

# 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](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com).

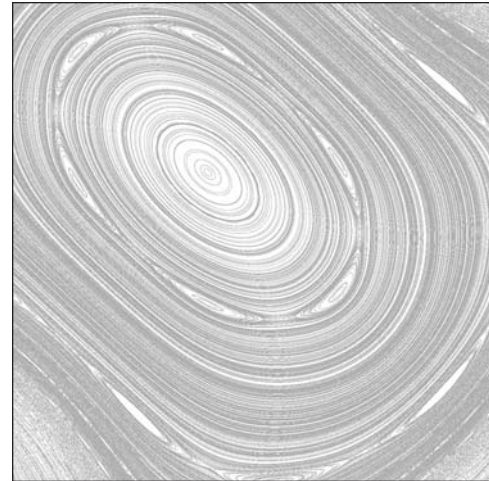
For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail [info@apress.com](mailto:info@apress.com), or visit [www.apress.com](http://www.apress.com).

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](http://www.friendsofed.com) in the Downloads section.

## Credits

<b>Lead Editor</b> Ben Renow-Clarke	<b>Production Editor</b> Jill Ellis
<b>Technical Reviewer</b> Lar Drolet	<b>Compositor</b> Dina Quan
<b>Editorial Board</b> 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	<b>Proofreader</b> Lisa Hamilton
	<b>Indexer</b> Broccoli Information Management
	<b>Artist</b> Kinetic Publishing Services, LLC
<b>Project Manager</b> Sofia Marchant	<b>Cover Image Designer</b> Bruce Tang
<b>Copy Editor</b> Ami Knox	<b>Interior and Cover Designer</b> Kurt Krames
<b>Associate Production Director</b> Kari Brooks-Copony	<b>Manufacturing Director</b> Tom Debolski



## Chapter 6

# COLLECTING AND WORKING WITH AGGREGATED CONTENT

---

by Chris Charlton

Most RIAs are complex—sometimes really complex. And in the context of a social media network, the ways in which you utilize content are a vital aspect of your application. The ability to generate, store, and access a wide variety of content types, from a large number of users, across a wide variety of platforms, is a requirement of any application in this space. As we all know, content is data, and managing that data properly is a fundamental requirement of any well-engineered application.

There are, of course, a huge number of topics that I could cover in this chapter. I've chosen to highlight a few of the key data features of the RMX—ones that would be applicable to many different applications—including how we handle data schemes for tagging, sharing, syndicating, and aggregating content from both external and internal sources. Those data sources, whether internal or external, can come from web services, XML, server scripts, relational databases, or any combination of these.

## Data sources

In planning the RMX, we knew up front our most important data source would be our relational databases. We chose MySQL knowing it's an impressive work horse that powers the back end of many popular Web 2.0 sites like Digg, Flickr, YouTube, Wikimedia (Wikipedia), and the web pages for NASA, and it's the backbone of our content management framework, Drupal.

## Databases

In an ideal scenario, there'd be a helpful, happy database administrator (DBA) who has the honor of overseeing database management full time. If there is no DBA, you should at least have someone in a data architect role to create a data model that is efficient and scalable, as oftentimes interface developers do not have either sufficient skills or sufficient grasp of the overall project requirements to execute this role properly. Instead, interface developers usually care more exclusively about how to interface with the data, rather than how to structure it. Usually developers interface with databases in the form of services or what's commonly known as an application program interface (API). However, in recent years, the term "API" has become easily synonymous with Web 2.0 as it should, since it promotes syndication by allowing interaction with content in any fashion. Any simple, well-documented set of services (or methods) is good enough to be considered your API, just don't tell the marketing team.

Your choice of database shouldn't have an impact on your overall database structure, but it will impact details of that schema, such as data and data types. Databases are usually interfaces with a database management system or DBMS. All DBMSs—MySQL, Microsoft SQL Server, SQLite, and so on—have standard and nonstandard features, usually to satisfy developer or enterprise requirements or performance boosts. Oftentimes though, following the simple route with your initial data design is best since that simple base can be built on top of. Don't waste time trying to gauge or answer questions of model complexity, just design. As a rule of thumb, keep all database designs simple; they'll grow before you know it. RMX started out simple.

