

AdvancED Flex Application Development

Building Rich Media X

R Blank
Hasan Otuome
Omar Gonzalez
Chris Charlton



AdvancED Flex Application Development: Building Rich Media X

Copyright © 2008 by R Blank, Hasan Otuome, Omar Gonzalez, and Chris Charlton

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-896-2

ISBN-10 (pbk): 1-59059-896-2

ISBN-13 (electronic): 978-1-4302-0441-1

ISBN-10 (electronic): 1-4302-0441-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

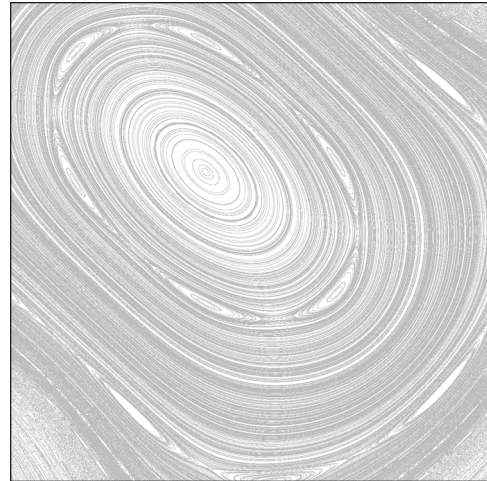
For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor Ben Renow-Clarke	Production Editor Jill Ellis
Technical Reviewer Lar Drolet	Compositor Dina Quan
Editorial Board Steve Anglin, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jason Gilmore, Jonathan Hassell, Chris Mills, Matthew Moodie, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh	Proofreader Lisa Hamilton
	Indexer Broccoli Information Management
	Artist Kinetic Publishing Services, LLC
Project Manager Sofia Marchant	Cover Image Designer Bruce Tang
Copy Editor Ami Knox	Interior and Cover Designer Kurt Krames
Associate Production Director Kari Brooks-Copony	Manufacturing Director Tom Debolski



Chapter 11

ADVERTISING AND FLEX

By R Blank

Advertising is a vital aspect of many Internet-based projects, and the RMX is no different. In this chapter, I will discuss some of the options available for banner and instream (or video) advertising, explain the problems with using most mainstream solutions inside of Flash-based applications, and show how we solved these issues for our project—utilizing open source technologies.

Why advertising matters

The RMX is free. Free to members and visitors, that is. In reality, the RMX—and any web-based application like it—costs real money, even when not accounting for our own time spent developing and maintaining the application, and especially when dealing with bandwidth- and storage-hogging video. But we, the owners, incur those costs and do not pass them on directly to our community.

One of the ways we try to make back some of that cost, as with most any widely trafficked site, is through advertising.

Although in the early days of the Internet, advertising failed to produce on the promises and expectations of many businesses and analysts, today advertising can make you a decent amount of money. This is especially true when the community consists of a highly specific and desirable market demographic—in this case, the Adobe user communities.

The mechanics of online advertising are quite simple but also quite powerful and varied. Someone wants to show an ad, so they buy space. Unlike television advertising, where space is based on channel, time, and geography, online advertising can be based on a much more complex set of variables—all entirely transparent to the user. You can deliver ads based on the content of a page (for instance, an ad for guitars along with a blog post review of a new Ovation). You can deliver ads based on the previous browsing history of the user (for instance, showing certain ads only to more frequent visitors or to members who have previously posted job opportunities on the jobs board). Or you can deliver ads based on reverse IP lookup (to get the geographical location of the visitor based on his IP address) or gender (based on a user profile the user has filled out). Or you can use a combination of all of these factors, and many more.

The goal is to deliver the most relevant ad that you can to that viewer at that point in time. This brings the most value not only to the advertiser, but also to the viewer. That is, consumers derive real value, and sometimes enjoyment, from exposure to more-relevant marketing messaging. And advertisers can get much more detailed information about the track record and success of individual ads and advertising campaigns—indeed, advertisers expect detailed metrics on their advertising. Any advertising management system or network will offer this type of data; it's one of the key reasons to use such a system instead of just building your own from scratch. Because, after all, all you're doing is loading media into a web page, and we all know a thousand ways to do that.

You can either consume ads from an existing third-party advertising network or sell your own ads. Using an existing network, while much easier, is frequently less lucrative and can provide less-relevant messaging to your visitors. One of the simplest options is Google AdSense (www.google.com/adsense), which is free to implement and use. Google gives you some code to insert in your site, and based on the words that Google sniffs in the pages in which that code is embedded, Google AdSense delivers contextually relevant advertising. If users click those ads, you get some money.

If you want to sell your own ads, you need an ad management system. It will help you manage advertising campaigns (with options like expiration dates and impression throttling, which ensures ads are only shown a certain number of times) and provide you the tracking metrics your advertisers will require. Many solutions are available on the market, from open source (read: free) to full custom ad networks (read: definitely *not* free).

