPTER'S TOOLBOX

sourcemate BETA
for Flash Builder 4

USING FLASH CAPABILITIES IN FLEX APPS
THE ULTIMATE CHECKBOX LIST PATTERN
BUILDING EXPERT SYSTEMS IN FLASH WITH EXSYS CORVID

INFLUXIS

RELEASE THE INTERACTIVE BEAST

Application Hosting Services for the Adobe Flash Media Interactive Server

Adobe

STREAMING
Shred the restraints of traditional streaming video and free your inner beast to build the wildest apps.

COLLABORATION
Build any collaborative app you can imagine - social theatres, user recorded video, instant live data, and more.

EXPERTS
Our FMS global network has been growing since 2002. We know FMS and can help you achieve app greatness.

Basic plans from
$9.95/mo

Advanced plans from
$220/mo

Dedicated plans from
$895/mo

Custom plans from
$6/mo

INFLUXIS: The Flash Media Server Specialists **www.influxis.com**

Editor's Note

Dear Readers!

The recent introduction of the new Apple iPad has stirred the discussion over the future of web content and application runtime formats, and shed light onto the political and business battles emerging between Apple, Adobe and Google. These discussions are often highly polarized and irrational. Developers of Rich Internet Applications and Content have been blogging and buzzing about HTML5 and whether it is a Flash Killer technology since Steve Jobs published his Thoughts On Flash letter in April. His position explaining why Flash would not be supported on Apple iPhones, iPads and iPods is explained, but if we readers think Steve is being forthright in helping us understand the wisdom of his decision, then think again...more explanation and thougths on the Special Report on page 10.

This month, we have a great selection of articles, but let's start from tools that we recommend for every Flash/Flex Developer – SourceMate: An ActionScripter's Toolbox – a fantastic tool. Everyone should try it, but before you do, please read Louis DiCarro article on page 14.

In The Game section, you will find Chris Hughes' article on how to Monetize Your Web Game really worth reading – so you could make great money on your knowledge.

Besides that, learn how to use Flash Capatbilities in Flex apps, see what is The Ultimate Checkbox List Pattern, and much more inside! Let's not waste your time – let's skip to the great August issue content.

We look forward to sharing more information with the community in the coming weeks and months. If you have questions, comments, or concerns, please don't hesitate to contact me directly, or ask our authors about any details.

I want to thank you for your continued commitment to the FFD mag community, and I look forward to new opportunities to work together.

With best regards,.

Ewa Samulska
ewa.samulska@ffdmag.com

FLASH&FLEX

Editor in Chief: Ewa Samulska ewa.samulska@ffdmag.com
Proofreaders: Betsy Irvine, Patrick French

DTP Team: Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl
Art Director: Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Senior Consultant/Publisher: Paweł Marciniak

Publisher: Software Press Sp. z o.o. SK
ul. Bokserska 1 02-682 Warszawa Poland Worldwide Publishing

Software Press Sp. z o.o. SK is looking for partners from all over the World.
If you are interested in cooperating with us,
please contact us by e-mail: *cooperation@software.com.pl*

Thanks to the most active and helping beta testers:
Russell TangChoon, Lee Graham, Jassa Amir Lang, Ed Werzyn, Yann Smith-Kielland, Justus, Csomák Gábor, Kevin Martin, Charles Wong, Ali Raza, Almog Koren, Izcoatl Armando Estanol Fuentes, Lionel Low, Michael J. Iriarte, Paula R. Mould, Rosarin Adulseranee, Sidney de Koning

To create graphs and diagrams we used smartdraw.com program by SmartDraw company.

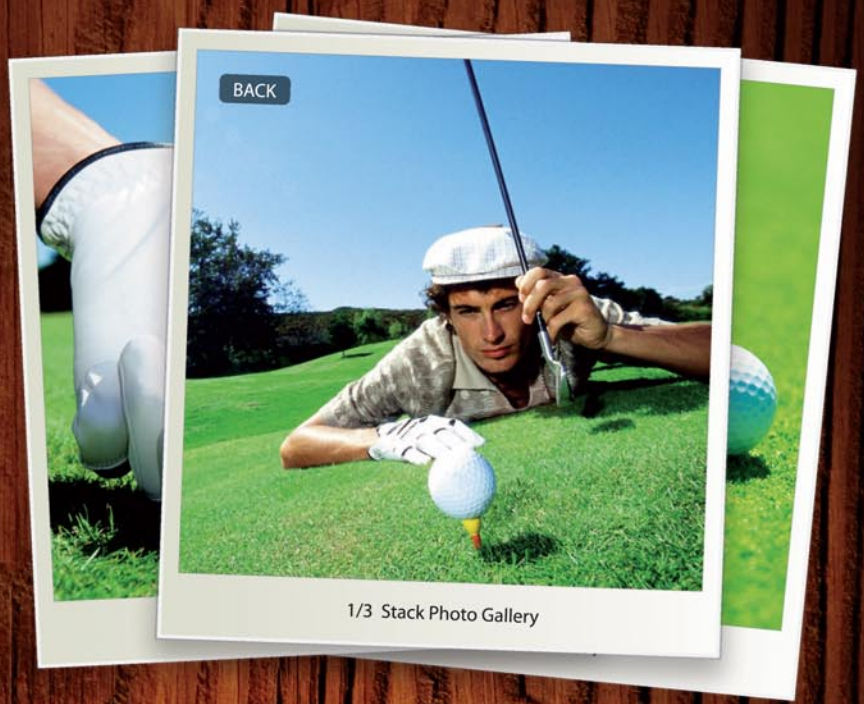
The editors use automatic DTP system ARPUS
Mathematical formulas created by Design Science MathType™

PROFESSIONAL FLASH GALLERIES
COLLECTION

www.**flash-gallery**.com

Stack Photo Gallery

Natural looking tool emulating a set of photos
– captions for photos
– dynamic shadows



Art Flash Gallery

Fully customisable and flexible flash gallery
– text descriptions for images
– fullscreen mode

Zen Flash Gallery

Flash gallery with incredible 3D rotation effect
– folder hierarchy support
– slideshow mode

PhotoFlow Flash Gallery

Famous streamline photo flow effect
– customizable interfase
– dynamic reflections

COMMON FEATURES

XML Support

Flexible Configuration

Various Transition Effects

Complete Docs

Trial Version

FOR MORE INFORMATION VISIT **flash-gallery.com**

CONTENTS

Tip of the issue

Flash Error with Flickering Tweens

Tom Rassweiler – Manager of Game Development

Sometimes when a flash movie is compiled some of the animated tweens will become corrupted in the .swf. This is a very strange problem and results in animations seeming to flicker, displaying only the *keyframes* and not the tweened frames. This is made more difficult because the compiler does not give you any warning of this corruption. You will only see it when you play the movie.

The bug is easily fixed by going into the library of the fla and opening the corrupted movie clip, and then recompiling. No actual adjustment of the code is required.

However, it is still time consuming and irritating, so it is possible to create a JSFL command that will go through the process of opening and looking at each movie clip before compiling. Using this preventative measure, you can avoid this error entirely.

```
doc = fl.getDocumentDOM();
lib = doc.library;
it = lib.items;
for(var i in it){
            if(it[i].itemType == „movie clip”){
                  lib.editItem(it[i].name);
      }
}
```

Adobe Solution PARTNER

DIGICRAFTS

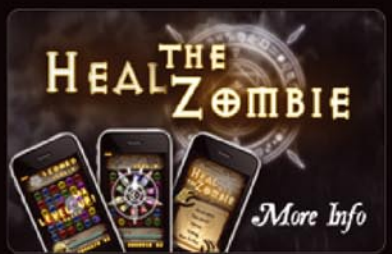
DockMenu 3D NEW

3D Menu Flash Component

Visit us for more info

www.digicrafts.com.hk/components

We also do iPhone game
www.digicrafts.com.hk/iphone



Dw Dreamweaver Extensions

LR Lightroom Plug-in

Fl Flash Components

Fx Flex Components

More products available on **digicrafts.com.hk/components**

Follow us on Twitter **twitter.com/digicrafts** and **Facebook**

Flash Player 3D

is Coming Soon!

Well just when we thought there couldn't be much more excitement after the launch of CS5, AIR Android & Google TV... Thibault Imbert, Product Manager at Adobe, released a blog post titled, Flash Player 3D Future session at Max 2010 (*http://www.bytearray.org/?p=1836*). In this post, he hints at many improvements for supporting 3D in Flash including better textured z-buffered triangles and GPU acceleration.

While the details are very hush hush until Adobe MAX, I have learned the following:

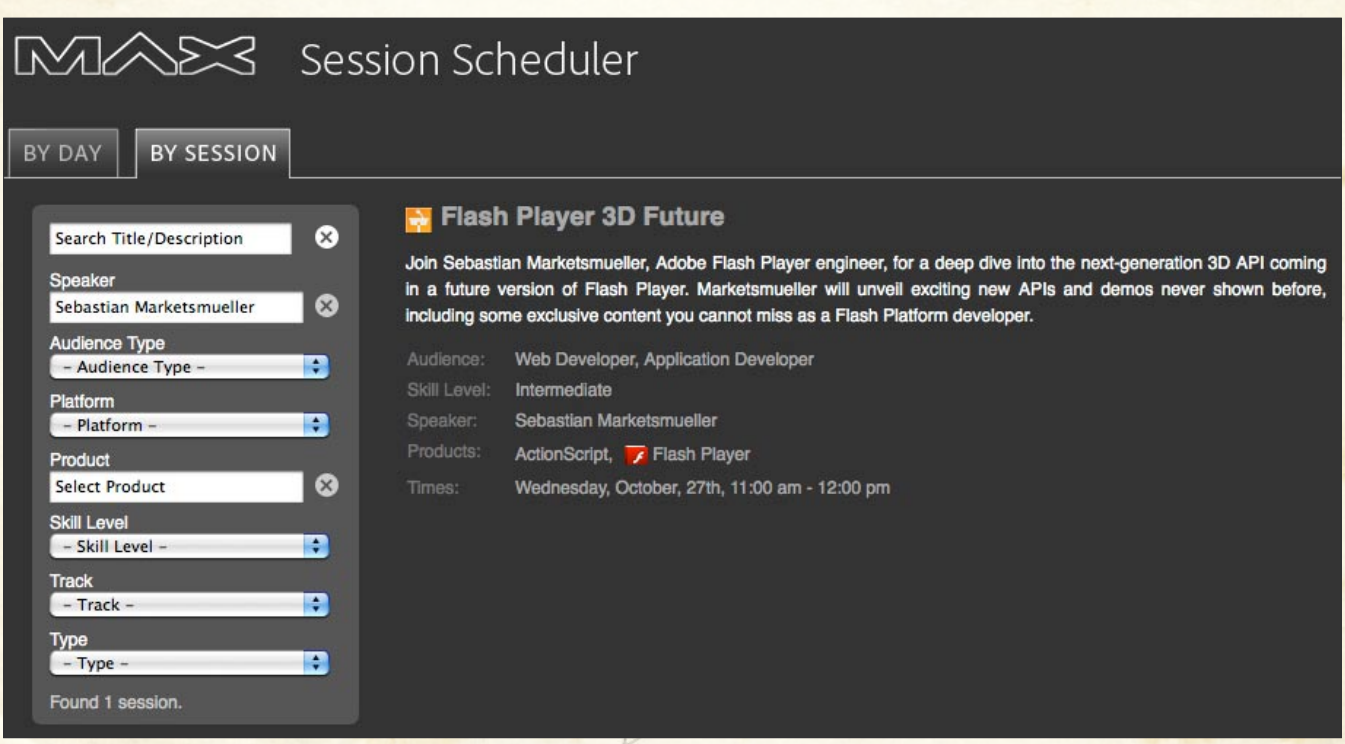
- There is a next generation Flash 3D API coming
- It will work across ALL devices that support Flash Player such as PCs, Android mobiles/tablets and AIR.
- They are saying it is nothing like you've seen before!

If I had to guess, I'm thinking that this new Flash 3D API is going to put things like Papervision 3D to shame.

So if you can head out to Adobe MAX October 23-27 in Los Angeles (*http://max.adobe.com/*) or stay tuned to Adobe.tv (*http://tv.adobe.com/*) for more information. If you are flip-flopping between attending MAX or not, check out all of the great Sessions (*http://max.adobe.com/schedule/by-day/*) that will be happening over the 5 day period!

LEE GRAHAM

Lee Graham is co-founder of TRImagination (http://trimagination.posterous.com), an educational app company based in the United States. He has been involved in developing interactive eLearning applications for five years and working with Adobe in beta testing Flash CS5, AIR 2.0, AIR for Android & Flash Player 10.1. You can connect with him on Twitter: http://twitter.com/donaldleegraham or his Blog: http://l33.me/.



```
/*
ask this question of prospective students
*/
```

What do you want to learn today?

```
/*
this tells 'em what we do
*/
```

Learn Flash, Flex, Contribute, DreamWeaver, XML, XSLT, JavaScript, XHTML, CSS and ColdFusion from an Adobe certified instructor, who:

- Specializes in training beginners to advanced learners
- Conducts corporate on-site training
- Provides consulting services

```
/*
let others sing your praises
*/
```

"It's rare to find an instructor who has personality and the ability to teach complex subjects. Kevin is great!"
–Kelley Sullivan, Power Integrations

KR KEVINRUSE + ASSOCIATES

www.kevinruse.com	001.408.496.6846	kevin@kevinruse.com	lstevens@kevinruse.com

The Minority Report

With the now infamous letter from Steve Jobs bashing Adobe Flash and supposedly explaining why Apple will not allow Flash content on its i-everything devices, developers steeped in Adobe Flash and Flex technologies have to be pondering the future of their craft. Fortunately, for us, Mr. Jobs' letter isn't the last word in this already far-too-long saga of tech industry titans.

When considering the useful life of applications and content, developers and authors must rely more on the motivation to create it than the underlying technologies supporting it.

Ignoring HTML5 and the H.264 Video Codec would be kin to trying to write web applications with COBOL, it's something we just aren't going to do. Fortunately Adobe integrates with both well.

When Steven Spielberg cast Tom Cruise as John Anderton, head of the precrime department of the police in Washington D.C. 2054, he probably didn't realize that the technology portrayed as futuristic in his 2002 production, *Minority Report*, would arrive just eight years after its debut onscreen. And while Spielberg's aim was to entertain, his visionary clairvoyance is helping define the future of Adobe Flex and Flash Technologies.

While the movie contains many portrayals of computing technology, two particularly poignant scenes are when Cruise waves a gloved hand in front of a video wall to drag-and-drop video clips and play-pause-resume the visions of precogs, and when he walks swiftly through the mall of the future trying to ignore video walls that advertise Lexus and Guinness and even call his name as he passes. The kiosk of the future not only recognized who John Anderton was but it also tailored the ad content to his demographic and historical record as a consumer.

What does this have to do with Flash and Flex? You ask... the answer is *everything*. Let's back up.

Developers of Rich Internet Applications and Content have been blogging and buzzing about HTML5 and whether it is a Flash Killer technology since Steve Jobs published his *Thoughts On Flash* letter in April. His position explaining why Flash would not be supported on Apple iPhones, iPads and iPods is explained, but if we readers think Steve is being forthright in helping us understand the wisdom of his decision, then think again. What Jobs doesn't explain in his now infamous letter is his announcement of a new mobile advertising platform called iAd on April 8th.

It would appear that blocking all of the *New York Times'* ads on his new iPad reader isn't to protect your battery life, but instead to give him a way to sell air, and I don't mean Adobe Air! An article published by Forbes on April 20, 2010 titled *Great Expectations* notes that while iPad sales account for only 4% of Apple's revenue, the announcement of the iAd service caused Apple's stock price to increase by 8%. Consider the fact that when the more than one million iPad users pull up publications on their iPads with Flash-based advertising, what they see isn't an ad at all, instead they see a white-out section of the page where the ad would normally appear and a shiny blue heart in the middle of it. I can almost see Jobs smirking from his Palo Alto office every time I see the heart. I am surprised he didn't take a bite out of the now all-to-familiar icon.

But Steve isn't the only one whose recent actions we might consider when choosing our future development platforms. Another key player in our saga is Ian Hickson, the

editor of the HTML5 draft specification. Ian's paycheck comes from advertising as well. His employer, Google Inc. is on track to earn as much as $28 billion in revenue this year and some report as much as 99% of it will be from advertising revenues. Forget search engines and web browsers, Google has shown all of us that the future of the Internet will be the same as television's past. It will be determined by commercials and sponsors not developers and authors. Kind of makes me feel very insignificant as I write mxml every day for my seemingly oh-so-small clients, huh?

But before we fold up our netbooks and head to the remote islands of Fiji we all should take pause and consider what this means to anyone with programming experience. And by *programming* I don't mean the producers of the next NCIS episode. The question we must ask is if Google derives its revenue from advertising, and Apple is headed in the same direction, what do these vendors reveal to us through their acquisitions and new initiatives and business models? Consider that while some of Google's acquisitions are companies such as AdMob, Teracent and Invite Media, all online advertising companies, other companies purchased include technology acquisitions such as On2, the owners of the VP8 video codec and Metaweb and PlinkArt, innovators in search engine technologies.

If we consider where Google spends its time and money we can grasp what the most influential technology company in the world thinks we should be doing as we develop our measly applications from one day to another. And no I don't mean world domination! Before *Antennagate* Apple's market capitalization exceeded Microsoft's for a few weeks. And while these now decades old companies are valued in excess of $200 billion by their shareholders, consider that Google is not far behind at a value of over $150 billion. Google is more influential because they are more pervasive and have gained their dominance in a shorter period of time. Remember that Google started in 1998 and went public in 2004, two years after Spielberg's film.

And what does Google say about Flash? They ride the fence. Their Android OS now found on vendor's smart phones everywhere supports both Flash and HTML5. Their YouTube site plays H.264, as well as Flash Video and many other codecs. While Google is intent on being part of shaping future standards like Ian's HTML5 specification, they in practice are technology agnostic. They have no religion or bigotry. They love everyone. Maybe that should be the vendor with the heart icon.

But Google's indifference to technology wars may prove that they are in fact the biggest driver of change. By doing nothing, they watch from the sidelines as formidable titans duke out their seemingly petty differences about html tags, namespaces and codecs. It reminds me of Neville Chamberlain's position on Germany and Italy's politics prior to World War II. And in the realm of cyberspace the coming sea change will be just as disruptive to Internet commerce as real war is to international trade.

As more dollars flow into companies selling applications and content stemmed by traditional companies' desire to sell everything from diapers to cheese snacks, more dollars are available to those of us who program. And whether today we write Flex applications for a manufacturing company's Intranet or code ads for a publisher of GeoSpatial search engine optimization, it makes no difference. We live in a growing economy. In spite of nations going bankrupt and government bailouts, those of us privileged enough to be skilled in creating something from nothing and selling air or helping others sell air have bright futures.

Apple will derive revenue from vertical integration similar to how Rockefeller made oil money after giving away kerosene lamps for free. The air they will sell is the invisible host of the cellular signals and much like the air that carried RF signals to your NTSC television sets until the ATSC standard was introduced in 2009, the air that Google, Apple and others sell will ultimately host products such as Adobe Air as well. The reason is best exemplified by a program written by Chris Teso that demonstrates how a webcam image can be captured and analyzed in real time tracking his finger as it selects and moves objects across his screen.