### MySQL

For the RMX, we knew we'd be running it on a LAMP (Linux, Apache, MySQL, and PHP) stack, MySQL being our database software, of course. MySQL hosting is available practically anywhere around the globe, and it's so good at its job that you can get good results from basic usage, and great results in terms of scalability. Those who get to expand their skill into the realm of higher relational database concepts and software functionality will agree that MySQL has web app written all over it.

MySQL is free, which makes it a potential winner for any project. The software itself runs databases and some tools to interact and administer databases. As for extensive tools for MySQL, we look to the large array of tools that third-party software makers develop. The MySQL group provides support services and has done so for years. Their recent addition of network tools and enterprise-level monitoring shows businesses that MySQL isn't just a cool database for web geeks but a serious contender in commercial and enterprise markets. But, we geeks can grin, since the "cool factor" does follow MySQL, powering the most popular Web 2.0 services and sites today.

MySQL itself has a really good set of features like query cache, custom functions, stored procedures, and triggers. While we don't cover these features in this book, you can easily find information on them in the many books on the market about MySQL, SQL, and relational databases. What we do cover can generally be applied to any database software you want to run; if we cover something that is MySQL specific, we will point it out.

*Apress carries MySQL books for all levels, from beginner, to pro, to expert, even definitive guides. We suggest you check any or all MySQL titles out at [www.apress.com](http://www.apress.com), since we know your MySQL usage may become habitual.*

## XML

XML is a great standard because it can be as loose or strict as we need it to be and it's cheap. Better than CSV and proprietary TXT files, XML files are solid data sources. Full document formats themselves are all XML. Some have been for a long time now, like Microsoft Office and the newly supported OpenDocument format. Since XML is the parent of all these formats, it's fitting that XML is also parent to every syndication format that exists on the Web today.

The Flex framework has really good XML support. Not just loading XML, but also chewing through XML with its E4X features, like XPath, that make working with XML a breeze. You can generate XML directly in ActionScript or with MXML. Most web languages or databases should have basic XML support, so applying XML throughout your application shouldn't be a problem.

### XML structures and schemas

The **XML structure**, also known sometimes as a **schema**, is important. When you're designing your XML structure, start by just coming up with a simple idea of XML tags or groups of tags and their interrelationships. Always remember, don't overdo it with your schemas; they can and should remain simple.

### XML web feeds

It was mentioned that XML is the parent of every syndication feed on the Internet. RSS seems to be winning popularity for web site syndication, especially for publishing frequently updated content, such as news headlines and blog entries. Adoption of web feeds like RSS made it possible for people to keep up to date with their favorite content by subscribing to it and being notified as new content becomes available. The software checks periodically, as often as people like—daily, monthly, or even hourly. Its popularity is new, but RSS itself is not. There are officially different versions of RSS, but RSS 2.0 is currently the best, as it allows custom XML extensions.

FeedBurner ([www.feedburner.com](http://www.feedburner.com)) is one syndication company that uses XML extensions extensively. This company is famous for providing feed syndication stats, but the heart of its output relies on XML extensions, since it takes any feed you give it and dresses it up with fancy extensions from Yahoo (MRSS), Apple (iTunes), and dozens more. Since XML extensions are really easy to support and adopt, FeedBurner even allows anyone to submit an extension and instantly adds it to its service offerings.

ATOM, RDF, and OPML are other popular XML-based feeds. OPML complements RSS, while the others are alternatives to RSS. Offering these formats on your web site or application is not required but helps complement your syndication platform.

## Sharing outside the network

There are a few ways that networks and services allow you to share content outside their domain—web links (permalinks), send-to-friend e-mails, and embed code. When you're planning to provide sharing features like these, each has their own set of rules.

### Sharing permalinks

A URL is easy to pass around. It can be passed via instant messaging, e-mail, web sites, blog posts, and even verbally or in broadcast and print media. **Permalinks** provide unique URLs assigned to content for

people to revisit and refer to that content as long as the content is available. The term comes from the phrase “permanent link.” Permalinks are most commonly noticed on blogs, since they’re a definitional requirement of blogs and a fundamental feature of the blogging community (in which bloggers often cross-link), and when you follow a link to a specific post, you are following a permalink. These permanent links work as unique keys. We use the unique keys to generate a permalink to serve the specified content.