To open source or not to open source?

As with most any similar decision, the verdict comes down to this: do you have money to spend, and are the open source alternatives usable? In the case of advertising on the RMX, the answers were “Not really, no” and “Yes.” The paid ad management systems like Accipiter, 24/7, and DoubleClick (now owned by Google) provide tremendous functionality and performance. At the same time, they can cost a lot of money, anywhere from \$1,000–10,000 a month and much more depending on your traffic.

So, for this reason, we chose one of the preeminent open source advertising campaign managers, OpenAds (www.openads.com). OpenAds (formerly known as phpAdsNew) is a pretty powerful and functional open source ad management system, well supported by its community with frequent updates.

I'll get into how we actually work with OpenAds in one moment. But first, I want to touch on a couple of additional aspects of online advertising that are very relevant for Flex and Flash developers to understand and consider when planning applications.

Flash and ads: Play nice, kids!

We all know that Flash has become an incredibly popular format for delivering online advertising. The ads can be incredibly cool and engaging, even at really small file sizes. Of course, you can also have video and audio seamlessly integrated with the advertising experience, with no additional plug-ins. Your ad can even be dynamic, pulling from an RSS feed, for example. And, with options like Eyeblaster (www.eyebalster.com) and PointRoll (www.pointroll.com), you can have user-initiated expandable ads. These expandable ads, always constructed and delivered with Flash, actually grow out of the standard banner area on user interaction (say, a click) to reveal a much larger canvas with all the functionality that Flash has to offer, including interactivity, animations, and even inline video. Expandables are really micro-sites or mini-applications that allow the viewer to participate with the brand and message in a meaningful and enjoyable way, without ever leaving the page he is viewing. This experience-rich type of advertising exploits the tremendous power of Flash, and a lot of Flash developers make a good living building these ads.

But, just because Flash is a great option for developing ads, it doesn't mean actually consuming ads in Flash is just as easy and popular. In fact, at Almer/Blank, we've had to chop up many an otherwise beautiful Flex and Flash application, just to make space for the frames and layers to hold the ads.

Why? Because almost every ad on the Internet is invoked with JavaScript or PHP. When you sign up for Google AdSense, you get JavaScript to paste into your pages. When you install and use OpenAds, you get JavaScript code to insert into your pages. This code is called an **invocation code** since the code loads, or invokes, an ad. And, while Flash can communicate with JavaScript and PHP, Flash can't directly load and interpret JavaScript or PHP, so you cannot have your advertising invocation codes in your Flash application.

Why not just utilize DIV layers to place the ads above the Flash? Unfortunately, that solution is unreliable cross-browser/cross-platform, since in some browsers, Flash will always render on top of all other content, regardless of depth.

In fact, the only really robust out-of-the-box option for Flex and Flash developers to integrate seamless ads into any application or web site is DART Motif Flash-in-Flash from DoubleClick (www.doubleclick.com/us/products/dart_motif_for_flash_in_flash/). But DoubleClick is the most expensive of the paid options, so it's totally out of consideration for all but the largest Internet presences.

So, as I said, at Almer/Blank we've had to chop some client applications that would have been perfectly delivered as single SWFs into as many as eight or nine SWFs in a page, just to support the ads.

What about instream ads?

Instream ads are video ads. They are often referred to as preroll and postroll ads (depending on whether they precede or follow the main video content). Any site planning distribution of significant amounts of video—especially Flash video—will want to consider delivering instream advertising.

And while all of the major ad management networks (such as the ones I mentioned previously) offer instream management and delivery, the problem with instream advertising is that the options for delivery management are far fewer than for banners. Google (at least at the time of writing—it's bound to change in the near future) does not offer a free instream advertising network the way it does with text banners with AdSense. You can get third-party instream advertising with a solution like

Brightcove's, but then you must use its player or API and host and deliver your content through that company. And OpenAds doesn't natively support the delivery of instream ads.

Our solution

When we started building the RMX, we had complete control of how the advertising would operate, so we decided to find a way around these two challenges. That is, we wanted to use the free and relatively powerful OpenAds, but we also wanted the flexibility to deliver ads to any part of the RMX, whether the specific RMX interface consuming the ads was built as HTML or Flash, and we wanted the same system that ran our banner delivery to also power our instream advertising.

So Daryl Bowden, one of our developers at Almer/Blank, came up with a solution to deliver ads (banner or FLV) from OpenAds into Flex and Flash applications. I want to share this with you in this chapter because, again, OpenAds is a pretty good and totally free solution, and this technique allows any Flex developer to offer a robust advertising solution along with his applications, pretty much out of the box for no cost. What's more, the same logic I'm about to explain can be used with most any ad management system that does not natively support delivery to Flash! (But you will have to modify the JavaScript and ActionScript for each case, because each system's code is different and utilizes somewhat different data, structure, and logic.)