In Teso's video demo of the Actionscript program he shows how Spielberg's glove isn't even necessary. Why Teso used ActionScript instead of Java is simple, speed. It is the same reason authors in the future will continue to use ActionScript and Flash instead of an HTML5 Canvas and JavaScript. While Java boasts of performance and speed improvements, it simply won't suffice the challenges interactive video requires. In the grand scheme of things when one considers all of the issues, touch screens are really not an issue, now are they?

Useful Links:

- Steve Jobs' letter – *http://www.apple.com/hotnews/thoughts-on-flash/*
- The HTML5 Specification Draft at W3C – *http://dev.w3.org/html5/spec/Overview.html*
- Chris Teso's ActionScript Demo – *http://www.christeso.com/blog/index.php/lab/minority-report-actionscript-webcam-interface/*
- Current demos of HTML5 – *http://html5demos.com/*

JOHN MARX

John Marx is General Manager of Dayspring Systems and Webspinner.com

Flash Builder for Force.com Available

The new offering integrates the Flash Platform with *Force.com* (from salesforce.com) to bring the richness of the consumer Web to enterprise cloud applications. This jointly developed IDE provides a single, powerful tool for building cloud-based RIAs, which can easily be deployed to end users through the browser using Adobe Flash Player or directly to the desktop via Adobe AIR. Developers can use Adobe Flash Builder for Force.com to extend or enhance existing salesforce CRM implementations and custom-built Force.com applications, or build entirely new applications to meet any demanding business need.

Source: Adobe Flash Platform Blog

Monetize your AIR applications with Melrose

Melrose – the monetization service previously known as Shibuya – is now live on Adobe Labs.

With just a couple lines of code you can add a complete license manager and payment solution to your AIR application. You can even create time and feature based trials.

Melrose works exactly like all the other application stores out there. Adobe don't charge any monthly or yearly fees but do take a percentage of your earnings (obviously there is no charge for free apps). In return they'll provide a safe and secure way to monetize your AIR applications. That means you don't have to worry about things like credit card processing and hosting your license manager logic. Your private dashboard will tell you how your application is doing. You can view charts on number of downloads of your apps, number of trials, number of purchases, revenue, number of activations, and trial to purchase histogram (i.e. conversions from trials to purchases).

Developers and publishers in 47 countries can use Melrose to distribute and monetize AIR applications.

Source: Serge Jespers

Flerry 1.1.2 released

Flerry is a Flex-Java bridge for Adobe AIR 2.0, is now at version 1.1.2. It is only a bugfix release.

Source: Adobe Flash Platform Blog

More Adobe Open Source with SourceForge

Duane „Chaos" Nickull got an email from Dave McAllister, and I've selected some paragraphs from it: *Today, we formally launched phase 1 of open@adobe, our new portal for open stuff. Open@Adobe (http://sourceforge.net/adobe) is the first instantiation of SourceForge's new developer platform. Open@Adobe is a site aggregating Adobe's openness programs, which includes source code hosting, such as the Adobe® Flex framework, and contributions from Adobe to standards organizations, as well as specifications.*

It has been clear for some time that we needed to become more aligned with development principles is seen in open source projects. We've always followed an open process model, with exposure to our bug bases, open discussion forums, roadmaps for products, and early access through Adobe Labs. However, our current repository was not meeting the desire to allow our projects to evolve in multiple directions simultaneously.

In short, Source Forge was beginning to redesign their forge. Because of this we had the chance to add our requests, such as the ability to link back to existing Adobe properties (like forums) and in the future, adding the ability to unify the bugbase to our engineering teams. We worked with a number of existing projects to get their input, and it was fascinating the input we did get.

We wanted to tap the creative and innovative energies of the open source community. SF today hosts over 250K projects and has around 3M downloads a day. We've seen some increase in our numbers since its been up, even though it wasn't announced to today. We wanted the ability for groups and individuals to self create projects; and in the near future to recognize the existence of projects built on our open source core technologies. There are planned updates during the next year, and the various projects will move from opensource.adobe.com to open@adobe as it makes sense.

from: Adobe Flash Platform Blog

Cell Your Game Contest

GameLicense.com Present: Cell Your Game Contest Maybe you've heard the news: the newest Android mobile OS fully supports Flash. Guess what that means? Now's the time to break into a brand new market for Flash games! Adobe and *FlashGameLicense.com* are proud to host a very special contest exclusively for members of FGL. Create or port a game for mobile Flash platforms and you could win fabulous prizes – not to mention open a whole new world of players and profit. There are over $30,000 in cash and prizes and 150 games will win at least $100. Go on, check it out! FGL has also created a new forum full of guides, FAQs, and support to help you take advantage of this new frontier. Adobe is providing them with the latest guidelines, APIs, and tools, and FGL will make them available to you there.

from: http://www.flashgamelicense.com/sponsor_pages/adobe/

News selected by Gábor Csomák

Add complex interactive decision-making logic to your Flash applications.

Integrate your Flash apps with the proven knowledge automation expert system Inference Engine used by thousands of businesses including Fortune 100 companies, government agencies and the military. Go beyond graphics and multimedia to a unique complex probabilistic problem-solving analytical core!

Build systems that dynamically emulate a back-and-forth conversation with an expert to provide detailed, user-specific recommendations and advice for diagnostics, support, pre-sales recommendations, regulatory compliance and many other decision-support tasks.

How…your Flash application posts data to the Exsys Corvid® Runtime, a Java-based Servlet program, which uses heuristic rules for analysis. That sends XML back to your Flash application for user follow-up questions, to display results and recommendations, or perform other tasks. Systems can have a single or multiple SWF files. The underlying rule logic is developed in the Corvid Development environment - an easy to learn tool with a 26-year track record of handling complex rule-based decision support logic. All of the Action Script needed for the Servlet interactions is provided with Corvid and easily plugged into your Flash applications.

Download a free 30-day demo version of Corvid with tutorials - get your first systems built in just a few hours. Corvid systems can be delivered with Flash; or Corvid's Java applet and HTML based servlet runtimes.

For more information and links to sample systems go to: http://www.exsys.com/flash.html

Powered By

exsys corvid
Expertise at Your Fingertips™

EXSYS®
www.exsys.com

SourceMate: An ActionScripter's Toolbox

I first learned about SourceMate by ElementRiver (http://www.elementriver.com/sourcemate) during a presentation of Flash Builder 4 at our local Flash Users group. The presentation was put on by Lee Brimelow and he said it was a must-have tool to go with Flash Builder. I immediately looked into the tool and found it invaluable to every day coding situations.

SourceMate is a plug-in for the Eclipse environment whether you are using Flash Builder or the Flex plug-in for Eclipse. Installation is easy using the *Install new software...* command within the environments and you can be up and running with SourceMate very quickly.

The program adds a menu-item to your Flash Builder install that gives a wide variety of assistants to code writing. It also adds options to the code assistant pop-up that can be used while coding. These options include refactoring, event generation, code extraction and code templates among other features.

Using these added features surpass Flash Builder's built in functions and make coding faster and easier. Whether your code uses basic OOP patterns on a small project or larger enterprise projects that follow strict designs, SourceMate can help make the process easier.

Code generation

What I like most about SourceMate and use the most is the code generation capabilities. Code can be generated on the fly from either the code assist pop up or the template view from within the program. Most of the redundant tasks such as for loops, function generation and event handlers can be created with ease.

Code generation is not a bunch of text that has been pasted in from an external template. It is live type and allows you to continue to work with the functions without having to move the cursor to the position where you want to edit.

For example, the event handler template starts by typing in the `addEventListener` command followed by the first parameter for the event type. After typing the comma to separate the parameters, bring up the code assist pop up (*Command or Control Space*). The first option will be *Generate Event Handler*. Selecting this option will complete the line with a generic function name and create an event handler function with the same name.

Now this is where SourceMate makes coding life easier for the user. The generated handler name will be highlighted, so to change the name of the function, just start typing. Not only will the name of the handler function change in the `addEventListener` line, but also in the generated function itself.

But there is more! After typing the name of the handler function, hit the tab key and the parameter type of the function is highlighted so you can change it. Once that is changed, hit tab again and you will be in the function body ready to start typing in the behaviours of the event. All without having to type out the basic frame work of the function. It is also possible to use code generation on plenty of pre-built templates such as for loops that are hooked to arrays, forin loops that are added to dynamic objects and constants with the same functionality and ease of use.

It is also possible to create new snippets and templates customised to your individual needs. If I find that I am checking for the same variable a lot, I will create a snippet that generates an if statement. It save me only a few seconds, but those seconds quickly adds up.

Custom snippets can also be shared with other users. This makes a good template system for a group of developers to keep their code consistent. Templates can also be created for custom libraries that may not have snippets already created for them.

Refactoring

If you are like me, you often code out your ideas. When those ideas end up working out, you want to use them in the actual project. The problem is that there is a lot of unnecessary repeated code that can be turned into a single function. Tracking down all of these variable and function names can be difficult, especially when there are multiple classes in the project.

This is where refactoring comes in and SourceMate is a definite improvement over the native Flash Builder

implementations. SourceMate can take highlighted code and create new methods, constants from values and interfaces from classes. The process is simple, straightforward and, most importantly, allows you to preview the changes to implement them.

If there are several lines of code that you find have been repeated throughout the class, you can turn these into a function by highlighting one of the blocks and selecting *Extract Method...* from the menu. A window will pop up and allow you to name the function. It will also look at what variables are being used in the lines of code and set those as parameters for the functions.

Most importantly, there is a preview button included in the window. Clicking on this button will show the original code next to the proposed changes in a diff style format. This allows review of the changes before they are applied. One problem I found using the *Extract Method* command is that it will allow multiple functions with the same name to be generated. This is easily avoidable if you use the preview command before committing the changes, but it is something to be conscious of.

Other useful tools include Constant extraction and generation, which allows you to convert a hard value to a constant. SourceMate will find all instances of that value and convert it to the new constant. *Extract Interface* allows a class to be converted to an interface class. The command will allow you to select which methods will be part of the interface and make the original class implement the new interface.

And much, much more...

There are a lot of user friendly features in SourceMate that will really help you get your projects done faster. While the code generation features alone make it worth installing, all the other features just keep adding to experience. Features such as ASDoc generation, TODO markers and ANT build file generation adds a lot of power to Flash Builder and makes tasks that were once tedious a lot easier.

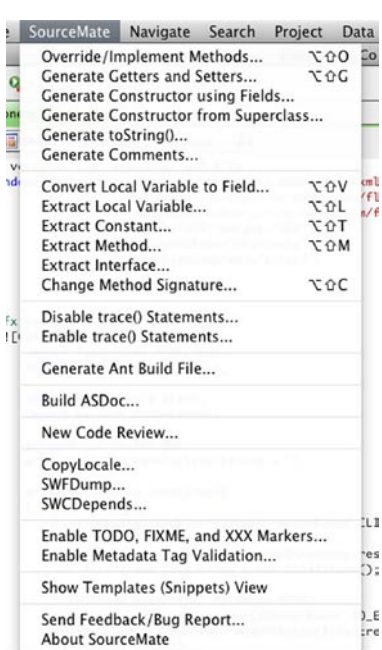


Figure 1. *The SourceMate menu in Flash Builder*

An extra feature that I particularly like is the Metadata Content Assistant. When there is a lot of external resources to embed in a project, the [Embed] tag can be a pain to implement repeatedly. By typing the opening bracket, the content assistant will open and allow you to choose a metadata tag, which will then insert the tag and its parameters.

Trace statements are usually sprinkled throughout a project's code and used for debugging purposes. When a project needs to be presented but is not necessarily finished, you don't want to show the trace statements but don't want to remove them either. The *Disable All* `trace()` *Statements* will search through your code and comment out the traces. To make the traces available again, select *Enable All* `trace()` *Statements* to turn them back on.

Conclusion

Once you install SourceMate and add it to your toolbox, you will wonder how you have lived without it. There are so many features that will fit so many different coding styles, almost everyone can find it useful.

Flash Builder is a powerful project but many developers who come from a programming background find some of the common features that are available in other IDEs missing. SourceMate works to replace and enhance those features. Every feature it adds is well thought our and works efficiently to get the task done.

A lot of tools that are available as add-ons to Flash Builder also seem to be geared toward the enterprise level projects and the more advanced user. What I really like about SourceMate is that you do not have to be an advanced user to use it. There may be some features that a user may not use, but it is still a worthy investment regardless of skill level. Getting up to speed with the product is quick and painless. There are not a lot of archaic commands that need to be learned before the program can be used effectively. Simply install it and you should be able to use the code generation right from the start. But it is also not restrictive in how it is used.

Ultimately, I agree with Lee. SourceMate is worth the investment and needs to be in your programming toolbox. It is very affordable and easy to use which will save you a lot of time in building your projects. It will also allow developers to concentrate on creative coding instead of redundant tasks.

LOUIS DICARRO

Louis DiCarro is a consultant based in NYC and has been working with Flash since the first version. He has taught web development at the college level and has worked for numerous large clients. He can be reached at louis.dicarro.ffd@gmail.com.

ENTITY

Data Interchange Evolved

Finding the parent of a Flex 3 control is easy, but the parentage of a Flex 4 control is harder to pin down. In this article we look at some ways of tracking down the visible parents of Spark components.

ENTITY is a lightweight data-interchange format. That is easy and fast for machines to parse and generate. It is based on conventions used in programming languages like LISP, JAVA, and C.

ENTITY is a text based format that is completely language independent, making it a robust and versatile data-interchange format. ENTITY is smaller in size and faster to parse/generate than both XML and JSON. It

Listing 1. *list's a simple FLEX program that take's a FLEX object and serializes it into entity text*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/
                    mxml" layout="absolute" creationC
                    omplete="init();">
        <mx:Script>
            <![CDATA[
        import entity.*;
        import mx.controls.Alert;

        private function init():void
        {
             var eObj:Object = new Object();
             var entArr:Array = new Array();

             /*
                   Build up a simple object.
             */

             for(var loop:int = 0; loop < 10; loop++)
             {
                  var currObj:Object = new Object();

        currObj.name = "name " + loop.toString();
        currObj.value = loop.toString();
        entArr.push(currObj);
                        }

            eObj.description = "a simple encoding test";
             eObj.array = entArr;

                        /*
                             Encode the object into ENTITY.
                        */

                        var entStr:String = Entity.encode(eObj);
                    }
           ]]>
               </mx:Script>
</mx:Application>
```

Listing 2. *Shows the encoded data that should be stored in the entStr variable after execution of the program in Listing 1*

```
(
        description:a simple encoding test,
        array:[
                        (name:name 0,value:0),
                        (name:name 1,value:1),
                        (name:name 2,value:2),
                        (name:name 3,value:3),
                        (name:name 4,value:4),
                        (name:name 5,value:5),
                        (name:name 6,value:6),
                        (name:name 7,value:7),
                        (name:name 8,value:8),
                        (name:name 9,value:9)
                    ]
)
```

represents exactly the same data as XML and JSON with fewer characters. The official entity libraries are free for non-profit use. The specification is open for all to use in the development of their own ENTITY encoders/decoders. I would encourage anyone creating their own parser to send me an email so that I can list it on the entity website at *http://www.entity-format.co.uk* for others to use.

Why Use Entity?

Entity is smaller than both XML and JSON and the author of the ENTITY format also wrote the world's fastest JSON PARSER JSwoof. ENTITY was actually created because I believed that the capability limits of the JSON format had been reached. It was time to stop creating optimized code to speed up the format and look at creating a new format conceived to be small and fast.

Tell Me More

ENTITY can represent four primitive types (Strings, Numbers, Booleans, & NULL) and two structured types (Objects, & Arrays) both of which can be found in most modern programming languages in some shape or form.

All of the official ENTITY encoders/decoders can digest JSON text streams, making it easy to incorporate ENTITY into existing systems.

It is fair to say that ENTITY is a young data interchange format, but it is going from strength to strength. At the time of writing this article ENTITY was only two weeks old and is already available on FLEX, C-SHARP, JAVA and VISUAL BASIC. There are many more official and third party Open Source ports on the way.

Listing 3. *Lists a simple program that takes some entity text and decodes it into a native FLEX object*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/
                    mxml" layout="absolute" creationC
                    omplete="init();">

    <mx:Script>
        <![CDATA[
  import entity.*;
  import mx.controls.Alert;

  private function init():void
  {
        var simpleENTITY:String = "(name:
                    wayne,value:#1000.0)";

        var eObj:Object = Entity.decode(simpleENTI
                    TY);
  }

        ]]>
       </mx:Script>
</mx:Application>
```

Listing 4. *Shows the encoded data from ENTITY and JSON for a single customer record. As can be seen ENTITY is much smaller whilst still representing the exact same data as JSON*

```
        ENTITY Format (295 Characters)

          (
              firstName:John,
              lastName:Smith,
             age: #25,
             blocked: ?f,
             address:
             (
                 streetAddress:21 2nd Street,
                 city:New York,
                 state:NY,
                 postalCode:10021
             ),
             phoneNumber:
             [
                 (type:home, number:212 555-1234),
                 (type:fax, number:646 555-4567)
             ]
          )

        JSON Format (359 Characters)

{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "blocked": false,
    "address":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber":
    [
       { "type": "home", "number": "212 555-
          1234" },
       { "type": "fax", "number": "646 555-
          4567" }
    ]
}
```

Listing 5. *List's a simple program that takes JSON data*

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                    layout="absolute" creationComplete="init();">
     <mx:Script>
          <![CDATA[
        import entity.*;
        import mx.controls.Alert;
        import mx.core.ByteArrayAsset;

       [Embed(source='../html-template/example.json',mimeType='applicat
                    ion/octet-
                stream')] [Bindable] public static var jsonTextClass:
                    Class;

       private function init():void
       {
           var jsonByteArray:ByteArrayAsset = ByteArrayAsset(new
                    jsonTextClass());
           var jsonText:String = jsonByteArray.readUTFBytes(jsonByteArr
                    ay.length);

           var jObj:* = Entity.decodeJSON(jsonText);     //decode JSON
                    text.

           if(jObj != null)
           {
               var entityStr:String = Entity.encode(jObj); //encode
                    object into ENTITY.