The magic of permalinks is done on the server by Apache with what are known as **rewrite rules** using a module called `mod_rewrite`. We retain unique keys in a database for each piece of content and search against those, allowing us to offer short URLs that Apache internally rewrites or redirects to content. It’s much easier to remember a shorter web link like `http://videos.richmediax.com/abc123` versus `http://videos.richmediax.com/?Population=25423642&category=Vegetation&content=abc123` or some other meaningless gibberish, which is the default behavior of many content management systems like Drupal. You can usually recognize a rewrite URL by its lack of query parameters. These types of links are also known as **clean URLs** or **search engine–friendly URLs**. Rewrite rules can be placed in a text file known as an **htaccess** file. Following is a sample of a `mod_rewrite` rule that takes a URL and checks it against a rewrite rule, redirecting the user’s request when the rule is met.

```
Options +FollowSymLinks
RewriteEngine on
RewriteRule player/media/(.*)/embedded/(.*) ➡
player.swf?media=$1&embedded=$2 [R]
```

The preceding code must be saved in a `.htaccess` file in any folder on your server. The rule shown, `RewriteRule player/media/(.*)/embedded/(.*)/ player.swf?media=$1&embedded=$2`, has two parts. The first part, which I’ll call the **rule** (or **test**), is `player/media/(.*)/embedded/(.*)`, and it checks all URL requests sent to the server. It will notice any requests that follow that rule, which requires there be some value where each `(.*)` piece appears. The second part, which I’ll call the **rewrite portion**, is `player.swf?media=$1&embedded=$2`. The `$1` and `$2` are numbered “tokens” that get populated or replaced by the characters that replace `(.*)` in the rule. If you’re familiar with regular expressions, like in PHP or ActionScript 3, then Apache’s usage of them for rewrite rules and tests should seem familiar.

Permalinks have been around longer than Web 2.0, so it shouldn’t be hard to find out if your server has `mod_rewrite` capabilities to try out a couple of rules yourself. Since rewrite rules use regular expressions, that skill set is required to pull off more complex rewrite rules. Again, `mod_rewrite` is an Apache module, so if you plan to use Windows as your server platform, you’ll have to get the `IIS_rewrite` engine.

*Flash Player has support for regular expressions, meaning your Flash, Flex, or AIR applications all have this powerful yet complex language for searching, matching, filtering, replacing, or any other crazy rocket science we developers come up with.*

## Sharing by e-mail

E-mail is the oldest form of web sharing. Much of the time, folks will pass a link in a digital or printed form, most commonly via e-mail. With e-mail communication, it is important to remember that there

are two basic types of e-mail: HTML and text. Sending e-mail, we're able to send both HTML and text **bodies** in a single message. Also known as **MIME types**, these message bodies have a lot of freedom and a lot of restrictions. HTML e-mails cannot contain JavaScript, because this is a security concern. HTML e-mails can, however, have images and CSS styling embedded or linked externally. Embedded imagery and styling add a lot of file size, and the greater the file size, of course, the worse the user experience. By externalizing the images and CSS, we trim the size considerably while still offering a stylized experience should the user accept our HTML content.

Bringing things around, links in e-mails sent from your site, like a "send-to-friend" feature, should preferably consist of permalinks. First, this can reduce the fear that software or a service may force linefeeds in the middle of a URL. This would render the URL practically useless. Also, by utilizing your own permalinks in shared e-mails, users will end up becoming familiar with these link schemes and they'll promote them on their own.