So, first I'll show you how to set up OpenAds so that you have an ad management and delivery system in place, and then I'll demonstrate how to get into Flex to consume those ads. Let's dive into the details!

Setting up OpenAds

Before you get to the fancy code that powers our solution, you have to get set up to deliver the ads for this walkthrough. To that end, you need to have an environment that will support an OpenAds installation. Basically, you need a server that has PHP version 4.4.2 or higher installed, as well as MySQL, preferably version 3.23.2 or higher.

For a more in-depth list of requirements, you may visit <http://docs.openads.org/openads-2.0-guide/system-requirements.html>.

Now that you have an adequate setup, you will need to point your favorite browser to www.openads.org. When you get there, you will be greeted with a link on the right side of the page inviting you to download the latest stable version of OpenAds (which, at the time of writing, is 2.0.11-pr1, as you see in Figure 11-1). Go ahead and click that link, and your download will begin immediately. Many people have reported that the Max Media Manager (the newest development version at the time of writing) works incredibly well; however, I prefer to stick with the sure bet.

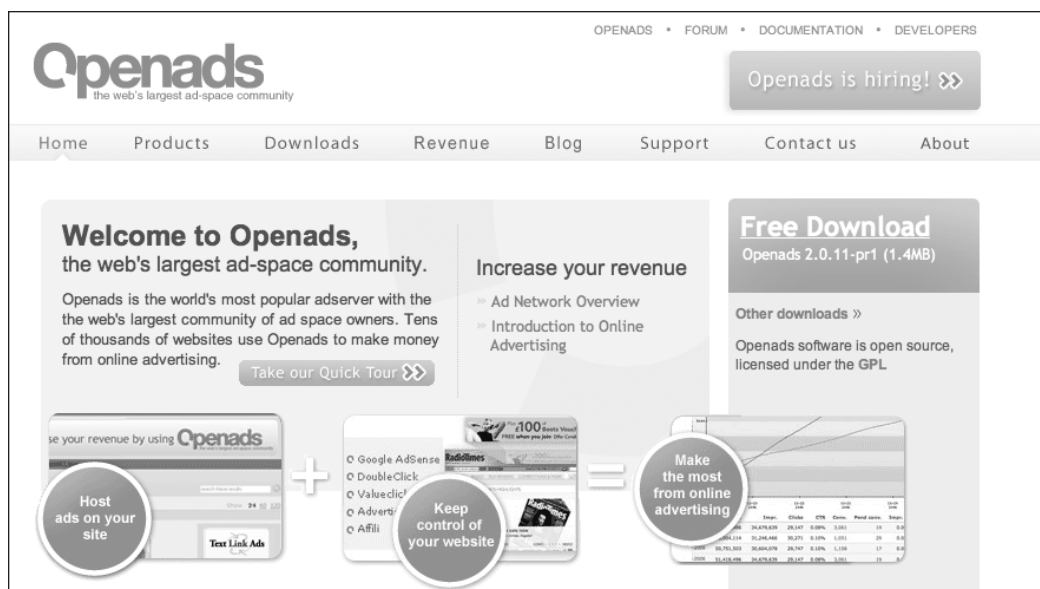


Figure 11-1. The OpenAds home page

Once the download has completed, you will need to extract the files onto your desktop. Then, open your FTP client and connect with your server. On your server, you will need to create a new directory to house your files; for this example, name it `/adserver`, which will be located at the web root. Once you have created this directory, copy all of the contents of the folder you downloaded into it. Now that the files are on your box, go to the `/adserver` folder on whichever domain you are using (such as `www.richmediax.com/adserver`) and you will see that OpenAds does all the hard work for you.

With your folder installed, you need to set up your MySQL database. If you have access to a web host control panel such as phpMyAdmin or Plesk, this will be a five-second job; if not, you'll need to use the command line. If you have trouble with this, you can find plenty of help at www.mysql.com.

Once you have completed installing OpenAds, it's time to get familiar with how it works. Almost anyone reading this book will find the administrative control panel easy to use; however, for less-tech-savvy folks, it can be a little difficult to get a grasp of exactly how it works. In either case, as with most open source applications, there is a huge user base out there just ready to answer your questions and give you whatever advice you may need.

You can find the OpenAds forums at <http://forum.openads.org/>. This is a great source for anything you might need relating to OpenAds.

To get into the guts of the application, simply point your browser once again to the `adserver` folder on your development domain, and you'll see the login screen pictured in Figure 11-2.