               Result.text = "ENTITY:\n" + entityStr + "\n\n";
               Result.text += "JSON:\n" + jsonText;
           }
       }
   ]]>
     </mx:Script>
     <mx:TextArea right="10" left="10" top="10" bottom="10" id="Result"
                    editable="false"/>
</mx:Application>
```

The Entity Library

The library is made up of eight classes EERROR.AS (Error Handling Functions), ESERIALIZE.AS (Object Serialization Functions), ETOKEN.AS (ENTITY Token Types), JTOKEN.AS (JSON Token Types), ETOKENIZER.AS (ENTITY Tokenizing Functions), JTOKENIZER.AS (JSON Tokenizing Functions) EMISC.AS (Miscellaneous Functions), and ENTITY.AS (Decoding And Encoding Functions). You can find full documentation of the ENTITY library on the ENTITY website. *http://www.entity-format.co.uk*

Creating Our First Entity Text Stream

Ok, we know what the entity format is all about. So let's create our first entity text stream see (Listing 1 and Listing 2).

Ok so we can encode entity. All we need to know now is how to decode the data we have created see (Listing 3).

Simple isn't it. There are three functions that are publically available in the ENTITY class that you will need for the encoding and decoding of data they are: `decode()`, `encode()`, and `decodeJSON()`

Making The Switch

Ok so you already use JSON. Why should you bother switching to ENTITY? Well, it is often said A picture is worth a thousand words'. So is some example output data from both formats see (Listing 4).

Porting

Ok, so you have some existing JSON data you would like to convert to ENTITY. If you are using one of the official ENTITY libraries this can be done with a single call. Convert's it into a FLEX object, and then encodes it into ENTITY.

It really is that simple; to start shrinking the data you pass around your applications and services.

Over And Out

You will see ENTITY available on a lot more platforms over the coming months. And you will can always be sure to find the latest versions of ENTITY at *www.entity-format.co.uk*

For the latest ENTITY news please follow the project on Twitter: *http://twitter.com/entity_format*

WAYNE IV MIKE

The author has worked in the games industry since 1999, initially programming for the GAMEBOY Color and PLAYSTATION one. He currently works for one of Britain's biggest regional newspapers as an application programmer, using C, C++, PHP and FLEX.

Wayne's spare time is spent writing open source applications and graphical demonstrations, designed to push PC hardware to the limit.

You can contact Wayne directly via email: wayne@entity-format.co.uk

KINDISOFT

Protect Your ActionScript 1/2/3 from All SWF Decompilers.

Protect your ActionScript 3 with secureSWF v3

secureSWF continues to be the most sophisticated ActionScript obfuscation, protection, and encryption solution available today. Version 3.0 features ActionScript 3 support for Adobe's Flash, Flex and AIR applications and is available for Windows, Mac OS X and Linux.

Now Supports Flash CS4

Main Features:

- Identifiers Renaming
- Control Flow Obfuscation
- Dynamic Code Wrapping
- Statement-level Randomization
- Literal Strings Encryption
- Encrypted Domain Locking
- Encrypted Loader Creator
- SWC Files Support
- Project Files and Command-line Interface

Learn More About secureSWF and Request Your Fully-Functional Copy At:

WWW.KINDISOFT.COM

Save 15% With Promotion Code: FFDM15

Setting up model-driven

development with LiveCycle Data Services ES2

LiveCycle Data Services ES2 is an important component for many companies migrating to a three-tier architecture for enterprise RIA development.

The advanced *Message Exchange Patterns* (MEPs) it enables are unparalleled and very robust.

This article is aimed at developers and distills the essential steps needed to set up model-driven development with *LiveCycle Data Services ES2* and *Flash Builder 4 beta 2*.

In this tutorial, we'll share these steps to help you get started quickly and walk through building a simple application using a new technology that brings model-driven development to Flex developers.

Note
You may have heard this technology referred to as *Fiber* (also sometimes spelled *Fibre*). This was the internal project name and the technology is now available for public download.

Requirements

In order to make the most of this article, you need the following software and files:

Flash Builder 4 beta 2

- Download (*http://www.adobe.com/go/flashbuilder4_download*)
- Learn more (*http://www.adobe.com/go/flashbuilder4*)

LiveCycle Data Services ES2

- Try (*http://www.adobe.com/go/trylivecycle_dataservices*)
- Learn more (*http://www.adobe.com/products/livecycle/dataservices/*)

Prerequisite knowledge
Some experience developing Flex applications will be helpful.

Setting up the environment
Here are the steps you need to get started developing in Flash Builder 4 using the modeler plug-in.

Download LiveCycle Data Services ES2 and install it using the standalone LiveCycle Data Services With Tomcat option.

Note
Selecting LiveCycle Data Services Web Application to use your own application server is slightly more complicated; the standalone option makes it easier to get started quickly. Download the modeler plug-in and unzip it (*http://trials.adobe.com/pub/esd/labs/livecycle_dataservices3/livecycle_dataservices3_modelerplugin_100509.zip*).

Note
Build 100509 of the modeler plug-in is for Flash Builder 4 beta 2. There are dependencies between the plug-in and the IDE, and not all builds of the plug-in will work with all versions of Flash Builder 4.

After you unpack the ZIP file, copy all the files from its `plugins` folder into your `<Flash_Builder_Home>/plugins` directory.

Navigate to your `<LCDS_INSTALL_ROOT>/tomcat/webapps/lcds/WEB-INF/` directory and open `web.xml` with a text editor; for example, `textEdit.app` (Mac OS X) or Notepad (Windows).

Locate the section of code below (it begins around line 26): see (Listing 1).

To uncomment this code, change the first line from `<!-- begin rds` to `<!-- begin rds -->` and change the last line from `end rds -->` to `<!-- end rds -->`.

Listing 1.

```
<!-- begin rds
    <servlet>
      <servlet-name>RDSDispatchServlet</servlet-name>
      <display-name>RDSDispatchServlet</display-name>
       <servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>
       <init-param>
          <param-name>useAppserverSecurity</param-name>
          <param-value>true</param-value>
       </init-param>
       <load-on-startup>10</load-on-startup>
   </servlet>

    <servlet-mapping id="RDS_DISPATCH_MAPPING">
       <servlet-name>RDSDispatchServlet</servlet-name>
       <url-pattern>/CFIDE/main/ide.cfm</url-pattern>
   </servlet-mapping>
end rds -->
```

Listing 2.

```
<!-- begin rds -->
    <servlet>
        <servlet-name>RDSDispatchServlet</servlet-name>
        <display-name>RDSDispatchServlet</display-name>
        <servlet-class>flex.rds.server.servlet.FrontEndServlet</servlet-class>
        <init-param>
            <param-name>useAppserverSecurity</param-name>
             <param-value>false</param-value>
       </init-param>
        <load-on-startup>10</load-on-startup>
    </servlet>

   <servlet-mapping id="RDS_DISPATCH_MAPPING">
      <servlet-name>RDSDispatchServlet</servlet-name>
          <url-pattern>/CFIDE/main/ide.cfm</url-pattern>
   </servlet-mapping>
<!-- end rds -->
```

Change the `useAppserverSecurity` value from `true` to `false`. The code should now look like this: see (Listing 2).

This change removes application server security from RDSDispatchServlet so you can focus on writing code during development and not security issues. If you do not make this change, you'll see an error in Flash Builder 4 notifying you that the RDS server was successfully contacted, but your security credentials were invalid.

Note

When you have finished this tutorial, reset this parameter to true or disable RDSDispatchServlet to prevent unwanted access to the servlet, which could expose destination details on your server.

Save your changes and close the text editor. Now you're ready to start up the server. There are two components you will need to start up: a sample database and the server itself.

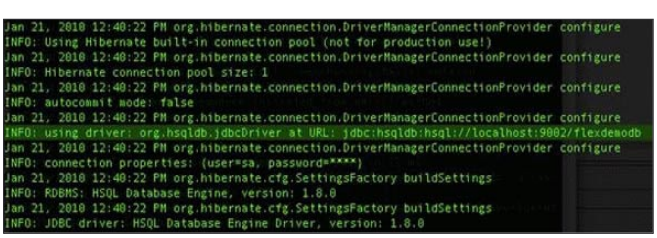


Figure 1. *Startup information showing the port that the database is using*

To start the database (an HSQLDB instance), open a terminal/command window, navigate to `<LCDS_INSTALL`

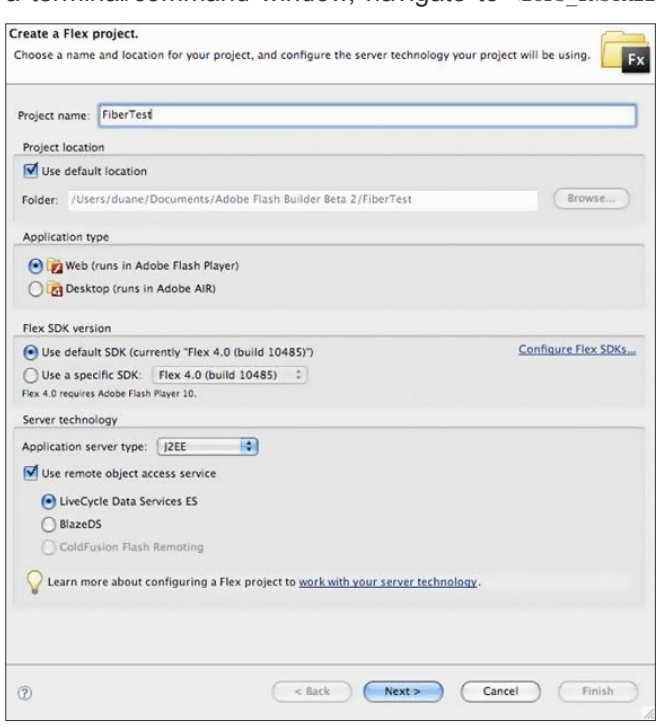


Figure 2. *Creating the new project*

Listing 3.

```
45 <Context privileged="true" antiResourceLocking="false" antiJARLocking="false" reloadable="true">
46    <!-- JOTM -->
47    <Transaction factory="org.objectweb.jotm.UserTransactionFactory" jotm.timeout="60"/>
48
49   <Resource name="jdbc/ordersDB" type="javax.sql.DataSource"
50        driverClassName="org.hsqldb.jdbcDriver"
51        maxIdle="2" maxWait="5000"
52        url="jdbc:hsqldb:hsql://localhost:9002/ordersdb"
53       username="sa" password="" maxActive="4"/>
54 </Context>
```

`ROOT>/sampledb`, and type *sh startdb.sh* (Mac OS X and Linux) or *startdb* (Windows).

Open a second terminal/command window, navigate to `<LCDS_INSTALL ROOT>/tomcat/bin`, and type *sh Catalina.sh* run (Mac OS X and Linux) or Catalina run (Windows).

As the server starts up, you will see a line that shows the port that hsqldb is using. By default it is 9002 (see Figure 1). You'll use this information to configure the data source.

Navigate to the folder `<LCDS_INSTALL ROOT>/tomcat/conf/Catalina/localhost` and open the `lcds.xml` file with a text editor.

Edit the file and add a reference to the data source as follows: see (Listing 3).

There are two databases that you can use here: *ordersdb* and *flexdemodb*. The code above uses *ordersdb*.

Save your changes and close the text editor. You may need to restart your server for this change to take effect.

Your environment is now set up and you're ready to start building your first model-driven Flex application.

Building a model-driven Flex application

Follow these steps to build a simple model-driven Flex application:

Start Flash Builder 4 and choose *File>New>Flex Project*.

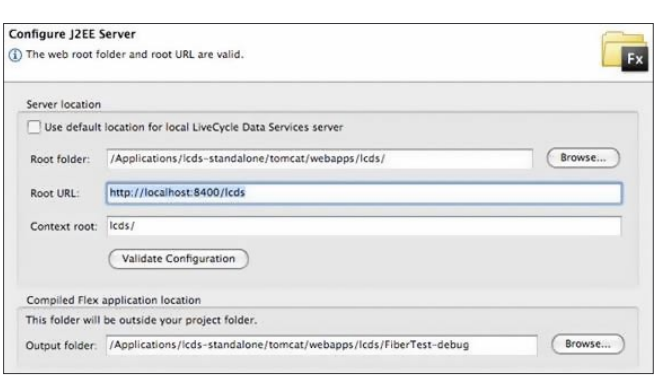


Figure 3. *Configuring the server*

Type *FiberTest* for the Project Name, select J2EE and LiveCycle Data Services ES as the server technology (see Figure 2), and click *Next*.

To configure the J2EE server, make sure the settings for Root Folder, Root URL, and Context Root are correct. By default on Mac OS X they are as follows:

Root Folder: `<TOMCAT_ROOT_DIRECTORY>/tomcat/webapps/lcds/`

Root URL: `http://localhost:8400/lcds`

Context Root: `lcds/` (Mac OS X) or `/lcds` (Windows)

Click Validate Configuration (see Figure 3) to ensure the root folder and root URL are valid and then click Finish.

Next, you must configure *Remote Data Services* (RDS) to enable the modeler plug-in to access the data source you just configured.

Choose *Window>Preferences*.

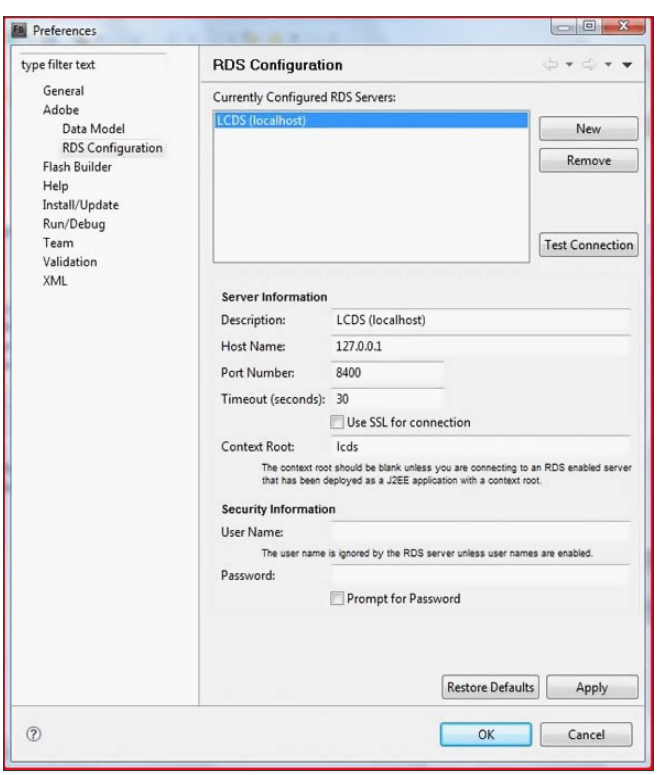


Figure 4. *Configuring RDS*

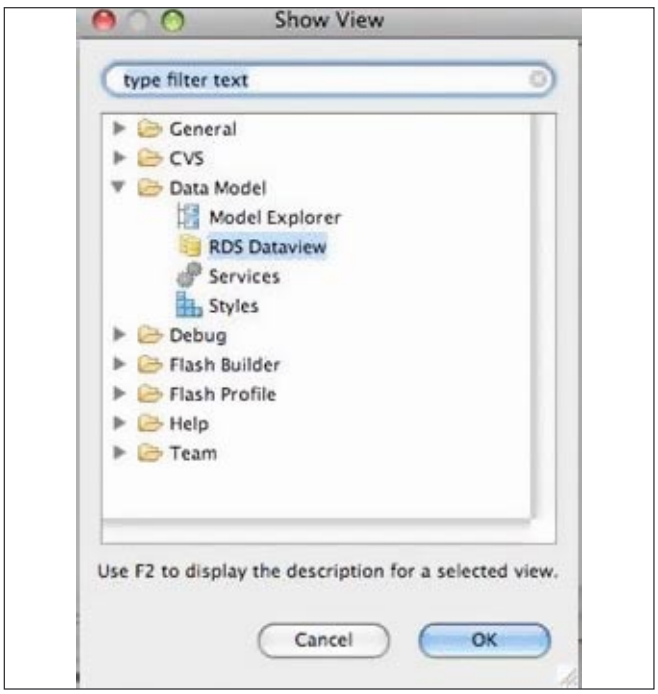


Figure 5. *Opening the RDS Dataview view*

From the list on the left, select Adobe and then RDS Configuration.

Click *New*.

For the Description type *LCDS* (*localhost*); for Host Name type *127.0.0.1*; for Port type *8400*; and for Context Root type *lcds*.

Leave the User Name and Password blank and click Test Connection to verify your settings (see Figure 4).

Click *OK.*

Choose *Window>Other Views*, expand the Data Model folder, and select RDS Dataview (see Figure 5).

If you want, you can drag the RDS Dataview panel to the lower left hand side of Flash Builder 4.

At this point, you should be able to expand LCDS (*localhost*) in the RDS Dataview panel and examine any of the database tables (see Figure

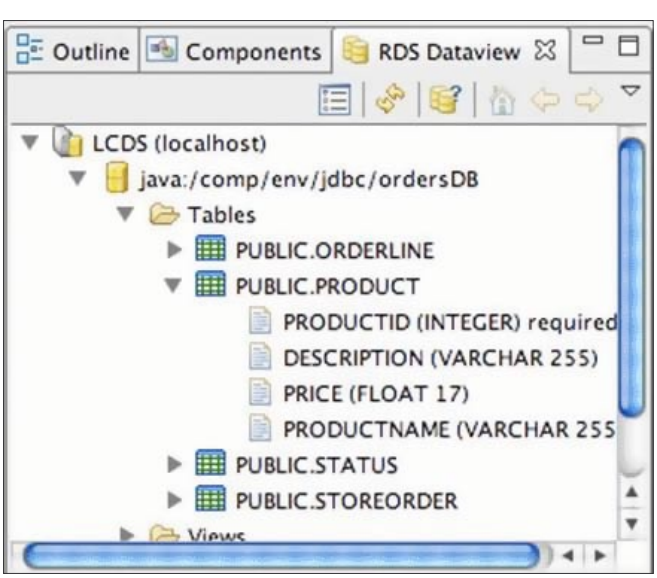


Figure 6. *Database tables shown in the RDS Dataview panel*

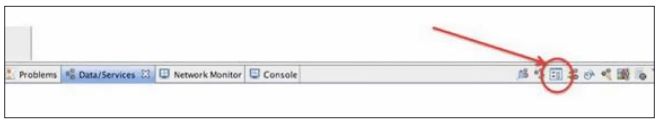


Figure 7. *The Edit Active Data Model button*

6). If you cannot connect to the server, right-click (or Control-Click) on LCDS (*localhost*), select RDS Configuration, and check your configuration settings. Common causes include security credentials (see the instructions on changing the `useAppserverSecurity` value in Setting up the environment *http://www.adobe.com/devnet/livecycle/articles/lcdses2_mdd_quickstart_02.html*) and a lack of data source mapping (see the instructions on adding a data source reference to lcds.xml in Setting up the environment *http://www.adobe.com/devnet/livecycle/articles/lcdses2_mdd_quickstart_02.html*).

To create a new FML file, switch to the Data/Services view and click the Edit Active Data Model button in the upper right (see Figure 7).

When the model editor opens, note that it supports Design view and Code view like other Flash Builder 4 editors.

Note: The modeler plug-in stores data for the Design View layout in the FML file.

Drag the PUBLIC.PRODUCT table from the RDS Dataview panel to the Design view area of the FML file (see Figure 8).

To deploy the active model to the server, click the Deploy Model to LCDS Server button (see Figure 9).

Note

You can use this same procedure to redeploy your model to the server if you later make changes to the model.

In the Deploy Data Model dialog box, type *FiberTest* (or use whatever name you gave your project), select Overwrite Existing Model, and Create/Recreate (see Figure 10). Click *Finish*.

Open.

`FiberTest.mxml` in Design view and drag a DataGrid component onto the Design area.

Click the Data/Services panel. If it is not visible, choose *Window>Data/Services*.

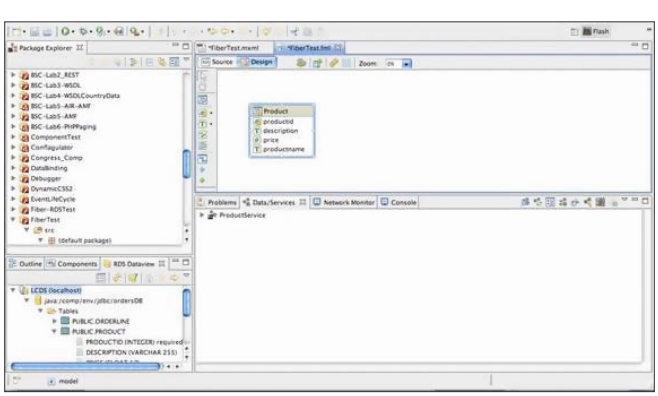


Figure 8. *The PUBLIC.PRODUCT table in Design view*

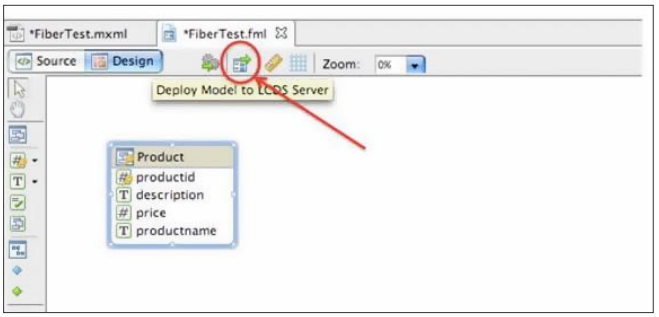


Figure 9. *The Deploy Model to LCDS Server button*

In the Data/Services panel, expand ProductService and then drag the `getAll()` method onto the DataGrid component in Design view (see Figure 11).

For each model deployed on the server, LiveCycle Data Services and Flash Builder 4 will generate methods that you can use to perform basic operations on the table represented by the model. The basic operations enable you to get all records (`getAll`), create a record (`createProduct`), update a record (`updateProduct`), and delete a record (`deleteProduct`). Apart from these methods, there are methods that you can use to filter records based on a value in a column of the table; in this case they are `getByProductname` and `getByPrice`, for example. You can also add custom methods to perform your own queries, but that is outside the scope of this article.

The DataGrid column headers will change to reflect the data returned by the call to the service. A link icon appears, indicating that the data is bound to the component.

Choose *Run>Run FiberTest* to run your project.

It should fetch data from the server and display it in the DataGrid component (see Figure 12).

Adding create, read, update, and delete operations

Of course, as a developer, you want more than just a simple display application. You're ready to add a form to enable CRUD capabilities.

Right-click (Windows) or *Control-Click* (Mac OS X) the `getAll()` method in the Data/Services tab and select Generate Form.

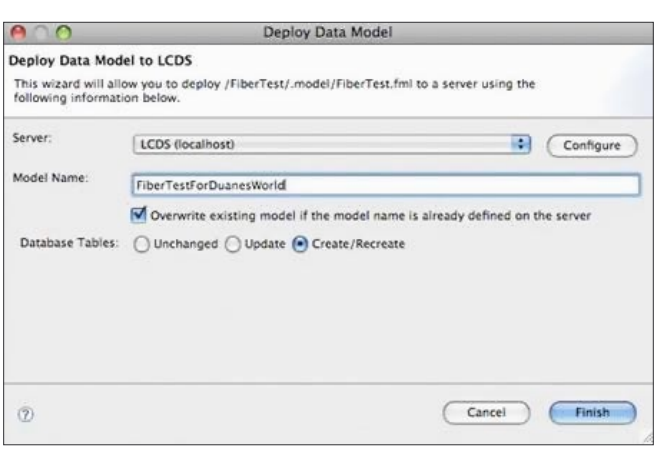


Figure 10. *Deploying the data model to LiveCycle Data Services ES2*

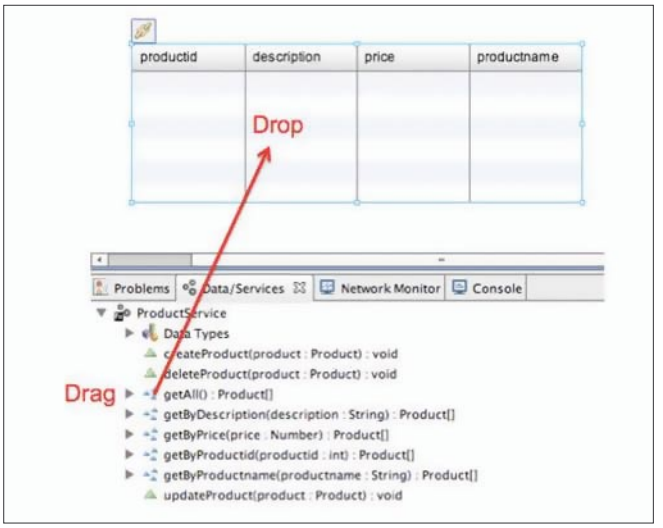


Figure 11. *Binding the getAll() method to the DataGrid component*

In the Choose *Form Type* dialog box, click *Model Driven* Form and click *OK* (see Figure 13).

After the form is generated, move it below the DataGrid component in Design view.

Save your project and run the application again.

Within the application, click Add and type some sample input in the Description, Price, and Productname fields (see Figure 14).

Click *Save* to update the database.

If you want, you can quit the application and rerun it to verify that the product you just added really was stored in the database.

Note

If you enter a non-integer price, you will notice that the price read back from the database is not exactly the same as what you entered. For example, I added a product priced at 7.3, and the price stored in the database was 7.300000190734. To understand why, examine the Price field in the RDS Dataview panel; it is defined as a FLOAT17, whereas in the actual form the price is defined as a Number. When you specify a price, save it to the database, and reload it, you are loading the FLOAT17 version of the data. This problem can be remedied fairly easily, but enumerating the steps is beyond the scope of this article. Now that you know you can create a new record, you can make a few more changes to enable update and delete functionality as well.

productid	description	price	productname
1	nice mouse	19	wireless mouse
2	slick keyboard	39	ergonomic keybc
3	best screen save	15	screen saver
4	100 cds	22	case of cds
5	flash memory	5	memory stick

Figure 12. *Live data in the DataGrid component*

Figure 13. *Choosing a model driven form*

Open `FiberTest.xml` in Source view and locate the following line (it should be near line 33):