Flex has no way to send e-mails without the help of a server component, like IMAP, POP, and so on. ActionScript 3 has the power to talk to these and many protocols, none of which are small feats, so you'll probably want to turn to a server-side scripting language like PHP. While PHP has a built-in `mail()` function that makes things easier, it does require a little attention to get good e-mails sent out easily. To cut out the headaches of dealing with MIME types from scratch, there is a free PHP class called `eMail`, which is available online at [www.phpclasses.org/trackback/browse/package/1644.html](http://www.phpclasses.org/trackback/browse/package/1644.html). Here, you'll see how to use this PHP `eMail` class and call it from Flex using the `RemoteObject` tag:

1. After installing or uploading the `eMail` PHP class, you set up a function:

```
<?
include("class_mail.php");

// POST vars sent from Flex
if (isset($_POST) && ($_POST['emailTo'] && ↵
$_POST['emailFrom'] && ↵
$_POST['emailSubject'] && ↵
$_POST['emailMessage'])) ↵
{
    sendEmail($_POST['emailTo'], $_POST['emailFrom'], ↵
$_POST['emailSubject'], $_POST['emailMessage']);
    return true;
} else {
    return false;
}

function sendEmail($to, $from, $subject, $message)
{
    // Create a new eMail object
    $email = new eMail("Flex PHP Mail Form", $from);

    // Subject line
    $email->subject($subject);

    // To (address)
    $email->to($to);
```

```
// HTML body message
$email->html($message);

// send e-mail
$email->send();
}
?>
```

2. Create the MXML form:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical">
  <mx:Script>
    <![CDATA[
      [Bindable]
      // Characters allowed by a legal e-mail address
      private var allowedEmailCharacters:String =
"A-Za-z0-9!\#\$\%\&\ '\*\+\-\./\=?\^\_\`{\|\}\~\. ";
      private var emailForm:EmailDTO = new EmailDTO;

      [Bindable]
      class EmailDTO
      {
        public var emailTo:String;
        public var emailFrom:String;
        public var emailSubject:String;
        public var emailMessage:String;

        public function EmailForm()
        {
        }
      }
    ]]>
  </mx:Script>
  <mx:HTTPService id="serviceSendEmail" showBusyCursor="true"
resultFormat="text" concurrency="last" url="email.php"
requestTimeout="60"/>

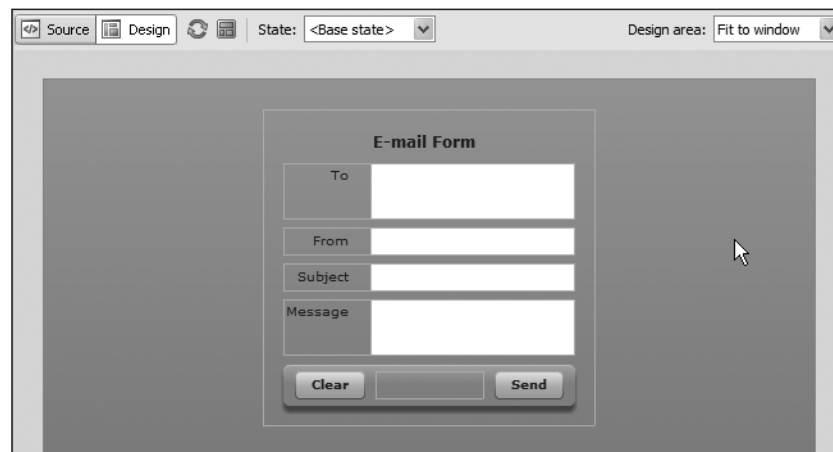
  <mx:Form id="frm_Email">
    <mx:FormHeading label="E-mail Form"/>
    <mx:FormItem label="To">
      <mx:TextArea id="emailTo" restrict=
"{allowedEmailCharacters}"
change="{emailForm.emailTo = this.emailFrom.text}"/>
    </mx:FormItem>
    <mx:FormItem label="From">
      <mx:TextInput maxChars="100" id="emailFrom" maxChars="319"
restrict="{allowedEmailCharacters}"
change="{emailForm.emailFrom = this.emailFrom.text}"/>
```