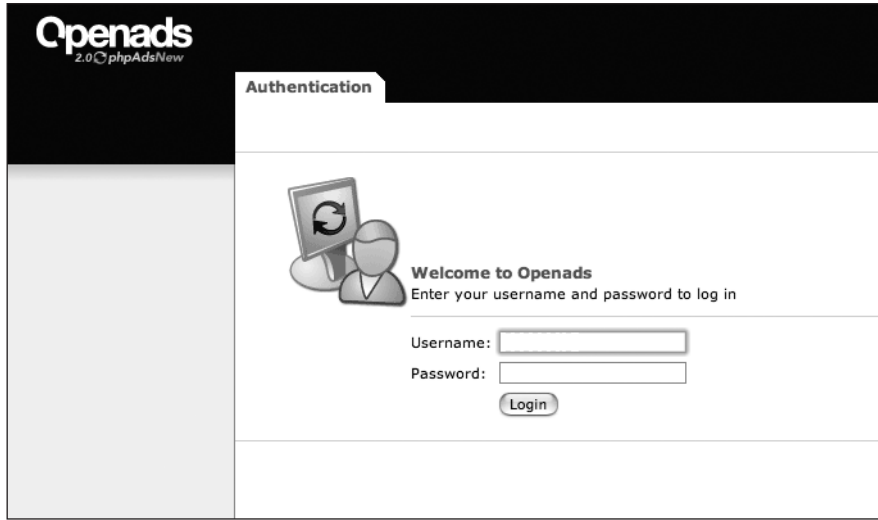


Figure 11-2. The login screen you should see after a successful installation of OpenAds

Once logged in, you will be taken to the Inventory screen (see Figure 11-3), which is the main screen for OpenAds and one that you will visit often. To get started, you first need to create a new advertiser. To do this, simply click the Add new advertiser link.

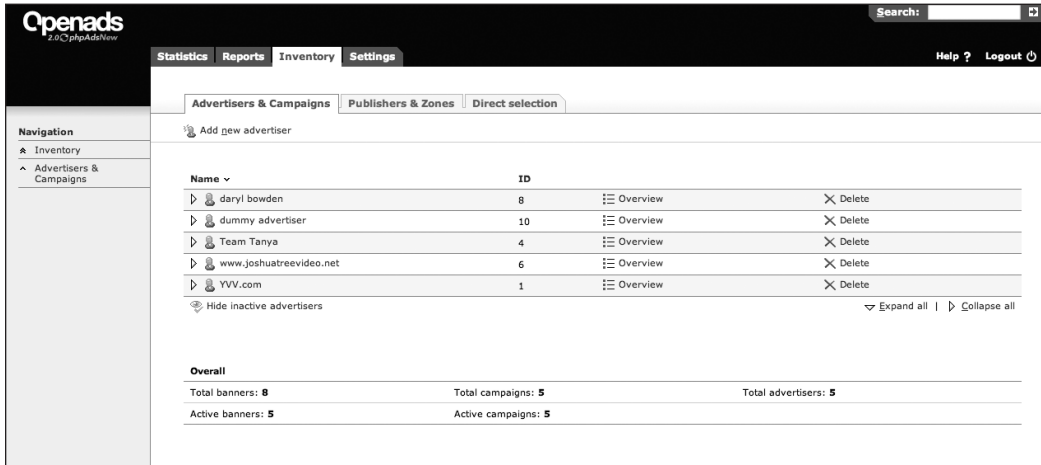


Figure 11-3. The OpenAds Inventory screen

On the Add new advertiser screen shown in Figure 11-4, you will assign your advertiser properties. You'll also notice that this advertiser can have its own login, which can be helpful if you would like your advertisers to access and modify their accounts directly.

Add new advertiser

Basic information

Name

Contact

E-mail

Language

Advertiser report

Send a warning when a campaign is deactivated

Send an advertising report via e-mail

Number of days between reports

Login information

Username

Password

Allow this user to modify his own settings

Allow this user to deactivate his own banners

Allow this user to activate his own banners

Figure 11-4. The Add new advertiser screen lets you specify properties for an advertiser.

Once you have filled in the requisite information, press the **Next** button to proceed to the next screen (see Figure 11-5), which will allow you to create a campaign.

A **campaign** includes a set of different ads along with the logic to deliver those ads, including start and end dates, maximum impressions, and priority. This page allows you to set up the start and end dates for the campaign (if there are any), as well as allows you to monitor the activity for this account (this page is also visible after the account has been created).

Once you're done here, click **Save Changes**, and then click the tab labeled **Banner overview**. Here you will stock your campaign with all the banners you need to get going.

Statistics Reports Inventory Settings

[id14] Almer/Blank Campaign > Untitled

Add new campaign

Basic information

Name Almer/Blank Campaign - Default

Contract information

AdViews remaining - Unlimited

AdClicks remaining - Unlimited

Activation date Activate this campaign immediately
- - -

Expiration date Don't expire this campaign on a specific date
- - -

Priority Distribute the remaining AdViews evenly over the remaining number of days. The target num
 Show banners in this campaign with high priority.
If you use this option Openads will try to distribute the number of AdViews evenly over the co
Limit the number of AdViews to - per day.
 Show banner in this campaign with low priority.
This campaign is used to show the left over AdViews which aren't used by high priority campa
Campaign weight: 1

Save Changes

Figure 11-5. You can create a campaign and set its properties on this screen.