```
<forms:ProductForm id="ProductForm1"
valueObject="{Product}" x="159" y="233">
```

Edit the line so it reads as follows (your `x` and `y` values may differ from those shown):

```
<forms:ProductForm id="ProductForm1"
valueObject="{dataGrid.selectedItem as Product}" x="159"
                    y="233">
```

This change will enable you to delete records and to make changes in the form and have them reflected in the data grid. The change is necessary because you generated the form from the data model, not from the data grid itself. As a result, the form was not automatically bound to the selected item of the data grid.

Switch to Design view and select the data grid.

In the Properties panel, set the DataGrid component's `editable` property to `true`.

This will enable you to edit data directly in the data grid, without using the form.

Save your project and run the application again.

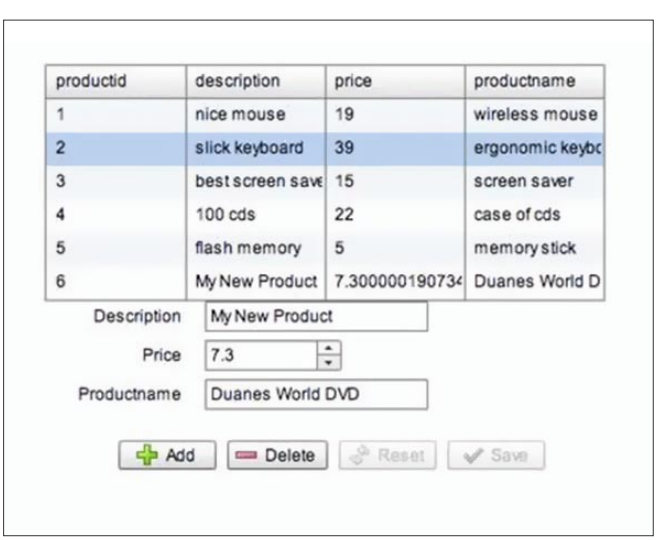


Figure 14. *Adding a new product*

Now when you select an item from the data grid, it will appear for editing in the form. Alternatively, you can edit data directly in the data grid. Any changes you make will be committed to the database when you click *Save*. Also, when an item is selected you can click *Delete* to remove it from the database.

Where to go from here

That's it. You've built your first model-driven Flex application and added basic CRUD functionality with a bare minimum of coding.

For additional reading, see Model-driven Applications in Using LiveCycle Data Services. To access the LiveCycle Data Services ES Test Drive, use your browser to open *http://localhost:8400/lcds-samples/testdrive.htm*.

Have fun unlocking the power of LiveCycle Data Services ES2 (*http://help.adobe.com/en_US/LiveCycleDataServicesES/3.0/Developing/WS31EA2385-DC66-4ecf-B256-B9BC65406ED4.html*) and Flash Builder 4.

SUJIT REDDY G

Sujit Reddy G is a Technical Evangelist for Flex at Adobe. He brings with him a great deal of expertise in Flex, J2EE and PHP. He specializes in building enterprise applications on Adobe Flash Platform and has a blog focusing on the integration of Adobe Flex with Adobe LiveCycle Data Services (and BlazeDS) at http://sujitreddyg.wordpress.com.

DUANE NICKULL

Duane Nickull, a senior technical evangelist at Adobe, is responsible for Adobe's messaging around enterprise solutions in the SOA and web services spaces plus other forward-looking aspects, such as Web 2.0. Previously Duane cofounded Yellow Dragon Software Corporation, a privately held developer of XML messaging and metadata management software, acquired by Adobe in 2003. He also served as CTO and President of XML Global Technologies, which was acquired by Xenos Group in early 2003. Visit Duane's blog.

MICHAEL PETERSON

Michael Peterson is a senior technical writer on the LiveCycle Data Services team.

The right SmartfoxServer protocol

for multiplayer games

SmartfoxServer is a popular tool to create multiplayer games. It has different communication protocols that you can use to exchange messages between the clients. In this article you will learn about each of those protocols and see how they perform in a real situation.

What you will learn…

- The SmartfoxServer communication protocols
- The amount of extra information that is added to each exchanged message
- Which protocol produces smaller and faster messages.

What you should know…

- You need a basic understanding of network communication
- Some familiarity with multiplayer games or applications able to communicate using a network

Online Flash games are quite common nowadays. It is easy for any player to find a specialized portal featuring several titles varying from single player to multiplayer gameplay. Both mechanics can be equally fun but I think the multiplayer one can give you hours of endless challenge and surprises. Beating a real opponent who is able to think the way you do is very rewarding and can be as challenging as the skills and strategies of the players involved. In order to create this type of entertainment, you must code a game which is able to communicate over some form of computer network. There are several multiplayer servers available out there, each one with different architecture and communication protocols.

SmartfoxServer

SmartfoxServer is a very powerful and flexible multiplayer server. It is written in Java, is simple to use and has a rich set of features, including a complete documentation and an active community. The server provides the developer with different tools to design and code a multiplayer game. One of them is the ability to create *extensions* which are server side code that extend the existing functionalities allowing the creation of new features and behaviors. You can create a custom extension to handle all your multiplayer game traffic so you can tweak it to the bones in order to produce a very specific and fast communication. It may reduce the network latency, but you still need to choose the best way to serialize the information before it can travel from one client to another.

Protocols

SmartfoxServer gives you three different protocols to serialize and send/receive information: XML, JSON and String-based (aka *raw protocol*).

The *XML* protocol is the default option used by SmartfoxServer and it allows the serialization of complex objects with any level of nested properties. Even though it is tempting to simply serialize a whole object and send it over the network, you must bare in mind it is extremely expensive.

The XML protocol adds a lot of extra information to the message in order to tell the server how to handle the data. For an application where messages are exchanged less often, it is not a big deal, but for real-time games it is unacceptable and will cause several network latency problems. If you can afford to choose

the protocol for your game you should always avoid the XML protocol or use it as a last resort.

The *JSON* protocol is an alternative and much faster way to serialize and send complex objects. It is based on JSON, a lightweight data exchange format which has a native compact structure.

Due to that characteristic the amount of extra information needed is smaller allowing a compression factor of 2 to 5 when compared to XML. If you must serialize complex data types then JSON is a good choice because it will not overload your messages and you don't need to write new code or drastically change your existing application to take advantage of this feature. All you need to do is to specify the message protocol as JSON when you are sending the data and SmartfoxServer will do all the rest for you.

The *String* (or raw) protocol is the fastest way to send data. It is a low level approach that gives you full control over which bytes to send and which bytes to strip out. It has little extra information packaged together with the message, so you must write your own serialization code if you want to send complex objects in the same way XML and JSON protocols do.

When designing a real-time game, every single byte transmitted counts and can potentially increase network latency problems, so reducing the amount of data is the highest priority. Using raw protocol will ensure your game is sending only what is mandatory and that the majority of the time between one message and another will be the result of physical factors such as network bandwidth, not message overloading problems.

Some numbers

Now that you know all about SmarfoxServer protocols, it's time to see them in action. The following usage information was collected from a real-time multiplayer game where two players bounce a ball on the screen. The data was recorded by intercepting the SmartfoxServer client API calls that are related to data transmission, such as the one responsible for writing the final message in the socket that is open with the server. The game used a customized serialization code so the message content could vary in time according to the object being serialized. In order to avoid this behavior and allow correct data to be collected, a constant content of 28 bytes was transmitted in all messages during the whole game session for all tree protocols.

The XML protocol was used to package private messages exchanged between the players. The messages had a total size of *113 bytes*. Since the game content being transmitted has a constant size of *28 bytes*, then *85 bytes* were not related to the game itself. Those bytes are extra information added by the server API to allow the serialization of complex objects. They represent *75%* of the message content. On the other hand the JSON messages (exchanged via extensions) had a total size of *90 bytes*. Excluding the game content there are *62 bytes* remaining, all of them being Smartfox extra information. It represents almost *54%* of the message size.

The raw protocol messages (exchanged via extensions) had a total size of *46 bytes* (18 bytes of Smartfox information). The majority of the content was related to the game and only *39%* was related to Smartfox. Even though *18 bytes* of extra information are better than the numbers of the XML and the JSON protocols, it is a relatively high footprint. You can still shorten the extension id (a string used by the server to identify the right extension to handle the message) and try to tweak the messages even more which can produce a footprint of *12 bytes*. As already explained the raw protocol produces very small messages which are better than the ones created by the XML and the JSON protocols.

Conclusion

SmartfoxServer is a powerful tool to create multiplayer games, but it will not solve all your network latency problems. Choosing the right protocol for the right situation is the key to mitigate the delay between the clients communication and ensure a The right SmartfoxServer protocol for multiplayer games.

FERNANDO BEVILACQUA

Fernando Bevilacqua is co-founder of Decadium Studios, a game development company, working with PC and web games; he is also the creator of As3GameGears.com, a site specializing in tools for Flash/Flex game developers.

Contact details: dovyski@gmail.com

Monetizing Your Web Game

Part 1

Currently there are many choices when it comes to monetizing a web game and it can be daunting for a developer to decide which model is best for him or her. On top of this, there are conflicting reports as to which models are truly lucrative.

The hope of this series of articles is to shine a light on many of the monetization methods to choose from by presenting hard facts based on case studies from a number of developers as well as statistics we have been tracking at *FlashGameLicense.com* and *GamerSafe.com*.

Part 1: Sponsorship and Licensing

Before I get into the ins and outs of licensing a web game, let me define some terms:

Sponsorship
A deal made between an entity (the sponsor) and a developer in which the sponsor pays to have their branding/ads in one of the developer's games. The terms Sponsorship and License are used interchangeably in most cases (and in all cases for the purposes of this article).

Primary License/Sponsorship
A sponsorship where the Sponsor has their branding in every copy of the game on the web except where the developer has explicitly sold a Secondary License (defined below) to another entity. The developer has complete freedom to remove the primary sponsor's branding and make any other changes to the game as long as it is licensed and locked to the other entity's domain.

Non-Exclusive License/Sponsorship
A Sponsorship where the license of the game is not exclusive to the buyer. The buyer is purchasing one custom version of the game.

Secondary License
(aka Non-Exclusive Site-Locked License)
A Sponsorship where the license of the game is not exclusive to the buyer. The buyer is purchasing one custom version of the game, and this version must be *locked* to the buyer's domain. This is the most common type of non-exclusive license and it is compatible with the primary license. The terms *non-exclusive license* and *secondary license* will be used interchangeably in this article.

Performance Bonus
A bonus paid by the Sponsor to the Developer based on pre-defined performance milestones. Bonus structures take many forms. A couple of examples are: a lump sum payout if a game gets a certain number of plays, or a CPC (cost per click) deal where the Developer is paid for each unique user sent back to the Sponsor's site.

There are many more licensing types, but these are the most popular and most important for this article. You can find a longer list of licensing types and terms here: *http://www.flashgamelicense.com/view_library.php?page=license-terms.*

At its core, the Primary Sponsorship model is simple: A Sponsor is interested in getting his or her branding into a game that will potentially be viewed and played by millions of people. In most cases, the ultimate goal is to get those users to click back to the Sponsor's site via the links in the game.

So, this is why your game is worth money to sponsors, but how do you get the most money out of the deal as possible? The trap many developers fall into is assuming that their game has a set worth to a sponsor, and that if the sponsor pays $x for the game, then they must surely be making more than $x from the game. This isn't entirely accurate. Sponsors make money by licensing games in two main ways. One, is they plan on the long term funnel of new users a game will bring them. And two, they sponsor many games in hopes that a handful will be successful. So, what this means is that they are investing in your game so that they can make their money back long term or, if that doesn't happen, that your payment will be absorbed by another – more successful – game. Of course, it is slightly more complex than that in the sense that they are getting brand association with your game and other perks like having high quality content for their site, but when thinking about how to get the most out of a deal the two main factors of long term earnings and uncertain returns should be considered the most. In short, if you can convince a sponsor that your game will have massive,

long term, appeal and that their investment is well spent, you can make more money.

Easier said than done, right? Maybe not. Having a great game is definitely most of the battle when it comes to sponsorships, but there is a lot you can add to a game to increase its worth to a sponsor (as an aside I want to mention that you should NOT release your game before you seek a sponsorship. Once a game is released *into the wild* it is virtually worthless to a sponsor). The easiest way to do this is to give players a strong incentive to click back to a Sponsor's site. This can be done by adding bonus content that is only playable on a Sponsor's site, or linking to walkthroughs on the Sponsor's site, or any number of other things. Take note, however, that you have to be extremely careful in how you execute, and present, this sort of tactic. You need to make sure that the game, on any site, is fully playable and not limited in any way. If players perceive the game as being nothing more than an advertisement, or that it is trying to trick them in some way, the game will undoubtedly be rated down and hidden on sites.

Before I continue I think it is important to point out something that will be a reoccurring theme in this series. It is important that you choose a monetization model for your games that you are comfortable with and that suites you. There are pros and cons to any path you choose. Some may have high risk, high reward. Others may be low risk low reward. Some may provide you with the creative freedom you desire, but limit you in how you can monetize the game, and so on. So, with this next bit of advice I caution you to decide what model you are most comfortable with. Based on what has been discussed this far, my recommendation is to strive to invest in the success of your game. This is usually a little bit more risky, but ultimately it drives everyone involved to push for the game's success. The reason it is more risky is because it usually means you take less up front in the deal. Here are a few examples of how you can invest in your game's success:

- Aim for a performance bonus (preferably based on CPC to the Sponsor's site).
- Include links in your game to your own, Developer, site where you have site ads.
- Aim to include Ads in the game and Microtransactions if appropriate. Implementing either or both tastefully is key.

Notice that none of these investments harms the sponsor. In fact, they should increase the value to the Sponsor as well. Services like GamerSafe and CPMStar have built in mechanics to share revenue with Sponsors. And since you make more money by making the game more successful, the Sponsor is benefiting from a very motivated developer wanting to make sure the game does as well as it can.

Another great thing about Sponsorships is that you get to benefit from the Sponsor his/herself. Sponsors want the game to do well, and in most cases they know what players like. A Sponsor's advice can be invaluable. Of course, you should never feel forced to make changes you feel will hurt the game, but you should be open to any suggestions a Sponsor may have.

Once you have accomplished these steps, the real fun starts. This is the point where you are actually pitching your game to sponsors. The easiest way to do this is at FlashGameLicense.com where we have an entire system built to assist you in this exact situation. The easiest and fastest way to find out your game's worth in Sponsors' eyes is to get them to bid on your game. Bidding wars often get heated and in the end the most motivated Sponsor ends up on top. And who else would you rather have investing in your game? And once your Primary Sponsorship deal is complete you can start to sell non-exclusive licenses. This has proven to be a great secondary source of income for the Developer, but also a harmless condition for the Sponsor to allow. The point of a non-exclusive license, for the buyer, is not to drive traffic but instead to retain traffic and acquire quality content. Most sites who buy non-exclusive licenses would never take the distributed version of the game anyway since they do not allow branding other than their own on their site. What this means is a sponsor is not losing any traffic, but the developer is able to resell their game indefinitely. Oftentimes a sponsor will ask for a short period of time after the Primary Sponsorship deal is done before you can sell non-exclusive licenses, however. This is merely an introduction to the world of sponsorships and licenses with web games, and is not meant to be all encompassing or definitive. I invite all readers to visit our site: *FlashGameLicense.com* to find out more. FGL is also THE place to buy or sell web games, so be sure to visit if you are hoping to do either or both. To summarize what has been covered here:

- Make a great game (or have it nearly finished)
- Don't release your game!
- Give players an incentive to click to the Sponsor's site
- Invest in the success of your game (if it is right for you)
- Be open to Sponsor's suggestions
- Get your game in front of as many Sponsors' eyes as possible and have them compete to get you the best deal
- Sell non-exclusive licenses

As a final note I want to stress that the Sponsorship model and all of the models that will be covered in future articles in this series are not mutually exclusive of each other. Developers can, and should, take advantage of all of them. I am merely presenting them individually as to make them less confusing to understand. The final article in this series will cover the best ways to combine as many of these monetization models as possible to maximize the revenue generated by your game.

CHRIS HUGHES

Chris Hughes – FlashGameLicense.com

Using Flash Capabilities

in Flex Apps

Being accessible via Internet, our RIA applications must provide the user with a unique experience. One of the most common ways to achieve this kind of experience is localizing the user interface. With flash.system.Capabilities we will be able to do this and many other interesting things in the application we develop.

What you will learn…

- Use Flash Capabilities to provide to localize and check system and runtime information
- Principles of Object Oriented Programming

What you should know…

- Familiar with general programming concepts such as data types, variables and functions

Using Flash Capabilities, we can achieve this and provide many other interesting functionalities in our application. In this article, I will show a sample application where we will apply the possibilities it opens to you.

Getting Started

Download the sample application at *http://code.google.com/p/flashcapabilities/* and save it in a folder of your choice. Then open Flex Builder and import the project file you just downloaded.

The Application UI

The application UI consists of a ComboBox, *TextInput*, *Button*, *TextArea* for messages and debug info panel and an *ApplicationBar*, beside the TitleWindow, etc. see (Figure 2).

The first thing we will be accomplish is localizing the user interface on the fly in four languages: *American English* and *Brazilian Portuguese* using `flash.system.Capabilities` class, if the user run the app in a system that is not using one of these two languages by default the Interface will be localized to English see (Figure 3).

FLASH CAPABILITIES PROPERTIES

`flash.system.Capabilities` provide properties that describe the system and player hosting an Application – the SWF file. Here is the list of some of the properties this class provides.

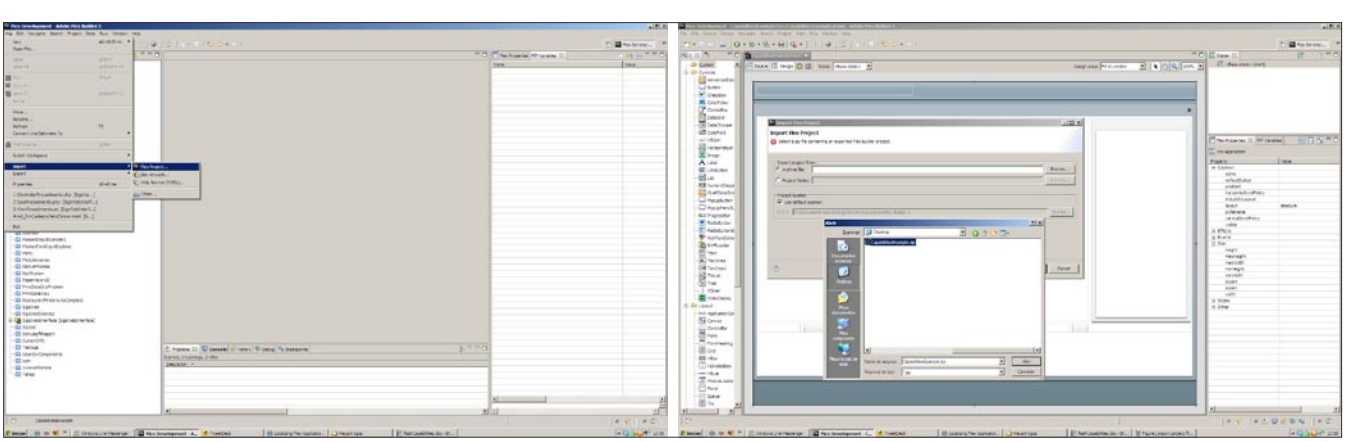


Figure 1. *Import project file*

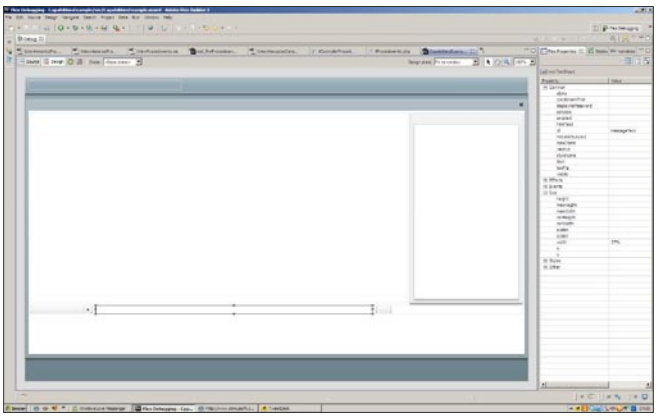


Figure 2. *Our Application UI*

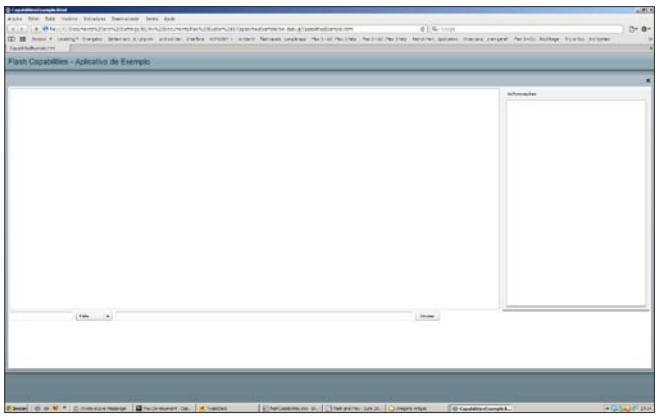


Figure 3. *App UI localized on the fly*

Listing 2. *Function that is called when the Application loads*