From the banner overview page, simply click Add new banner to take you to the Add new banner screen (pictured in Figure 11-6), and you can start uploading your banners. As you upload each banner, you can enter the destination URL (which specifies the page the user will be taken to when he clicks the banner) as well as the target (the browser target, just as in the `navigateToURL` ActionScript method: either `_top`, `_self`, or `_blank` depending on the browser window in which you want the destination URL to launch). When you upload a banner, remember to give the banner a clear description, as this will make it easier to differentiate later on when you may have hundreds of banners in your database. Continue to upload banners until you have uploaded all the banners you would like to include for this campaign.

That's all you need to do within OpenAds for now, so next you can dig into the Flex side of this solution.

[id14] Almer/Blank Campaign > [id15] Almer/Blank Campaign - Default > Untitled

Add new banner

Please choose the type of the banner

Local banner (SQL) ▼

Local banner (SQL)

Select the image you want to use for this banner

Check for hard-coded links inside the Flash file

Destination URL (incl. http://)

Target

Alt text

Status text

Text below image

Transparent background (Flash-only)

Keywords

Description

Weight

Figure 11-6. The Add new banner screen

Consuming OpenAds in Flex

As cool as this solution is, the Flex side of the equation is really rather simple. You need a little bit of ActionScript, one line of MXML, and you're done.

To get started, create a new Flex project. In your application file (I've called mine `openAds.mxml`), begin with the `ExternalInterface` class. This class enables your SWF to talk to its wrapper so that it can call a JavaScript function located on the HTML page in which your SWF is embedded. Not only that, but it also allows you to return data back to the SWF to be further manipulated by your Flex code. In fact, this class also works the other way around: you can also use it to call Flex functions from within the JavaScript.

I will show you all the Flex code involved and then go through it step by step with you.

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" creationComplete="callWrapperBanner()">
<mx:Script>
  <![CDATA[
import flash.net.navigateToURL;
import mx.controls.Alert;
import mx.collections.XMLListCollection;
import flash.external.*;

[Bindable]
public var adReturn:*;
public var javascriptReturn:XML;

[Bindable]
public var imageSource:String;
[Bindable]
public var imageClick:String;

public function callWrapperBanner():void
{
  //check to see if external interface is available
  if(ExternalInterface.available)
  {
    //callOpenAds is the name of the JS function
    //contained in the wrapper
    var wrapperFunction:String = "callOpenAds";
    //make the call to the wrapper and the JS function
    adReturn = ExternalInterface.call(wrapperFunction);
  } else{
    Alert.show("Failed to initiate external connection");
  }
  //convert HTML to XML
  var img:String = "<root>"+adReturn+"</root>";
  //correct malformed HTML that comes back from OpenAds
  img = img.replace("></A", "></A");
  img = img.replace("></DIV", "></DIV");
  //create new XML object and use that object to
  //parse out the tags you need
  javascriptReturn = new XML(img);
  imageClick = javascriptReturn.A.@href.toString();
  imageSource= javascriptReturn.A.IMG.@src.toString();
  //add event listener so that click-through and link still works
  // the way the user expects
  adImage.addEventListener(MouseEvent.CLICK, adClick);
  adImage.buttonMode = true;
}

```

```

private function adClick(event:MouseEvent):void
{
    //set variable to contain destination path for ad
    var ur:URLRequest = new URLRequest(imageClick);
    //send user to link on click
    navigateToURL(ur);
}]]>
</mx:Script>

<!-- Set source to the bindable variable imageSource which
contains the img src of the return from the JS -->

<mx:Image source="{imageSource}" id="adImage" />
</mx:Application>

```

To begin, you declare a few variables. First, you declare two strings, `imageSource` and `imageClick`, and both must be declared as `[Bindable]` since you will use these as the data source for both your `Image` component and your `click` event. The other two variables are `adReturn`, which will hold the return value from the JavaScript function in the HTML, and `javascriptReturn`, the variable you will convert to XML. `javascriptReturn` does not need to be bindable, as you will not be using it outside of this function.

You'll note that this file only contains two functions: one to call the ad and another to enable the click. The main function, `callWrapperBanner()`, uses the `ExternalInterface` class that I referred to at the start of this section. To set this up, you first set up a conditional statement to ensure that the `ExternalInterface` is available (meaning that JavaScript is enabled in the viewer's browser). Assuming this tests true (meaning JavaScript is available, which it will be about 96% of the time), you can proceed. If JavaScript is unavailable, this solution won't work—but in those cases, you wouldn't have been able to load ads into the web page anyway, regardless of whether the interface is Flash or HTML, since the ad can never be invoked by the invocation code.

Knowing that JavaScript is available, you assign your variable `wrapperFunction` the string `"callOpenAds"`. The naming of your variable is a very important step. The name that you give this variable needs to be the same as the JavaScript function you are going to create on your wrapper page; otherwise, the communication won't work, and your ads will not render correctly. When done, your conditional should look like this:

```

//check to see if external interface is available
If (ExternalInterface.available)
{
    //callOpenAds is the name of the JS function
    //contained in the wrapper
    var wrapperFunction:String = "callOpenAds";
    //make the call to the wrapper and the JS function
    adReturn = ExternalInterface.call(wrapperFunction);
} else{
    Alert.show("Failed to initiate external connection");
}

```

Next, you parse through the information that is returned to you from the wrapper and contained in the variable `adReturn`. You will prepend it with `<root>`, append it with `</root>`, and store it in the local variable `img`, typed as a string. This will convert the HTML that was returned into valid XML, so that your Flex code can access the relevant information.

This is where you run into the biggest issue with the OpenAds delivery system. The HTML it uses for displaying images is malformed (shh . . . don't tell anyone); the `` tag in the OpenAds-generated HTML does not include a proper closure, and therefore it cannot be recognized as XML by Flex without some modification. Fortunately, this failure (or "feature," I suppose) is a consistent one, so to remedy this, we use the `replace()` method, which allows us to parse through a string and replace a specified substring with another string value. When calling `replace()`, you pass the substring you wish to replace as the first parameter and the string to insert in its place as the second parameter. By looking through the returned information, you'll see that the `` tag should be closed right before the `<a>` tag closes. Here, you can see the method with the proper syntax, as used in this example:

```
img = img.replace("></A", ">/></A");
```

Now that you have corrected the form of your return, you can treat it as XML and parse through it. Take the `javascriptReturn` variable that you declared earlier and set it as a new XML object with the local variable `img` as the source. Now that you've converted the return value into valid XML, you can grab the hyperlink from the `<a>` tag in the return. To do this, assign your `imageClick` variable the value of the link, like so:

```
javascriptReturn = new XML(img);  
imageClick = javascriptReturn.A.@href.toString();
```

Here you grab the `href` attribute of the `<a>` node from our XML. Appending the call to `toString()` converts the data to a string value so that you may use it as the destination for a `navigateToURL` call. Next, you'll perform the same operation on the source attribute of the `` tag to grab the media source, like this:

```
imageSource= javascriptReturn.A.IMG.@src.toString();
```

Once you have the source for the banner, you'll hop out of the ActionScript and into your MXML, which consists of only one line:

```
<mx:Image source="{imageSource}" id="adImage" />
```

This is simply an `<mx:Image/>` tag, with an `id` of "adImage" and the source set to the bindable value of the `imageSource` variable. And now you've displayed your ad! And, if this were an instream ad, rather than a banner, your one line of code would look almost identical:

```
<mx:VideoDisplay source="{imageSource}" id="adImage" />
```

Next, you need to set up the `click` event that will allow the image to act as a user expects a banner to act. To do that, you'll jump back into the `callWrapperBanner()` function and append these few lines at the end:

```
adImage.addEventListener(MouseEvent.CLICK, adClick);  
adImage.buttonMode = true;
```

The first line simply adds an event listener to your image. When the image experiences a click event, it will fire the `adClick()` function. So, next you'll write that callback function. Like all of the Flex code so far in this chapter, `adClick()` is a very simple function consisting of only two lines:

```
private function adClick(event:MouseEvent):void
{
    //set variable to contain destination path for ad
    var ur:URLRequest = new URLRequest(imageClick);
    //send user to link on click
    navigateToURL(ur);
}
```

The first line within the function creates a new `URLRequest` object using the `imageClick` variable as the source. The next line simply redirects the user to the destination when the event is fired.

Finally, I would like to point out one very important line of code to you. In the opening `<mx:Application>` tag, you'll see a `creationComplete` event that calls the main function `callWrapperBanner()`.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="callWrapperBanner()">
```

Now you can compile your Flex application, since you have finished your Flex work (I told you the code was going to be easy); so to finish the job, you'll move on to the very minimal JavaScript this project requires.

To keep things easy, you're going to use the default HTML page that Flex creates when you compile your SWF. You can use any HTML editor to edit the file (I generally prefer to stay within the Eclipse environment, so I use the JSEclipse editor, which can be downloaded and installed from www.eclipse.org).

The first task for the JavaScript is to grab your invocation code—the code that returns the banners to you from OpenAds. To grab this code, you'll log back in to the OpenAds installation and click the Inventory link in the left sidebar. On the Inventory screen, click the Direct selection tab. Once you've gotten to the Direct selection screen (pictured in Figure 11-7), you will see a drop-down menu with a default state of Local Mode. Click that drop-down menu and select Remote Invocation for JavaScript. Then select the campaign you created from the Campaign drop-down menu. Once you have selected everything, click the Generate button at the bottom of the page (the button will be labeled Refresh once you've generated the code the first time). Now, you will see that the Banner selection box is populated with the invocation code. You'll grab that code by selecting it and copying it to your clipboard.

The screenshot shows the 'Direct selection' tab in OpenAds. At the top, there are three tabs: 'Advertisers & Campaigns', 'Publishers & Zones', and 'Direct selection'. Below the tabs, the text 'Please choose the type of banner invocation' is followed by a dropdown menu set to 'Remote Invocation for Javascript'. A section titled 'Bannercode' contains a text area with the following code:

```
<script language='JavaScript' type='text/javascript'
src='http://dev.yourvideoviews.com/adserver/adx.js'></s
<script language='JavaScript' type='text/javascript'>
<!--
  if (!document.phpAds_used) document.phpAds_used =
  ',';
```

Below the code is a 'Parameters' section with several fields and radio buttons:

- Banner selection:** An empty text input field.
- Campaign:** A dropdown menu showing '[id13] Almer/Blank - Default'.
- Target frame:** An empty text input field.
- Source:** An empty text input field.
- Show text below banner:** Radio buttons for 'Yes' and 'No', with 'No' selected.
- Don't show the banner again on the same page:** Radio buttons for 'Yes' and 'No', with 'No' selected.
- Don't show a banner from the same campaign again on the same page:** Radio buttons for 'Yes' and 'No', with 'No' selected.

At the bottom left of the form is a 'Refresh' button.

Figure 11-7. The Direct selection tab in OpenAds, from where you can grab your ad invocation code

Once you have copied the code, open up the default HTML file that Flex created for you. In this file, you will create a new DIV within the body of your document and paste the invocation code into it.

```
<div id="advertise" style="display:none; visibility: hidden;">
<script language='JavaScript' type='text/javascript'
src='http://dev.yourvideoviews.com/adserver/adx.js'></script>
<script language='JavaScript' type='text/javascript'>
<!--
  if (!document.phpAds_used) document.phpAds_used = ',';
  phpAds_random = new String (Math.random()); phpAds_random =
  phpAds_random.substring(2,11);
  document.write ("<" + "script language='JavaScript'
type='text/javascript' src='");
  document.write ("http://www.fakeserver.com/adserver/adjs.php?n=" +
  phpAds_random);
```

```

    document.write ("&clientid=7");
    document.write ("&exclude=" + document.phpAds_used);
    if (document.referrer)
    document.write ("&referer=" + escape(document.referrer));
    document.write ("'">" + "/script>");
//-->
</script><noscript>
<a href='http://www.fakeserver.com/adserver/adclick.php?n=a45038aa'
target='_blank'>
<img src='http://www.fakeserver.com/adserver/adview.php?clientid=7&
n=a45038aa' border='0' alt=''></a></noscript>
</div>

```

The only modification you need to make beyond pasting the code into your new DIV is that to assign the DIV an id of "advertise"—this id is very important to keep track of, as you will shortly reference this DIV layer by its id in your code.

Now that you have created your DIV and pasted in the invocation code, you will open a new JavaScript script block in the head of the page. In this block, you'll define one function, callOpenAds()—this should sound familiar from the ActionScript you programmed just earlier. This function body consists of only two lines:

```

<script language='JavaScript' type='text/javascript'>
<!--
function callOpenAds()
{
    var temp = advertise.innerHTML;
    return temp;
}
//-->
</script>

```

The first line utilizes a built-in JavaScript property, which will grab all of the HTML from inside the DIV that you just created. This is why it was so important to name your DIV—each HTML element in a document has an innerHTML property, which may be accessed by JavaScript to grab the contents from the element (in this case the DIV) and return it to your local variable, temp. In the next line, you return temp back to your Flex application.

Once these lines have been added to your default HTML page, you are done; simply upload the contents of your bin folder to your test site and browse to your HTML page. You'll see your ad appear on the page inside of your Flex SWF. If you go ahead and click it, you'll be transported to whatever site you set the target to during your campaign setup.

Voilà! You can now sell and deliver ads however you want!

Really, how powerful is this solution?

No doubt, this solution is simple. At least, the required ActionScript and JavaScript code is pretty low level. But that doesn't mean it isn't powerful. As I indicated earlier in the chapter, this technique can

be used with almost any ad management system that does not natively support Flash. I used OpenAds in this example because it's a free and easy-to-use solution that you can utilize in your projects. But, if a client wants to support standard or instream ads inside a Flash application off of an existing ad management system, in most cases, all you'll have to modify from this solution is how you parse the returned data. However, you must check with the terms of service of the ad management system to ensure your utilization of their system is permitted. (Click fraud is a very serious consideration in online advertising, so some networks are more restrictive than others in how they allow you to utilize their services; indeed, that might be one more reason to consider OpenAds.)