```
/**
 * Set event listeners and call the app functions
 */
public function load():void
{
   this.btn_send.addEventListener(MouseEvent.CLIC
               K, this.sendMessage);

   this.messageText.addEventListener(KeyboardEven
               t.KEY_DOWN,this.sendMessage);
            this.messageText.addEventListener(Keybo
               ardEvent.KEY_DOWN,this.sendMessage);
   this.checkLanguage();
   this.populateCombo();
   this.showDebugInfo();

}

//--------------------------------------------------
```

Listing 3. *Setting the localized texts to be applied in app controls*

```
/**
 * Check the language of client system
 */
private function checkLanguage():void
{
   var language:String = Capabilities.language;
   switch(language)
   {
     case 'pt':

        this.applicationTitle  = 'Flash Capabilities
             - Aplicativo de Exemplo';

        this.buttonLabel = 'Enviar';
        this.user        = 'Você';

        this.infoPanel = 'Informações';

        this.comboOptions = new ArrayCollection([{label:
              'Fala',data:'0'},

                {label:'Grita',data:'1'},

                {label:'Sussurra',data:'2'}]);
           break;

     case 'en':

        this.applicationTitle   = 'Flash
            Capabilities - Example App';

        this.buttonLabel = 'Send';
        this.user        = 'You';

        this.infoPanel = 'Information';

        this.comboOptions = new ArrayCollection([{label:
              'Speaks',data:'0'},
                  {label:'Shouts',data:'1'},
                  {label:'Wihspers',data:'2'}]);
           break;

     default:

       this.applicationTitle   = 'Flash
            Capabilities - Example App';
       this.buttonLabel = 'Send';
       this.user        = 'You';
       this.infoPanel = 'Information';
      this.comboOptions = new ArrayCollection([{label:
              'Speaks',data:'0'},
                       {label:'Shouts',data:'1'},
                  {label:'Wihspers',data:'2'}]);
     }
   this.localizeApplication();
}
//--------------------------------------------------
```

Listing 4. *Applying localized text to UI*

```
/**
 * Localize the application
 */
private function localizeApplication(language:
            String=null):void
{
   //Set UI controls to proper language
   this.appTitle.text  =
            this.applicationTitle;
   this.btn_send.label = this.buttonLabel;
   this.InfoPanel.title  = this.infoPanel;
}
```

Listing 5. *Setting the dataProvider for the comboBox*

```
/**
 * Localize the application
 */
private function localizeApplication(language:
            String=null):void
{

   //Set UI controls to proper language
   this.appTitle.text  = this.applicationTitle;
   this.btn_send.label = this.buttonLabel;
   this.InfoPanel.title  = this.infoPanel;
}
```

Listing 6. *Show some more info in debug panel*

```
/**
* Show SystemInfo
*/
private function showDebugInfo() : void
{
   this.debug.text += 'Player:' +
            Capabilities.playerType + '\n';
   this.debug.text += 'Audio:' +
            Capabilities.hasAudio + '\n';
   this.debug.text += 'OS:' + Capabilities.os
            + '\n';
   this.debug.text += 'Printing:' +
            Capabilities.hasPrinting + '\n';
   this.debug.text += 'Streaming Audio:' + Capab
            ilities.hasStreamingAudio + '\n';
   this.debug.text += 'Streaming Video:' + Capab
            ilities.hasStreamingVideo + '\n';



}
//-----------------------------------------------
```

- os – current operating system
- playerType – type of the runtime environment
- version – Specifies the Flash Player or Adobe AIR platform and version information

Localizing the UI

Let's open the `CapabilitiesExample.as` file inside the *logic* folder to check how our user interface was localized.

After importing the required classes to be used by our app, declare private variables and we will access and set their values via getters and setters methods/ functions to the corresponding language text and apply to the UI see (Listing 1).

Notice: This way of translating the UI is for demonstration only, I would use locates to translate the interface. This can be done by creating properties files where you would define the localized assets.

For the purpose of this article, i just created the attributes and used getters and setters to retrieve and set their values.

In another opportunity, I will show you how to translate the user interface using resource Bundles.

When the app starts, the load function runs (see Listing 2). Then the checkLanguage function is called, it checks the user's system language and sets the localized text to be applied to UI controls (see Listing 3). After that, the `localizeApplication` function applies the localized text to be interface elements (see Listing 4). The `populateCombo` function then sets the localized `dataProvider` for the `comboBox` and showInfo lists some of the Capabilities properties in the debug panel (see Listing 5 and Listing 6).

Complete Flash Capabilities Properties List

All other information about the user system is available in the System Info Panel. Here is a list of all Flash Capabilities properties:

Conclusion

As you can see in this small example application, with the use of the `flash.system.Capabilites class`, you can provide your app with many cool features.

For example, you can use the `serverString` property to send the information Flash Capabilities provides to a service in your server to keep detailed statistics reports.

The example application used in this article will be continuously updated to demonstrate other useful and interesting topics of the Flash Platform and Object Oriented Programming.

Comments and technical queries can be sent.

MARCELO PIRES

Marcelo Pires is a RIA Developer at RGSistemas, Brazil Where he's currently working on the development of the ESUS System and other interesting things

More info: www.rgweb.com.br

Contact details: email:marcelo@rgweb.com.br, twitter:@MarcPires

The Ultimate Checkbox List Pattern

Flex 4 containers are not close relatives of Flex 3 Containers. If you want to avoid family feuds, you need to take care of the children. In this article, we'll find out how.

What you will learn…

- How to use events for bi-directional communication
- How to use events to manage checkboxes in lists and data grids

What you should know…

- How events work, including bubbling events
- How to use item renderers

A common UI task is to add checkboxes to a grid or a list. Instead of adding Boolean property values to the dataProvider items, the Ultimate Checkbox List Pattern leaves the underlying display item alone and uses membership in a separate list or collection of items to indicate if a row is selected (e.g. a checkbox is checked). Often, this list of selected items itself is very useful.

The key to this pattern is anonymous, two-way communication between the UI component (e.g. a checkbox in an item renderer) and a controller (e.g. in a container). The list's controller need not know what components are sending messages asking if this or that item is or is not selected, and the view (i.e. the item renderer) need not know who is listening and answering its questions.

After introducing the Questioning Event pattern, we will use it to create a pattern specifically for checkboxes in lists and grids.

Using Events To Ask Questions

The Flash framework includes a messaging protocol that passes specific objects (that descend from flash.event.Event) from the sender to all of the listener functions registered for that event type. Events dispatched (aka *fired*, *launched*, *broadcast*, *thrown*) call these event handler functions. The dispatcher does not care how many listeners have registered themselves or what they do: it just calls every one registered for its event type.

If an event is flagged with the parameter *bubbles*, then all of its parent and grandparent containers will also dispatch the same event. This means that a listener registered on a container will receive all bubbling events dispatched by every component inside that container (that are of the *type* registered).

Event handlers are, at their core, callback functions. One can call them explicitly, but they are designed to be registered with a dispatcher, and wait for it to call them. One should keep these callback functions very lightweight *because dispatching events is not asynchronous*.

One fires an event by calling a dispatcher's dispatchEvent method, which then iterates through the list of registered listener functions, calling each one with the Event object as a parameter, and waiting for it to return before calling the next listener function; then the dispatchEvent method itself returns. The code that calls `dispatchEvent()` can assume that when that call returns, all the listeners have had a chance to see *and modify* the Event object.

We can use events as *bidirectional* messages between the dispatcher and the listeners, not just a one-way notification system; the event object can receive information from the listeners as well as present information. Combined with bubbling events that allow containers to get events without registering with the originating dispatcher, this patterns allows nested objects like ItemRenderers to *ask questions* of their high-level

containers without each intermediate container passing the events up and the results down. This relationship is a loosely-linked one: the dispatcher does not know what object is setting the answer properties of the Event object; the listener does not know what object is dispatching the Event object. The form can add or remove containers without breaking the protocol.

For example, an ItemRenderer needs to know if the data it is displaying is part of the set that the user wants to focus on. The data itself does not know, but the top-level form object does. The ItemRenderer can dispatch an event containing its data object and a property for the current status see (Listing 1).

The item renderer needs to use this event to ask a question and then act on the answer (e.g. setting the isInUserFocus property). Note that we are creating the new event with the *bubbles* parameter see (Listing 2).

A container that holds the list or grid with the item renderer must listen for and answer the `AmIInUserFocusEvent`. Note that it registers the listener *with itself* because the bubbling event will act as if the container dispatched the event see (Listing 3).

Checkbox Selection Communication

Adding a checkbox to a list or data grid involves creating an item renderer that contains a checkbox component as well as some ActionScript. The previous examples show how that item renderer can dispatch a bubbling event to which some listener in the container will add data (e.g. answering the question). Checkbox views not only need to know if they are currently selected, but also need to signal when the user explicitly selects or de-selects them.

In addition to that, the item renderer that contains the checkbox needs to know when the selection list changes because something else changed it. This allows the UI to show the selection list, for example, and let the user de-select items from it as well as toggle them on and off from a source list.

One way is to register a listener function in the item renderer with an event dispatcher that sends change events whenever any part of the selection list changes. Don't be clever and optimize events for specific items. Don't depend on the type of event passed to the listener. On any change, every single view checks its selection status. While this might look like a lot of event traffic, it is inexpensive traffic. Remember that the Flex components re-use item renderers, so a grid will have only as many as it has visible rows (plus one or two).

A custom event for checking (and changing) selection might look like: see Listing 4.

The pattern uses this event for two types of communication: checking selection and signaling selection. The first time we check the selection, we also register the notificationListener see (Listing 5).

And the list controller registers the `notificationListener`, if present, and manages the changes in selection as

Listing 1. *Sample event class*

```
public class AmIInUserFocusEvent extends Event
{
  public function AmIInUserFocusEvent(type : String = AM_I_IN_FOCUS,
    bubbles : Boolean = true, cancelable : Boolean = false)
  {
    super(type, bubbles, cancelable);
  }

  public var rowData : RowData = null;
  public var isInUserFocus : Boolean;
}
```

Listing 2. *Code inside an item renderer*

```
private function onDataChange(aData : Object)
{
  var newEvent : AmIInUserFocusEvent = new AmIInUserFocusEvent(AM_I_
                    IN_FOCUS, true);
  newEvent.rowData = aData as RowData;

  this.dispatchEvent(newEvent); // ---- listeners execute here

  if (newEvent.isInUserFocus)
    text.styleName = "focusStyle";
  else
    text.styleName = "neglectedStyle";
}
```

Listing 3. *Code in the container (aka list controller)*

```
private function onInitialized() : void
{
  this.addEventListener(AM_I_IN_FOCUS, onFocusQuestion);
}

private function onFocusQuestion(event : AmIInUserFocusEvent) : void
{
  event.isInUserFocus = focusList.contains(event.rowData);
}
```

well as the inquiries. Note that the selection list can be any group of items or references, and we can listen for Event.CHANGE or any other event that accurately notifies the listener of changes see (Listing 6).

Listing 4. *SelectionEvent class*

```
public class SelectionEvent extends Event
{
   public function SelectionEvent(type : String,
     bubbles : Boolean = true, cancelable : Boolean = false)
   {
     super(type, bubbles, cancelable);
   }

  public var rowData : RowData = null;
  public var isSelected : Boolean = false;
  public var notificationListener : Function = null;
}
```

Listing 5. *Code inside an item renderer*

```
private function onDataChange(aData : Object)
{
  var newEvent : SelectionEvent = new SelectionEvent(AM_I_SELECTED,
                    true);
  newEvent.rowData = aData as RowData;
  newEvent.notificationListener = this.checkSelection;

  this.dispatchEvent(newEvent); // ---- listeners execute here

  this.checkBox.selected = newEvent.isSelected;
}

private function checkSelection(event : Event) : void
{
  var newEvent : SelectionEvent = new SelectionEvent(AM_I_SELECTED,
                    true);
  newEvent.rowData = aData as RowData;

  this.dispatchEvent(newEvent); // ---- listeners execute here

  this.checkBox.selected = newEvent.isSelected;
}

private function onCheckboxChange(event : Event) : void
{
  var newEvent : SelectionEvent = new SelectionEvent(SELECTION_
                    TOGGLE, true);
  newEvent.rowData = aData as RowData;
  newEvent.isSelected = this.checkBox.selected;

  this.dispatchEvent(newEvent);
}
```

The Selection Toggle Event Cycle

- The view dispatches an `AmISelected` event with a reference to its notification listener function in it.
- The list controller adds the event listener
1. The user clicks the checkbox
2. The item renderer dispatches a `SELECTION_TOGGLE` event
3. The controller updates the selection list
4. The list dispatches a `CHANGE` (or `COLLECTION_CHANGE`) event to all the listener functions
5. The view's listener functions dispatch an `AmISelected` event (without a listener reference in it)
6. The list controller received the `AmISelected` event object and sets its `.isSelected` property based on the list
7. The item renderer checks the .isSelected property and updates its display based on the answer see (Figure 1)

Multiple States and Multiple Dependencies

In addition to controlling the selection state, the Ultimate Checkbox List Pattern can support any sort of operational control within the list as well, For example, we can use it to control which items can be selected. The new event simply adds an additional flag: `isEnabled`. Note that it defaults to *true* so if the listener does not implement any actions, it defaults to the intended behavior see (Listing 7).

The item renderer could change the style or behavior of the control based on `isEnabled` see (Listing 8).

The list controller can use any expression or protocol to decide if any specific `rowData` should be enabled or disabled. In this example, it uses the total number of selected items see (Listing 9).

A complex example uses attributes of `rowData` itself in combination with system or user rights see (Listing 10).

As each event can have more than one listener, we can use separate methods on the controller to add separate layers of validation for the `AM_I_SELECTED` events: one method could check the user's rights; another could check the state of the form; and another could enforce the maximum number of selected items. As long as each of these event handlers respected

the value that any other listener might have set (e.g. if `(event.isEnabled)` ...), then we don't have to worry about the sequence they get called in (e.g. by using listener priority).

Listing 6. *Code in the container (aka list controller)*

```
protected var selectionList : ArrayCollection = new ArrayCollection();

private function onInitialized() : void
{
  this.addEventListener(AM_I_SELECTED, onAmISelected);
  this.addEventListener(SELECTION_TOGGLE, onSelectionToggle);
}

private function onAmISelected(event : SelectionEvent) : void
{
  event.isSelected = selectionList.contains(event.rowData);

  if (event.notificationListener != null)
  {
    selectionList.addEventListener(CollectionEvent.COLLECTION_
                  CHANGE,
        event.notificationListener, false, 0, true);  //  weak reference
  }
}

private function onSelectionToggle(event : SelectionEvent) : void
{
  if (event.isSelected && (!selectionList.contains(event.rowData)))
    selectionList.add(event.rowData);
  else ((!event.isSelected) && selectionList.contains(event.rowData))
    selectionList.remove(event.rowData);
}
```

Using a Proxy Event Dispatcher

Another complexity is pushing changes from different sources to the views. For example, if the UI has a radio button that filters out items of a certain type, the function listening to changes in the selection list won't know when that radio button changes. We could register the notification listener to multiple sources, however that does increase the number of registrations in memory and makes the event cycle a little more complicated.

Another way is to centralize the change event notification in the controller by creating a stand-alone EventDispatcher, registering all notification listeners with it, and having it proxy change events from any number of sources. Any change to the selectionList is forwarded by the event dispatcher through a listener. Any other events or changes can call `announceChange()` to trigger every view's `AM_I_SELECTED` event see (Listing 11).

Notes

Selection List vs. dataProvider Object flags

• The Ultimate Checkbox List Pattern has several advantages over using

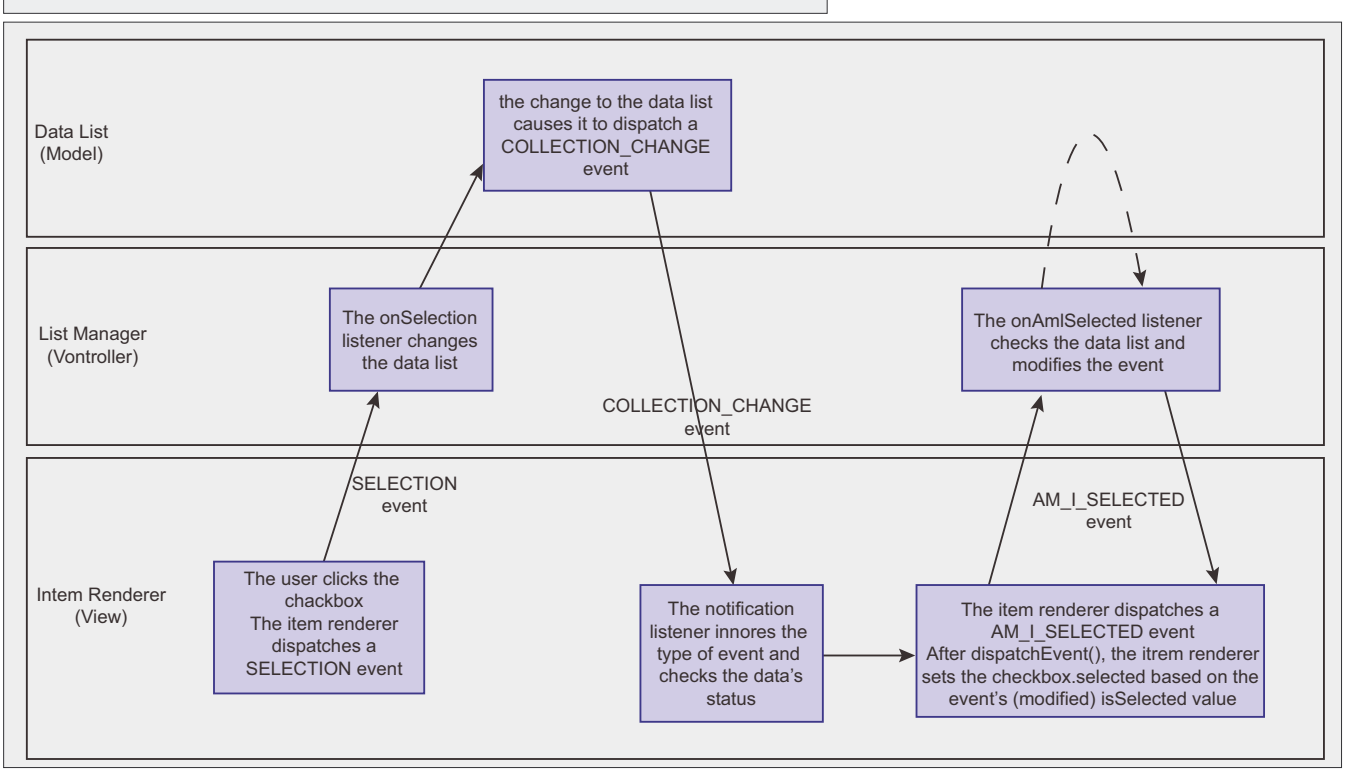


Figure 1. *A SELECTION_TOGGLE event cycle*

a flag in the dataProvider item to persist checked status:

- The dataProvider items can remain pure value object (e.g. unchanged from an loosely-linked data module)
- The same item can have separate selection-states on an infinite number of lists without making change after change to a data object's structure
- The same data value can be synchronized even when represented by different object instances (or even different object types). The selection list can determine membership using an id value or values rather than object identity.
- The list of selected items is always current and available at a high level; one does not need to iterate through the dataProvider items
- Items remain selected even when they are not visible or if they are not in the dataProvider's collection. One can move off of a page and find previously selected items are still selected when one returns.
- Items can be selected and deselected across several different views One can present the user with both browsing lists and searching capability, and every item on the lists and in the search results will reflect prior and future selection states.

Listing 7. *SelectionEvent class with isEnabled*

```
public class SelectionEvent extends Event
{
  public function SelectionEvent(type : String,
    bubbles : Boolean = true, cancelable : Boolean = false)
  {
    super(type, bubbles, cancelable);
  }

  public var rowData : RowData = null;
  public var isSelected : Boolean = false;
  public var isEnabled : Boolean = true;
  public var notificationListener : Function = null;
}
```

Listing 8. *Code inside an item renderer respecting isEnabled*

```
private function checkSelection(event : Event) : void
{
  var newEvent : SelectionEvent = new SelectionEvent(AM_I_SELECTED, true);
  newEvent.rowData = aData as RowData;

  this.dispatchEvent(newEvent); // ---- listeners execute here

  this.checkBox.selected = newEvent.isSelected;
  this.checkBox.enabled = newEvent.isEnabled;
}
```

Listing 9. *Code in the container (aka list controller) setting isEnabled*

```
private function onAmISelected(event : SelectionEvent) : void
{
  event.isSelected = selectionList.contains(event.rowData);
  event.isEnabled = selectionList.length < MAX_SELECTION_COUNT;

  if (event.notificationListener != null)
  . . .
}
```

Setting List Controller Listeners

Setting listeners in the startup event cycle is tricky. In general, I use `addEventListener()` in the creationComplete event for listeners on the form, however this does not work if the checkbox list or grid is a design-time child on the form (e.g. it's in the MXML or created by `createChildren()`). By the time the `creationComplete` event fires, the grid has already initialized and all the item renderers have dispatched their first AmISelected events, which receive no answers because the listener is not set up yet.

Register `amISelected` listeners in the *initialized* event.

Messages vs. Interfaces

A different way of loosely linking parts of a system is interfaces. One could pass an instance of an interface that had `toggleSelection` and `amISelected` methods to an item renderer. The class of that instance could be anything; the caller would not know who was executing the method and the method would not know who was calling it. Passing an instance more than one layer deep, however, creates a loosely-linked chain, and the more links in a chain, the more fragile it is.

Every container between the list manager (e.g. the form or component's top-level container) would have to pass the instance on to every child that could hold it (probably because the children implemented an interface themselves). Components that had no interest in this interface would still be responsible for passing it on.

Messages, however, are already part of every component's interface: they allow listeners to hook into the messages;

the message framework moves the event objects through the containership model; and the event objects themselves can change without changing the interface. One can even encapsulate some set of components and containers into a library component and embed it in another container without changing this pattern.

On the 'Net

Sample Application with View Source – *http://flex.santacruz-software.com/SampleMessageBasedCheckboxListApplication/SampleMessageBasedCheckboxListApplication.html*

Message-Based MVC

The Ultimate Checkbox List Pattern resembles a Model-View-Controller: the item renderer would be the view; the list of selected items would be the model; and the event listeners on the list manager would be the controller. In a classic MVC pattern, the controller would push the changes into the view, and would need to know how it worked (i.e. what method to call or property to push). The UCLP has the model pushing a change notification into a generic listener in the view, and then the view pulls the change information from the model; the module (and the controller, for that matter) do not know anything about the view besides a reference to the view's notification listener.

Listing 10. *Code in the container (aka list controller) setting isEnabled*

```
private function onAmISelected(event : SelectionEvent) : void
{
  event.isSelected = selectionList.contains(event.rowData);
/*     event.isEnabled defaults to true even if the view is not enabled,
       but might be set false by a different listener  */
  if (event.isEnabled)
  {
    event.isEnabled = ((!this.isReadOnly) && (!global.isDemo) &&
     ((event.rowData.access == ALL_ACCESS) ||
     ((event.rowData.access == ACCOUNT_ACCESS) &&
       (event.rowData.ownerId == global.user.userId)) ||
     ((event.rowData.access == MANAGER_ACCESS) &&
       (event.rowData.departmentId == global.user.departmentId) &&
       (global.user.accessLevel == MANAGER_ACCESS_LEVEL)));
  }
  if (event.notificationListener != null)
    . . .
}
```

Listing 11. *Using a proxy event dispatcher in the controller*

```
private var _eventDispatcher  EventDispatcher = new EventDispatcher();
protected var selectionList : ArrayCollection = new ArrayCollection();
private function onInitialized(event : Event) : void
{
  this.addEventListener(AM_I_SELECTED, onAmISelected);
  this.addEventListener(SELECTION_TOGGLE, onSelectionToggle);
  selectionList.addEventListener(CollectionEvent.COLLECTION_CHANGE,
                    announceChange);
}
private function onAmISelected(event : SelectionEvent) : void
{
  event.isSelected = selectionList.contains(event.rowData);
  event.isEnabled = selectionList.length < MAX_SELECTION_COUNT;
  if (event.notificationListener != null)
    _eventDispatcher.addEventListener(Event.CHANGE, event.notificati
                    onListener);
}
private function announceChange(event : Event = null) : void
{
  _eventDispatcher.dispatchEvent(new Event(Event.CHANGE));
}
```

Matching By Value vs. Matching By Reference

As shown in the sample application, one can use a more robust test to see if a rowData is in the selected list than simply checking `getIndexOf()`. While the `rowData` might be a complex Value Object, this pattern also lends itself to simple data objects that might be dynamically created to wrap one or two column values. In this case, the object itself might not come from a global (i.e. singleton) store; one might have several simple instances that represent the same values, but not the same instances in memory.

Make It Work For You

Use this pattern to support your needs: if you have different representations of the same concept and you want to select or de-select the concept, then make the `AmISelected` check if the concept is on the list; if the objects in question are references to singleton objects on a central list (e.g. value objects), then check if the object itself is referenced by the list.

RICHARD C HAVEN

Richard Haven is an old-school OOP developer who remembers the problems objects and interfaces addressed when they were exciting and new. His experience with Delphi and its application framework has transferred surprisingly well to ActionScript 3 and the Flex application framework.

He works as a consultant, writer, international trainer, and blogger (noclevercode.wordpress.com) available at rhaven@noclevercode.com

Flex 4

Parental Concerns

Finding the parent of a Flex 3 control is easy, but the parentage of a Flex 4 control is harder to pin down. In this article we look at some ways of tracking down the visible parents of Spark components.

What you will learn…

- How the parent properties differ in Flex 3 and Flex 4
- Important differences between parent and owner
- How to work with MX and Spark parents in a consistent manner

What you should know…

- The Flex 3 parent relationships of Containers and children
- The difference between MX and Spark containers
- How to create and build Flex 4 applications

Last month I explained some differences between Flex 4's new Spark containers and Flex 3's MX or *Halo* containers. In particular, I looked at the way in which parent-child relationships are redefined. In the last article, I concentrated on the differences between MX *children* and Spark *elements*. This month I want to look at the other side of that relationship: parents.

When you put a control into an MX container, the container becomes the control's parent. So if button1 is added as a child of panel1, you can enter the code `button1.parent` in order to retrieve a reference to panel1.

But that is not the way Spark containers work. If you put button1 into a Spark panel, `button1.parent` will retrieve a reference to an unseen Group object. In fact there would be four *levels* of parent controls (three Groups and a PanelSkin) intervening between the button and the Spark panel containing it. In other words, while a Spark container such as a Panel may be the first *visible parent* of a control, it is not, in fact, its actual parent.

Parents and Owners

If you ever need to manipulate parent/child relationships at runtime, this can pose obvious problems. Imagine, for example, that you want to move a component from one container into another in response to user interaction or size a control as a percentage of its container. In Flex 3, you would do this simply by accessing the control's `parent` property. In Flex 4, when using Spark containers, you can't do that.

This is a real-world problem. It is one which my company was forced to solve in order that to implement a Flex 4 version of the visual design environment for our Flash Platform IDE, Amethyst. Even though Amethyst is built into Microsoft Visual Studio, the Amethyst Designer is a Flex application. It allows programmers to use MX and Spark components in the same design and

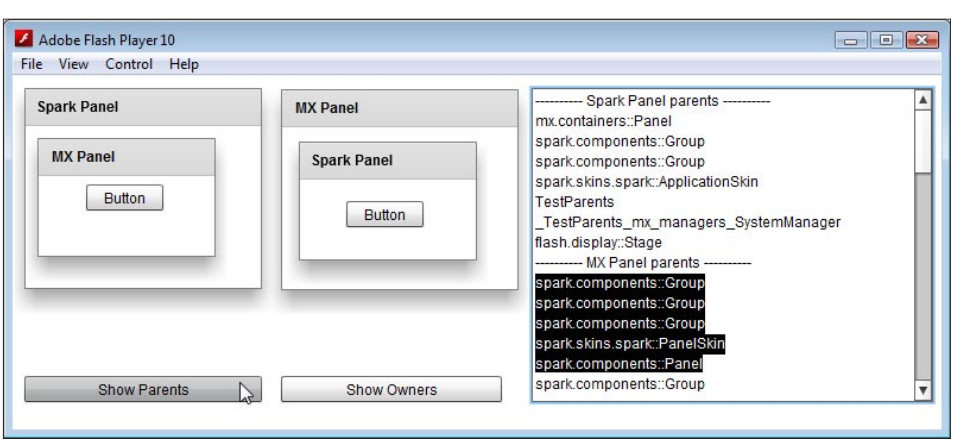


Figure 1. *This application shows that the parent of the MX Panel (on the left) is not the Spark Panel which contains it! There are four intervening invisible parents between the MX Panel and the Spark Panel. Our task is to find the first visible parent*

to drag any component into any container. Whenever a component is dropped into a container, the code has to calculate all the parent-child relationships. These may go *many levels deep* – say if an MX button is dropped into a Spark Panel inside an MX Canvas inside a Spark BorderContainer inside an MX Accordion in a Spark application.

Before the advent of Flex 4, all these kinds of relationships could be calculated simply by calling `parent.parent.parent` and so on until no more parents (`null`) were returned. But since Spark containers have, in effect, redefined the meaning of `parent` this no longer does the trick.

This is not a problem unique to the Amethyst Designer. Any Flex 4 application may use a mix of Spark and MX controls (and bear in mind that Spark has no equivalents of certain MX controls such as Navigators, so this is not an unlikely scenario). As a consequence, Flex 4 developers need a safe and reliable way of traversing back through an unknown number of parents in order to locate the first *visible parent*.

One simple way of doing this would be to use the `owner` property instead of the `parent` property. The default value of `owner` is the same as an MX `parent`. That is, if you have button inside a Spark Panel, `button1.owner` will return a reference to the Spark Panel in just the same way that it would if it were in an MX Panel. But there is a problem. While `owner` usually returns the visible `parent`, there is no absolute guarantee that it will do so. The `parent` property is *intrinsic* – it is read-only and defines an existing relationship that cannot be changed at runtime. The `owner` property, however, is writeable. It may be altered at design time or at runtime.

For example, it is perfectly legitimate to define button1 as the `owner` of button2. This would cause huge problems for any code which assumes that `owner` always returns a reference to a control's container. Worse still, button1 could set button2 to be its `owner` while button2 sets button1 to be its `owner`. If your code now tries to climb back through button1's chain of `owners` it will do so for ever or until your program crashes. So, in short, while the `owner` property works like the Flex 3-style `parent` most of the time it is not safe to assume that

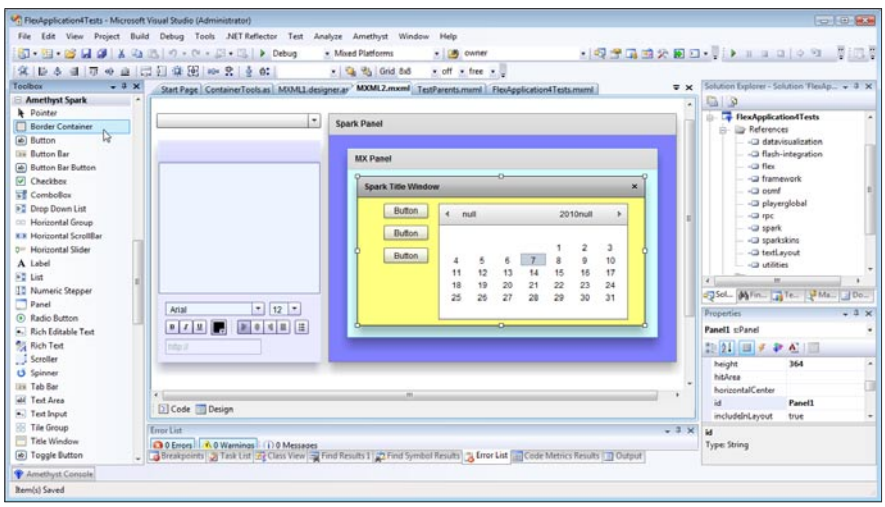


Figure 2. *The Amethyst Designer is a real-world Flex application which allows developers to create user interfaces by dragging and dropping components into both MX and Spark Containers. Here you can see a design that nests controls several levels deep inside MX and Spark parent containers*

Listing 1. *Determine whether or not a control is a visible container*