Now, I say this will work with “almost” any system, because it doesn't work with all; in fact, it doesn't work with Google AdSense, which, of course, is a major network. That's because this solution allows you to use application-native invocation codes written in JavaScript or PHP to *invoke* an ad. But this solution still relies on the ad that is returned from the invocation to itself be in a format that Flash can understand—which includes SWF, FLV, and H.264 video; JPG, PNG, and GIF (nonanimated); and the limited subset of HTML that Flash TextFields can interpret. Google AdSense returns HTML that cannot be understood by Flash, and therefore Google AdSense cannot render in the Flash Player.

Beyond the banner

I already illustrated how this solution works not only for banners, but also for pre- and postroll instream ads. But, by adding your own code logic to supplement what you've done here, you can support most any type of advertising model.

An **overlay** ad is an ad that superimposes some other media, usually a video, without pausing or otherwise affecting the main content. Overlays often include some transparency (whether it's SWF, PNG, or GIF), and if they are Flash ads, they can also include interactivity. Since you can build your own Flash overlays, you can even have an overlay interact with the video player; for example, you could integrate interactivity into the overlays, and if the user does interact with the ad, pause the video playback. And like the expandable ads I mentioned previously, overlays can also grow over the content they superimpose. Since overlays cover other content, they are less likely to be ignored by viewers.

As an example, let's say you wanted to load an overlay ad on top of a video starting from when the viewer is 30 seconds into the video and display it for 30 seconds. First, you could create a new campaign in OpenAds to manage your overlays (separately from your prerolls, for example), and then upload some custom SWF content to serve as the overlay media. You'd have to paste the invocation code for the new campaign into your HTML, as you did for your banners and instream ads earlier in the chapter.

Then, you would add some additional logic to your Flex application so it would know when to call an overlay. For instance, you could create a `Timer` object to call a function that pings the current time-code of the video being played back. When the viewer is at 30 seconds, the function would call an ad from the overlay campaign, as you did with your banner earlier, and display it in an `Image` component. The `Timer` callback can then hide the ad after it's been displayed for 30 seconds.

Using the same type of logic, you can also support interstitials. **Interstitial advertising** is advertising that occurs in the middle of some other content, again usually video. Just as a preroll ad plays before your main video content and a postroll plays after, an interstitial plays at a certain point during the video, pausing the video while it is visible. For example, you could interrupt a three-minute video after two-and-a-half minutes and display an ad (generally of any type supported by Flash—though they

tend not to be as interactive as overlays or expandables). This would work just as I explained for overlays, except your Timer would pause and hide the video to display the interstitial, and then show and resume the video when it hides the ad.

Advertising in AIR

You might be wondering if any of this applies to Flex application development for AIR. Well, the good news is, yes it does. But, because of the power of AIR, you don't even need it.

AIR includes a web browser with JavaScript support. So AIR applications can run HTML and JavaScript as well as Flash. In fact, the root of your application can be HTML or it can be Flash. Therefore, very similarly to embedding Flex-generated SWFs into a web page, you can embed your Flex-generated application inside an HTML page and export that as an AIR application. That HTML shell can include the JavaScript and DIV layers you used in the web-based example, and the Flex-generated application can call the JavaScript in the same way. You can even build in some additional support if the user of the AIR application is offline while running the application; for example, you can pull from some locally stored ads, precached on the user's machine, or display some other preset content (such as animated logos or "Did you know?" help content to assist the user with the application).

However, recall the discussion much earlier in the chapter on why I've needed to chop up Flex applications in the past to display ads. When Flash content runs on the Web, it can only interpret a very small subset of HTML, and you can only display that content in TextFields—it can't load HTML content into a frame within the Flash application itself. Well, with AIR, you can create frames for HTML content within your Flex application; so you can have a display area the size and location of your advertising that displays HTML content which includes the logic to load and display ads. Your Flex application can then communicate via the ExternalInterface with a JavaScript function that would trigger the display and refresh of those ads, without going through all the bother of having Flex parse the results, display the ad, and manage the clicks. You can even support Google AdSense this way!

Summary

I started this chapter discussing some of the core considerations of delivering ads in your Internet applications. I then covered a specific technique to bring ads—both banners and instream—into your Flex applications with some really simple code and some free software. I really love techniques that are both simple and powerful. And just to reiterate, this same logic can be applied to ads delivered from any ad management system that doesn't natively support Flash Player delivery.

Continuing with our utilization of powerful, community-supported open source solutions, in the next chapter Hasan is going to introduce you to the basics of working with Drupal, an open source content management system, showing you how to build a blog in minutes. After that, just as I demonstrated in this chapter how to harness OpenAds for Flex, Hasan and Omar will show you how to build Flex applications off of a Drupal-powered CMS.