```
private static const SPARK_CONTAINER:String = "SPARK_CONTAINER";
private static const MX_CONTAINER:String = "MX_CONTAINER";
private static const NOT_A_CONTAINER:String = "NOT_A_CONTAINER";

public static function getVisibleContainerType( aCtrl:
                    DisplayObjectContainer ):String {
   var cType:String = NOT_A_CONTAINER;
   if( aCtrl is Container ) {
      cType = MX_CONTAINER;
   } else if( aCtrl is SkinnableContainer ) {
      cType = SPARK_CONTAINER;
   } else {
      cType = NOT_A_CONTAINER;
   }
   return cType;
}

// is aCtrl some kind of (MX or Spark) Container?
public static function isAContainer( aCtrl:DisplayObjectContainer ):
                    Boolean {
   return ( getVisibleContainerType( aCtrl ) != NOT_A_CONTAINER );
}

// is aCtrl NOT some kind of (MX or Spark) Container?
public static function isNotAContainer( aCtrl:DisplayObjectContainer ):
                    Boolean {
   return ( !isAContainer( aCtrl ) );
}
```

Listing 2. *return the first visible parent of a control*

```
public static function visibleParent( aCtrl:DisplayObjectContainer ):
                    DisplayObjectContainer {
   var theParent:DisplayObjectContainer;
   var continueLooking:Boolean;
   var returnval:DisplayObjectContainer;
   continueLooking = true;
   theParent = aCtrl;
   if( theParent != null ) {
      while( continueLooking ) {
         theParent = theParent.parent;
         continueLooking = ( ( theParent != null ) && ( isNotAContainer(
                    theParent ) ) );
      }
   }
   returnval = theParent;
   return returnval;
}
```

Listing 3. *Display the first visible parent (here ta is a TextArea)*

```
private function showAllVisibleParents( aCtrl:UIComponent ):void {
var c:DisplayObjectContainer = aCtrl;
   while( c != null ) {
      c = ContainerTools.visibleParent( c );
      ta.text += "\n" + getQualifiedClassName( c );
   }
}
```

it will always work in that way. For a more reliable way to find the `visible parent` of a control, you need to do a bit more coding.

Finding A Visible Parent

One way of doing this would to categorize each control as either a Spark container, an MX container or a non-container. For the sake of simplicity I'll assume that you will only consider Spark SkinnableContainer components and MX Containers as *visible* and will ignore Spark Groups. You can now write some simple functions to determine whether or not any given control is a *visible* container (Listing 1). This makes it pretty easy to write a function that returns the first *visible parent* of a component (Listing 2). Assuming this code is placed into a class called ContainerTools you can now call `ContainerTools.visibleParent(c)`, where *c* is some component, in order to find the first MX Container or SkinnableContainer in its chain of parents (Listing 3).

If you wanted to consider Groups as *visible* parents too, you would have to decide whether or not the Groups inside SkinnableContainers should be ignored. In most cases, you probably would want to ignore them – that is, you would want to treat only those Groups that have been explicitly added to a design as being *visible* parents (even though, just like MX canvases, they may not be visible unless borders or colours are applied). One way to do this would be to maintain a reference list – a simple array would do – of all the components in your application. When looking for parents, you would then be able to check if a control exists in the reference list; if not, it should be ignored. This would ensure that, for instance, a Panel would be in your reference list whereas its contentPane (this is the container, nested inside a Panel, that actually `contains` the Panel's controls) would not.

The more checking you add to your program, of course, the more complex the coding becomes. For many programs, it may be that using the `owner` property will suffice. This has the advantage of simplicity even though, as I explained earlier, it is not wholly bulletproof. If you plan to assign new owners at runtime, you might want to set a variable (`called`, `perhaps`, `originalOwner`) to the initial `owner` value prior to doing so. Incidentally, you should also be aware that even though you may explicitly assign one control as the `owner` of another, in some cases, `owner` may be reassigned by the Flex framework. This is something I encountered with SkinnableContainer objects. I assigned the *owner* property of a SkinnableContainer to a specific control (which was not in the SkinnableContainer's chain of parents) only to discover that Flex subsequently reassigned `owner` the SkinnableContainer's `parent` container.

The strategy which you adopt when handling MX and Spark parents depends on the nature of the application you are coding. For absolute reliability in all circumstances, you may need to do quite a bit of hand coding. For a simple application, the `owner` property may be all you need. At any rate, I hope that my two articles give you a few hints on various ways of dealing with the problems posed by parents and children when using MX and Spark containers in Flex 4 applications.

HUW COLLINGBOURNE

Huw Collingbourne is Director of Technology at SapphireSteel Software. Over more than 20 years, he has programmed in languages ranging from Ruby to C# and has been a technical columnist for computer magazines such as PC Pro and PC Plus. He is the software lead of the Amethyst Designer, the Flex user interface design environment of the Amethyst IDE for Visual Studio. SapphireSteel Software: http://www.sapphiresteel.com/

SIMPLIFY GAME DEVELOPMENT BY USING GAMERSAFE

GamerSafe is a powerful tool that allows you to add quality features to your Flash game quickly and easily.

- **Universal Saved Games**
  So players can resume their game even on another website!
- **Advanced Trophy System**
  Reward your players with points they can spend on game stuff!
- **Customizable Leaderboards**
  Bring out the competitive spirit .. painlessly!
- **Newsletter Support**
  Keep the players updated on your newest games!
- **Micro-Transactions**
  Create another revenue stream for your game!
- **LevelVault**
  Players can store & share levels that they create in your game.

GAMERSAFE
Your gaming life, simplified

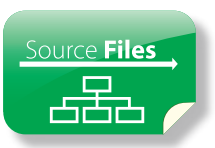
FIND OUT MORE

GAMERSAFE.COM

Flex and PHP

Charting

One of the places where Flex and PHP go really well together is in creating data visualizations.



What you will learn…

- How to structure your Zend Framework application for easy charting
- How to create interactive charts with Flex

What you should know…

- Basic knowledge of the Zend Framework project structure
- Basic knowledge of Flex

PHP makes it incredibly easy to retrieve information from a database and manipulate it while Flex provides a user interface layer that is interactive and capable of quickly calculating shapes which is perfect for something like charting. One of the main issues though is how to structure the data on the PHP side in order to bring it into a Flex application.

Normally, you retrieve a set of data in a pseudo-random order, potentially ordered by id, and then plot that data. In a lot of cases, that works just fine. But for complex data types or for charts where you want to make it easy to interactively drill down into data, you have to use some tricks of the Flex framework to plot the data correctly. Luckily we can organize the data in PHP and send it over in a way that makes it more simple to plot on a chart.

We'll start with a dataset from *http://peakbagger.com/* that returns a number of high mountains from a few different states. Initially, the data on the chart will show the aggregate number of peaks with a specified minimum level of topographic prominence (*http://en.wikipedia.org/wiki/Topographic_prominence*) but allow the user to interact with the chart and click on it to get a list of peaks with both elevation and prominence. One of the benefits of Flex is that we can just grab the data from the database up front and then change the chart without having to make another call to the database.

The structure of the data is an array of arrays with the subarray containing data about individual peaks. The database and object structure looks like this:

Listing 1. *PHP Class for selecting data from the database*

```
class Peaks
{
   public function getBarChartPeaks($maxStates, $minProminence)
    {
       $tbl = new Model_DbTable_Peak();
       $select = $tbl->select()
                  ->from('peaks', array('state',
                      'prominence',new Zend_Db_
                      Expr('SUM(prominence) AS promTop')))
                  ->group('state')
                  ->order('promTop DESC')
                  ->limit($maxStates);
         $rs = $select->query();
         $res = array();
       foreach ($rs as $row) {
         $where = array('state = ?' => $row['state'],
                      'prominence >= ?' => $minProminence);
           $stateRes = $tbl->fetchAll($tbl->getAdapter()-
                      >quoteInto($where,""), 'prominence
                      DESC');
          $res[] = $stateRes->toArray();
         }
       return $res;
    }
}
```

```
peaks |
--------------------------------
id     int
peak_name varchar(200)
elevation int
prominence    int
county varchar(200)
state      varchar(2)
range      varchar(200)
isolation float
```

When the data is returned to Flex, it's sent as an array of arrays that we group and sort using the database abstraction classes of the Zend Framework. The code that is called directly from Flex looks like this: see (Listing 1.)

But these rely on a structured Zend Framework project.

The first step in the Flex application is to connect to the PHP service.. I won't go into detail on how to do that, instead, I'll point to this introduction to using the Data Wizards to call PHP functions (*http://www.adobe.com/devnet/flex/articles/flashbuilder4_php_part1.html*).

Listing 2. *ActionScript functions for calling PHP classes*

```
        protected function getBarChartPeaks(maxStates
                    :int, minProminence:int):void
        {
           getBarChartPeaksResult.token = peaksSer
vice.getBarChartPeaks(maxStates, minProminence);
        }

        protected function getBarChartPeaksResult_
resultHandler(event:ResultEvent):void
        {
           chart.dataProvider = getBarChartPeaksRes
                    ult.lastResult;
        }
```

Listing 3. *MXML Code for the charts*

```
   <mx:ColumnChart x="10" y="10" id="chart" creationCom
                    plete="getBarChartPeaks(5,5000);"
                type="clustered" showDataTips="true">
      <mx:horizontalAxis>
         <mx:CategoryAxis id="catField" dataFunction
                    ="categoryFunction" />
      </mx:horizontalAxis>
      <mx:series>
         <mx:ColumnSeries id="colSeries" dataFunctio
                    n="dataFunction" />
        <mx:ColumnSeries id="promSeries" yField="prominence"
xField="peak_name" displayName="Prominence" visible="false" />
      </mx:series>
   </mx:ColumnChart>
```

Using that method, there are a couple of functions that belong in the Script block: see (Listing 2).

With data being returned correctly we can start creating the `ColumnChart`: see (Listing 3).

There are two parts to this column chart, the horizontal axis and then a `ColumnSeries`. The `ColumnSeries` is what actually contains the data. The axis can be a linear axis,

Listing 4. *Script block and code for creating custom data formats*

```
<fx:Script>
<![CDATA[
import mx.charts.chartClasses.AxisBase;
import mx.charts.chartClasses.Series;
protected function dataFunction(series:Series, item:
                    Object, fieldName:String):Object
{
     if(fieldName == "yValue")
     {
          return item.length;
     } else if(fieldName == "xValue")
     {
          return item[0].state;
     } else {
          return null;
     }
}
protected function categoryFunction(axis:AxisBase,
                    item:Object):Object
{
     return item[0].state;
}
]]>
</fx:Script>
```

Listing 5. *The clickHandler for the interactive chart*

```
<mx:ColumnChart x="10" y="10" id="chart" creationComp
                    lete="getBarChartPeaks(5,5000);"
type="clustered" showDataTips="true" itemClick=
"chart_itemClickHandler(event)">
```

Then add the event handler in the script block:

```
import mx.charts.events.ChartItemEvent;
protected function chart_itemClickHandler(event:
                    ChartItemEvent):void
{
catField.categoryField = "name";
catField.dataFunction = null;
colSeries.dataFunction = null;
colSeries.yField = "elevation";
colSeries.xField = "name";
promSeries.visible = true;
chart.dataProvider = event.hitData.item;
}
```

Listing 6. *Code for the chart animation*

```
<mx:SeriesSlide id="slideIn" duration="1000"
                    direction="up" />
<mx:SeriesSlide id="slideOut" duration="1000"
                    direction="down" />
<p>Modify the ColumnSeries tags to use the
SeriesSlides by adding the showDataEffect and
hideDataEffect to apply them to the chart.</p>
<mx:series>
<mx:ColumnSeries id="colSeries" dataFunction=
"dataFunction" showDataEffect="slideIn" hideDataEffec
                    t="slideOut" />
<mx:ColumnSeries id="promSeries" yField="prominence"
xField="name" displayName="Prominence"  visible="false"
showDataEffect="slideIn" hideDataEffect="slideOut" />
</mx:series>
```

Listing 7. *Full Flex code*

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
            xmlns:s="library://ns.adobe.com/flex/spark"
             xmlns:mx="library://ns.adobe.com/flex/mx"
minWidth="955" minHeight="600" xmlns:peaks="services.peaks.*"
xmlns:peaksservice="services.peaksservice.*">
   <fx:Script>
      <![CDATA[
            import mx.charts.CategoryAxis;
            import mx.charts.chartClasses.AxisBase;
            import mx.charts.chartClasses.Series;
            import mx.charts.events.ChartItemEvent;
            import mx.charts.series.ColumnSeries;
            import mx.controls.Alert;
            import mx.rpc.events.ResultEvent;
            protected function dataFunction(series:
Series, item:Object, fieldName:String):Object
            {
            if(fieldName == "yValue")
            {
               return item.length;
            } else if(fieldName == "xValue")
            {
               return item[0].state;
            } else {
               return null;
            }
            }
            protected function categoryFunction(axis:
                    AxisBase, item:Object):Object
            {
            return item[0].state;
            }
           protected function chart_itemClickHandler(event:
                    ChartItemEvent):void
            {
                    catField.categoryField = "peak_name";
                    catField.dataFunction = null;
                    colSeries.dataFunction = null;
                    colSeries.displayName = "Elevation";
                    colSeries.yField = "elevation";
                    colSeries.xField = "peak_name";
                    promSeries.visible = true;
                    chart.dataProvider = event.hitData.item;
                    }
                    protected function getBarChartPeaks(maxSta
                            tes:int, minProminence:int):void
                    {
                    getBarChartPeaksResult.token = peaksSer
vice.getBarChartPeaks(maxStates, minProminence);
                    }
                    protected function getBarChartPeaksResult_
resultHandler(event:ResultEvent):void
                    {
                    chart.dataProvider = getBarChartPeaksRes
                            ult.lastResult;
                    }
      ]]>
   </fx:Script>
   <fx:Declarations>
      <mx:SeriesSlide id="slideIn" duration="1000"
                    direction="up" />
      <mx:SeriesSlide id="slideOut" duration="1000"
                    direction="down" />
      <s:CallResponder id="getBarChartPeaksResult"
result="getBarChartPeaksResult_resultHandler(event)"/>
      <peaksservice:PeaksService id="peaksService"
fault="Alert.show(event.fault.faultString + '\n'
+ event.fault.faultDetail)" showBusyCursor="true"/>
   </fx:Declarations>
   <mx:ColumnChart x="10" y="10" id="chart" creatio
nComplete="getBarChartPeaks(5,5000);"
                  type="clustered" showDataTips="true"
itemClick="chart_itemClickHandler(event)">
      <mx:horizontalAxis>
            <mx:CategoryAxis id="catField" dataFunct
                    ion="categoryFunction" />
      </mx:horizontalAxis>
      <mx:series>
            <mx:ColumnSeries id="colSeries"
dataFunction="dataFunction" showDataEffect="slideIn"
                    hideDataEffect="slideOut" />
          <mx:ColumnSeries id="promSeries" yField=
"prominence" xField="peak_name" displayName="Prominence"
visible="false" showDataEffect="slideIn" hideDataEff
                    ect="slideOut" />
      </mx:series>
   </mx:ColumnChart>
</s:Application>
```

which would show sequential values, a date/time axis, or a logarithmic axis. To plot more arbitrary values, for instance, a group, use the `CategoryAxis`. For this, the data is states, so the `CategoryAxis` is used. There is just one piece of data to chart, the number of peaks in each state, so we just need one `ColumnSeries`.

Normally, if the data is flat, you can use the `xField` and `yField` properties of the `ColumnSeries` component and set them to corresponding keys in the object. If you have a more complex dataset, like an array of arrays, you need to manipulate the data before charting it. The `dataFunction` property can be set on both axes and series to do just that. It takes a function as a value and then the function returns the value that you want to appear on the chart. Create a function called `categoryFunction` for the `CategoryAxis` and a function called `dataFunction` for the `ColumnSeries` and put them in between the `fx:Script` tags see (Listing 4).

The category function is pretty straight forward. It will be called as many times as there are arrays in the main array. It just pulls the state out and returns it as a category. The function for `ColumnSeries` is a bit more complicated. Any series with values has both `x` and `y` values and the `dataFunction` needs to return both of those. It is called twice the number of times as there are arrays in the main array because it gets called for both the x and the y values. Using the `fieldName` property we can figure out which one is being called. If it's the `yValue` then we return what we want to plot, the number of peaks in that array. If it's the xValue then we want to make sure it's associated with the category so we return the state.

If you run the application now, you'll see the count of peaks on the chart. Now lets allow users to drill down into the individual state and see the elevation and prominence of each peak. In Flex you can attach events for specific user actions on the chart. In this case add an `itemClick` handler and function to the `ColumnChart` so that your chart XML data looks like this: see (Listing 5).

Because the data structure is just an array of objects we can null out the dataFunctions on both the `CategoryAxis` and the `ColumnSeries` and replace them with the `yField` and `xField` values for the data we want to chart. In this case, we'll plot the elevation along the `Y` axis and show the name of the peak along the `X` axis.

The last step is to add some animation so that when the user clicks on the chart, the chart fades away and appears gracefully. To do that, we can use the series effects. Add a couple of `SeriesSlide` tags in between the `fx:Declarations` tags see (Listing 6).

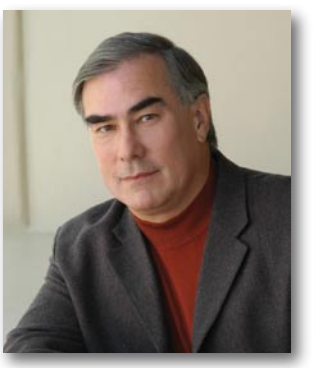
And that's all there is to it. Now you can run the application and see the final result. Here's the full Flex code: see (Listing 7).

KEVIN SCHROEDER, RYAN STEWART

Building Expert Systems in Flash with Exsys Corvid

Add Complex Logical Processing to Flash

Dustin Huntington, President of Exsys Inc.



Dustin Huntington has been a leader in the expert system field for almost 30 years. He is the primary author of several of the most popular expert system development tools and inference engines. He received his BS and MS degrees in physics and numerical analysis for Steven's Institute of Technology. Specializing in advanced technology, Dustin has worked in several government agencies and consulted with many companies worldwide. In 1983 he established Exsys Inc. and it's flagship expert system software. With tens of thousands of users, the company's products continue to evolve focusing on enterprise-wide knowledge distribution and new web technology needs. Exsys systems can be delivered online via web browsers, making this technology particularly effective for interactive web sites to provide advice for diagnostics, product support, regulatory compliance, pre-sales advice, HR and a wide range of other areas needing expert decision-making advice.

Who is Exsys Inc?

Exsys is a company specializing in expert system development tools. We have been in the expert system business for 27 years – a very long time for any software company, and essentially forever for an expert system company. Exsys was one of the first companies to produce expert system development tools for PCs (others were aimed at LISP machines when we started) and the first to find a practical way to field interactive expert systems on the web.

Our focus has been to produce expert system development tools that could be easily learned by non-programmer domain experts, but with the power and flexibility needed to handle the logic of complex real world problems. Most systems are delivered over the web with user interfaces implemented in Java or HTML, but our recent interface with Adobe Flash enables the most complex user interfaces and greatly extends what can be done with Flash.

What are Expert Systems?

The term *Expert System* is used for many types of programs that provide advice on a complex decision-making or support task comparable to what an end user would get from a human expert on the subject. Some are rather simplistic implementations of a single decision tree, where the user's input basically takes them out through the branch points of the tree to an end point that is the *answer* or advice. This type of simple logic is easy to implement in many ways, from linked HTML pages to virtually any programming language. However, most significant real-world decision-making tasks are not so simple. They require simultaneous consideration of multiple independent, sometimes competing factors, often with levels of *probability* or confidence that must all be folded together to produce the best advice. These are the types of systems that have much higher value since they can automate and disseminate knowledge and advice that would otherwise require access to a human expert. To build such systems requires separating the logic of the decision-making process from an *Inference Engine*.

We consider *expert system* to mean systems that perform a complex decision-making task through an inference engine that utilizes human readable rules describing the expert's logic and process. This dynamically drives a session that emulates an interactive conversation with the human expert to produce situation-specific advice tailored to the end user.

What is an Inference Engine?

Very simple logic can just be hard coded using nested IF statements in almost any programming language. However, complex, probabilistic logic becomes very difficult to hard code, and a maintenance nightmare due to the ripple effects of seemingly minor changes. The solution is to have a way to describe the steps and factors the expert uses to make a decision that is easy to read and maintain, which can also be used by a computer to drive an interactive session. The Inference Engine is the program

that does this. It allows the expert to write readable If/Then rules, similar to the way they would explain how they made a decision to another person. The inference engine processes those rules to determine:

- Which ones are relevant to what it is trying to determine
- What information it needs to evaluate the IF conditions of the relevant rules
- If there are other rules that can be used to derive the information it needs
- If necessary, what questions it should ask of the end user
- How to ask the questions of the end user
- Determining when it has enough information to reach a conclusion
- Presenting the conclusion (advice) to the end user
- Performing procedural operations as needed by the system
- Providing external interfaces to databases and other programs
- Using Resource Files for systems that run in multiple languages

In effect, the Inference Engine handles all the difficult issues that would have to be hard coded, and allows the actual decision-making logic to be stated in a relatively free-form way that does not require programming knowledge.

How are Exsys Expert Systems Built?

Exsys provides a well proven, proprietary inference engine implemented in Java. This inference engine is designed to work with the rules created using the Exsys Corvid® Development Tools. Corvid enables you to rapidly describe the expert's logic and process used to solve a particular problem. It provides an easy way to build and structure IF/Then rules in English (or any other language) and algebra. The rules are easy to create, read, understand and maintain. Exsys Corvid is designed to be simple to learn. The fundamentals of building systems can be learned in a few hours with on-line tutorials. (See how easy it is yourself, a free 30 day version of Corvid can be downloaded from: *http://www.exsys.com/download.html*). Corvid systems can be run using our Java based Exsys Inference Engine with a variety of end user interfaces, including Flash.

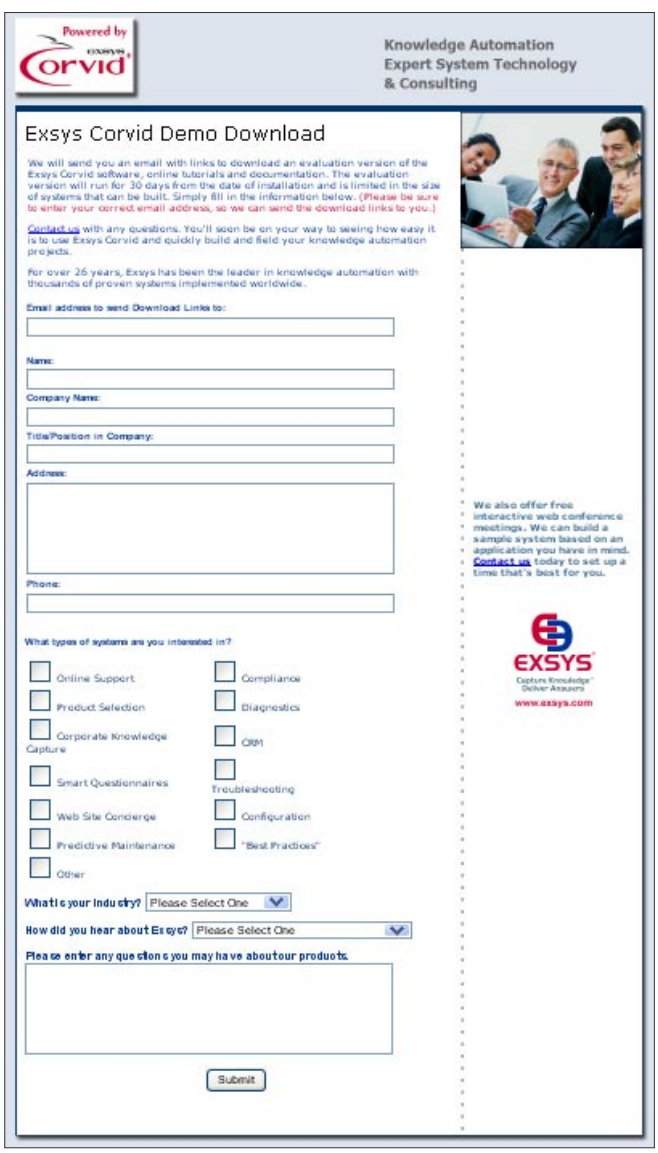
Who has Used Exsys Tools to Build Expert Systems?

Exsys has over tens of thousands of users worldwide including most of the Fortune 100 companies, government agencies, hundreds of universities and many types of organizations, which have built high value expert systems. Corvid is not a niche tool aimed at only one type of problem. It is a tool for describing ANY type of decision-making logic and it has been used in an amazingly wide range of domain areas. It has been used for diagnostics, pre-sales product advice, regulatory compliance, environmental, manufacturing, construction, help desks and many other areas. Systems include everything from how to irrigate peanuts for maximum yield, to real-time configuration of a Navy warship's electrical systems to take an incoming missile strike.

From in-house support systems to customer service subscriptions, to public facing interactive aids to commercial apps; the fielded application range is vast. Most of our customer's consider their Exsys systems such a strategic advantage they do not allow us to talk about them, but you can see what some of our users have done on our Case Studies page at *http://www.exsys.com/cases.html*

How is Exsys Integrated with Flash?

While expert systems can be embedded and invisible, most are interactive and emulate a conversation with a human expert. The inference engine dynamically



determines what information is needed based on information previously provided. Earlier answers may have made some questions unnecessary or have indicated areas that need to be examined in more detail. Since the expert system asks questions dynamically driven by the logic and process of the human expert, the sequence and depth of questions matches a conversation with the human expert. Once all the necessary questions have been asked, the system can provide situation-specific advice to the end user.

The processing of the rules and logic is handled by the Exsys Inference Engine, but there are many options as to how the questions and advice are presented to the end user. Exsys supports various options for the end user interface. Systems can even be designed to run in multiple languages. The simplest approach is to run systems on the client machine in a Java applet and ask questions in the applet window. Systems can also be run with a Java Servlet version of the Exsys Inference Engine. This can either generate HTML forms to ask questions or use Flash SWF files for the interface.

When using a Flash interface, the Flash program sends data to the Exsys Runtime (*Inference Engine*) by posting data to the Java servlet via a URL. The inference engine processes this data, and using other data and the rules in the system, determines what question to ask next. Exsys then sends information on what to ask back to the Flash program as XML data. This data can include all the information the Flash program needs to ask the question(s), such as prompts to use, allowed value options, limits, graphics, language options, etc. The Flash program takes this information and configures the interface to ask the question(s). It can even load a different SWF if needed

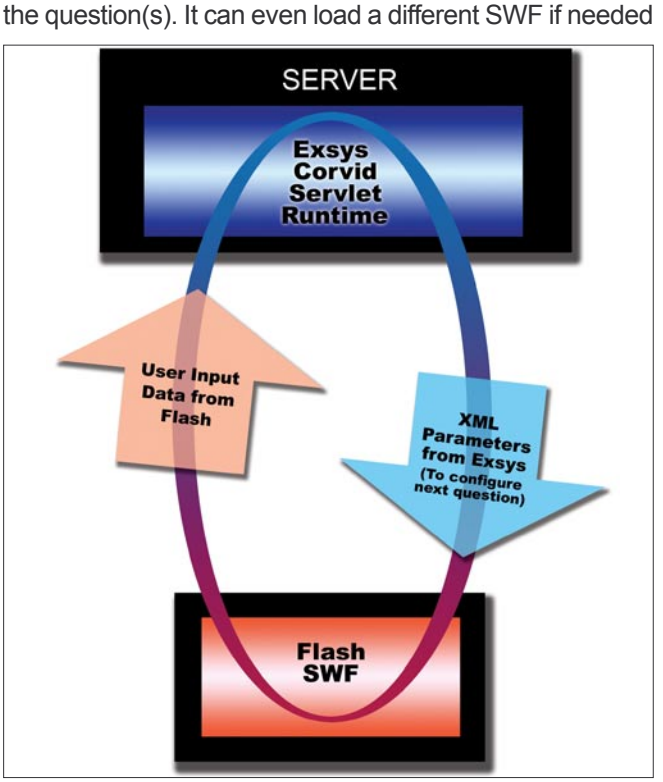


Figure 1. *Exsys Corvid Servlet Image*

to ask questions in a different way (Figure 1). When the end user provides their answers, this new data along with a session ID is passed back to the Corvid Runtime on the server, which will process it and send back XML for the next question. This repeats as many times as needed until the system reaches its conclusions. At that point the XML data sent back from Exsys can be used by Flash to display the results and advice to the end user – which could include graphics, animation, video, sound, etc. Since the individual servlet session is suspended while it waits for the end user interaction on the client side, this approach is highly scalable even for high demand systems.

One of the many advantages of this approach is that the Flash interface layer is completely separate from the system logic. Flash is used to create beautiful interfaces, which is what it does best. None of the decision-making logic has to be coded in Flash. That is all handled in the servlet-based inference engine. Changes in the logic only require modification of the Exsys Corvid system files on the server with no change in the Flash program. Likewise, If there are changes in the user interface, just modify the Flash program and the system logic does not have to be touched. Different development teams can work on each aspect independently without problems.

The same system can even be delivered in Flash and non-Flash versions. It only takes a few changes to modify the user interface that the Inference Engine uses. A system can easily be converted to use HTML forms rather than Flash – however, obviously HTML (even HTML5) does not offer the flexibility and power of Flash.

For the technical details of the Exsys/Flash interface see: *http://www.exsys.com/FlashDetails.html* (This document assumes a working knowledge of Exsys Corvid and the Exsys Servlet Runtime)

What Else Does the Exsys/Flash Interface Offer?

Using the Exsys Inference Engine under Flash provides many other capabilities besides traditional expert systems. Exsys is designed to allow complex logic and analysis to be easily described and offers many types of external interfaces. Flash systems needing to do complex logical processing that is difficult to do in Action Script, can off-load that job to the Exsys Inference Engine on the server. This is particularly true of systems that do pre-sales probabilistic product recommendation – a big competitive advantage capability enabling e-commerce sites to provide end users with expert analysis and recommendations of what product is best for individual site visitors. This type of logic is quite difficult to hard code, but Corvid has features specifically for this type of problem.

The Exsys Inference Engine is very powerful and easy to integrate with Flash. Instead of trying to program, debug and maintain decision-making logic in Action Script, a small Exsys Corvid program is a much easier solution.

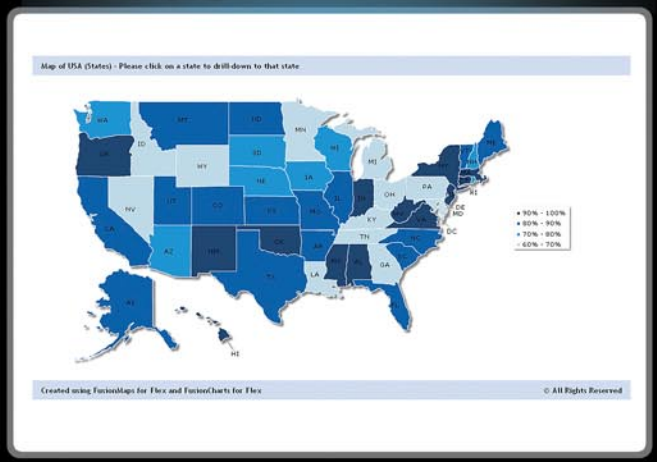
Need advanced data visualization in Adobe Flex?

FusionCharts helps you build stunning charts, gauges and maps for your Flex solutions in no time at all. Offering over 50 chart types and 300 maps, it offers you a powerful reporting experience for all your Flex projects.

FusionCharts for Flex

- Animated & interactive charts
- Over 50 charts & gauges
- True 3D capabilities
- Export as Image/PDF/CSV
- Extensive drill-down supported

FusionMaps for Flex



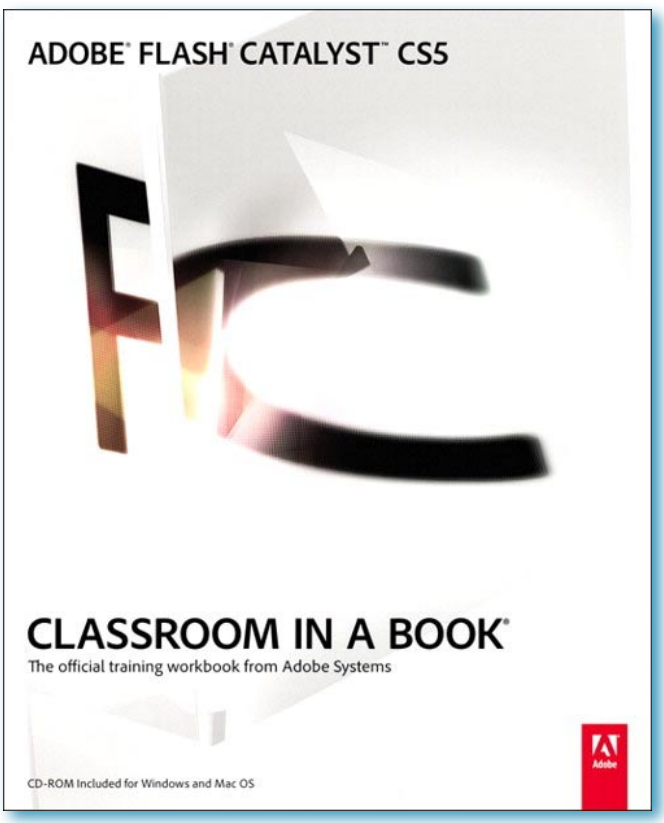
- Interactive & data-driven maps
- Over 300 maps provided
- APIs for extensive drill-down
- Export as Image/PDF/CSV
- Real-life demos & code samples

Download now from

www.fusioncharts.com/flex

sales@fusioncharts.com | www.fusioncharts.com

Adobe Flash Catalyst CS5
Classroom in a Book



Authors: The Adobe Creative Team
Publisher: Adobe Press
ISBN: 978-0-321-70358-3
Pages: 368
Website: *http://www.adobepress.com/bookstore/product.asp?isbn=0321703588*

What I admire the most about Adobe's classroom series is that they are not like traditional books containing hundreds of pages and doing a job that is meant for a documentation or a reference. Rather they take the reader alongside in a step by step manner and frame-up the skills while doing small amusing projects.

This book consists of thirteen chapters and discusses almost all the aspects of Adobe Catalyst CS5; from utilizing artwork to creating pages full of states and transitions. It also has a chapter on Audio/Video integration and few chapters concentrating on stuff that interests *RIA* designers. Catalyst made RIA development in hands of designers? Isn't the idea behind it contemplating? Anyways, it finally discusses some round tripping and most interesting subject (for me at least) taking projects to Flash Builder 4.

Only problem with this book is that I didn't find sufficient information about Catalyst-Flex workflow.

Concisely speaking, if you are designer and you want to do some RIAs, then Catalyst along with this book is the right thing for you. Though in my opinion RIA developers still do need a separate book for Catalyst-Flex workflow that targets large applications.

Table of Contents

by Ali RAZA
Adobe Certified Instructor
Sun Certified Java Programmer
Zend Certified Engineer