```

</mx:FormItem>
<mx:FormItem label="Subject">
  <mx:TextInput maxChars="100" id="emailSubject" ↵
change="{emailForm.emailSubject = this.emailSubject.text}"/>
</mx:FormItem>
<mx:FormItem label="Message">
  <mx:TextArea id="emailMessage" ↵
change="{emailForm.emailMessage = this.emailMessage.text}"/>
</mx:FormItem>
<mx:ApplicationControlBar width="100%">
  <mx:Button label="Clear"/>
  <mx:Spacer width="100%" height="100%"/>
  <mx:Button label="Send" ↵
click="{serviceSendEmail.send(emailForm);}"/>
</mx:ApplicationControlBar>
</mx:Form>
</mx:Application>

```

The form should look like Figure 6-1 in design view.



**Figure 6-1.** Flex form for the e-mail PHP script

You see the form is composed of mainly MXML with a dash of ActionScript. I'll explain the code briefly, but I don't want to give too much away since you'll be learning all about forms and validation in Chapter 8. We add an ActionScript variable, `allowedEmailCharacters`, which is a string filled with valid characters that we use as a whitelist in most of our form fields. This variable is bindable and is used as the value of the `restrict` attribute in many of the `<mx:TextInput/>` tags. The `restrict` attribute is very powerful, allowing only listed characters to be entered into a field while completely ignoring all other characters or keystrokes that a user may try to enter. Features like this are all covered in Chapter 8.

Each form field has a `change` attribute that assigns a value to the `EmailDTO` object as the user types or makes edits in the form field. Using a DTO simplifies integrating additional events and tasks a form

may be expected to perform. This separation helps you when you want to add validators and then eventually hand off the form data.

The form in this exercise is designed to pass the form data off to a PHP page on the server. Time to test our form.

### 3. Upload files and test.

If all the files are uploaded, you can fill out the Flex form, and you should get an e-mail shortly after clicking the Send button, which should fire the `send()` method of your `HTTPService`. The following e-mail output is what we used for an early alpha of the RMX's e-mails.

*Another industry professional like yourself felt you should see this RMX hosted video:  
<http://videos.richmediax.com/abc123>  
Please click or visit the link above to see the video that was recommended to you to watch.  
—RMX  
RichMediaX .com—turn-key community tools for Adobe & Open Source user groups worldwide.*

As you can see in the e-mail text, we use our permalink. You'll want to use your permalinks over and over, everywhere, like in e-mails and other sharing methods such as embed code, which you'll read about in the next section.

## Sharing and embedding video

As soon as video on the Web became popular, there came the need to share these videos beyond just a standard link. And, thanks to applications like Adobe's Flash Player, people caught on to the fact that you could embed media into web pages and blog posts, resulting in today's Internet video boom. Now, video services allow users to take a single video and place it anywhere in their own web page or site by copying and pasting code provided by the video sharing service.

### Embedding with JavaScript

JavaScript code to embed Flash video is usually very small. When using JavaScript for embed code, we pass around a `<script>` tag that points to an external JavaScript file that takes a few encapsulated arguments to know which video to return. For example:

```
<script type="text/javascript" src="http://www.embedmedia.com/↵  
embedcode.js.php?w=100&h=100&media=976"></script>
```

As you can see in the `src` attribute of the `<script>` tag, the file `embedcode.js.php` has query parameters attached to its location. Using a query string, like `?w=100&h=100&media=976`, allows us to pass any arguments or parameters we'd need to customize the embed code output. Taking a look at the code sample again, notice the width and height are `w=100` and `h=100` while `media=976` is some ID or content key to the media the link is requesting. In the back end, this would parse and return a JavaScript line,

document.write(preparedEmbedCode);, that contains the embed tags with parameters whose values are derived from the query string variables.

## Embedding with XHTML

When it comes to passing around plain HTML or XHTML code, sometimes a simple `<embed>` tag suffices, but it's up to you and external sites if code should contain both an `<object>` and an `<embed>` tag. Regardless of the distribution method, each requires width and height attributes to be set. Leaving SWF dimensions out can lead to unexpected results.

You also need to point to the target SWF file in your embed code. The `<embed>` tag uses its `src` attribute for the SWF's location; the `<object>` tag uses the `value` attribute. You see both the `src` and dimensions set in the following embed code:

```
<embed id="rmxEmbeddedPlayer_976" width="640" height="480" ↵  
title="RMX, centralizing cool videos" ↵  
src="http:// videos.richmediax.com/player.swf" ↵  
flashvars="media=976&embedded=true"/>
```

You'll notice there's a little more than just width, height, and `src`. There's a `title` attribute that provides users with an informative tooltip when they mouse over the embedded media. You add a unique `id` attribute by concatenating the media ID and giving it a value of `"rmxEmbeddedPlayer_976"`—named for its purpose and which media element it will render. This is to help you call each instance of the embedded video by this ID through external interfaces like JavaScript.

Looking at the rest of the embed code, you'll notice two more attributes with values; the `src`, which is the SWF location, and the `flashvars` attribute, which carries all the query parameters the application will need to run. While this works practically anywhere, some web sites may require external embedded media or services to only come from an approved source. Why? It's their business, it's their traffic, it's their house—so they want you to play nice, that's all. Too bad some don't play nice themselves.

For example, MySpace will add and change your embed codes by tweaking `<embed>` attributes and even adding some you didn't add: `allownetworking=internal`, `allowScriptAccess=never`, `enableJSURL=false`, `enableHREF=false`, and `saveEmbedTags=true`—all could cause your widget to be inoperable or just work incorrectly. You should log how each (target) site alters your embed code.

In addition, expected FlashVars could easily be targeted for removal within your embed code; this is something we have seen. One solution around this is concatenating FlashVars as query parameters in the `src` attribute like so:

```
<embed id="rmxEmbeddedPlayer_976" width="640" height="480" ↵  
title="RMX, centralizing cool videos" ↵  
src="http:// videos.richmediax.com/player.swf↵  
?media=976&embedded=true"/>
```

Technically, there isn't a major difference between an embed code with FlashVars and one without (using the query parameters workaround). The FlashVars attribute arrived in the Flash world as a cleaner standard for passing in external values into embedded Flash elements on pages. Before FlashVars, everyone used the plain query strings method, but appending URL-encoded variables to SWF addresses basically prevents proper browser caching of SWFs, which is one reason FlashVars are considered superior. Also, query strings have some character limits, so moving to a valued attribute

allows us to add as much data as we need, only resulting in a larger page size. We could also use HTML or PHP headers to prevent caching of the page and all elements in it; most may find this the cleanest solution.

Now, imagine you encounter a site that strips out all FlashVars and SWF query strings (removing everything that comes after the question mark in your `src` attribute URL). You'll need to hide the parameter values somehow. Try using the permalink `mod_rewrite` technique mentioned earlier. The following sample code is the third style of embed code, no FlashVars or query string values, using a `mod_rewrite` URL. This unique URL has both the server location and all the necessary arguments disguised as the SWF's location.

```
<embed id="rmxEmbeddedPlayer_976" width="640" height="480" ↵
title="RMX, centralizing cool videos" ↵
src="http://videos.richmediax.com/player/media/976/embedded"/>
```

As you can see, the embed code has the server URL, same as the other embed codes, but this time it's not pointing directly to a SWF. Instead you see a clean URL. Looking closer, you can see it says `player/media/976/embedded`, which Apache will internally rewrite or redirect to another location pointing to the real SWF like `player.swf?media=976&embedded=true`. Now that's a winner—this custom embed code can be embedded around the world reliably, today. I say “today” because at any time popular sites could create rules that remove your embed code if there's no file extension in the `src` attribute. If this is the case, to get around this, add a fake file name, like `video.swf`, at the end of the embed `src` URL, which technically doesn't exist and doesn't really affect the code much. Here's how the embed code would look changed to use the fake SWF in the `src` URL:

```
<embed id="rmxEmbeddedPlayer_976" width="640" height="480" ↵
title="RMX, centralizing cool videos" ↵
src="http://videos.richmediax.com/player/media/976/embeddedvideo.swf"/>
```

## Using RSS

The syndicated web is very large and will only get more widespread. XML feeds, such as RSS and the like, all have one focus—to check for and fetch the latest relevant content. The content can be news on a web site, the latest blog posts, photo feeds, podcasts, or video feeds. Folks choose their own software to consume XML feeds, allowing the software to do all the grunt work of pinging web sites for updated content. I use the term “XML feed” because that's really what RSS is, but RSS is not alone. Other popular syndication formats are RDF and ATOM. Both are more robust and advanced than RSS, but RSS is the most popular since it's the type of feed many people have grown to recognize.

You'll learn how to generate a valid RSS feed, and extend the feed to use custom tags and popular media tags. First, I'll give you an overview of the basics of consuming a feed in the most-used browsers today. Syndicated content has a pseudo-standard icon that most browsers and applications have adopted (see Figure 6-2).

The Mozilla Foundation was the first to popularize the icon in their web browsers. The icon then became a standard once it was offered as an open design freely to promote content syndication features in general, not just RSS. If you wish to adopt the icon, and I recommend you do, check out a web site called Feed Icons; this site not only offers information, but also has the icon in both vector and bitmap formats available free for download at [www.feedicons.com](http://www.feedicons.com) (see Figure 6-3).



**Figure 6-2.** The most popular syndication or feed icon to date

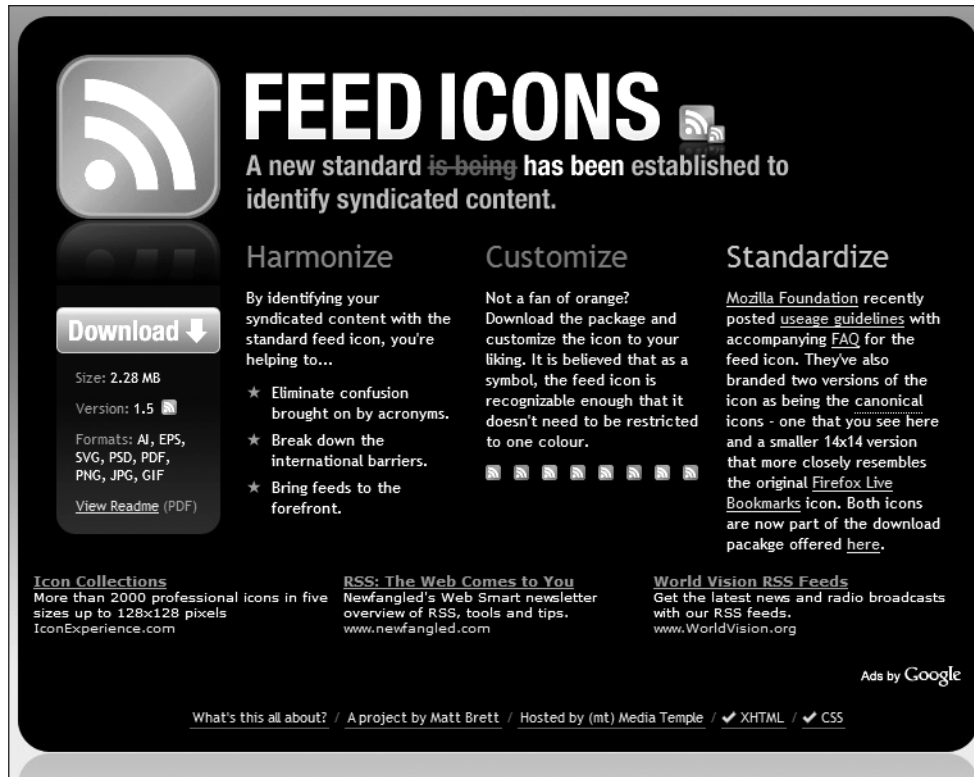


Figure 6-3. Feed Icons web site

Applications, like web browsers, typically use the feed icon to show which sites are notifying them there are feeds available for the URL you are visiting. Mozilla Firefox (see Figure 6-4) and Microsoft Internet Explorer (see Figure 6-5) use the standard feed icon. Apple Safari doesn't use the standard icon but their design isn't hard to miss (see Figure 6-4).



Figure 6-4. Mozilla Firefox and Apple Safari browsers and their XML/RSS feed icon

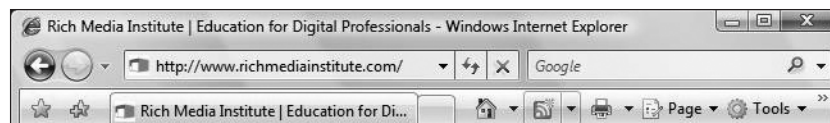


Figure 6-5. Microsoft Internet Explorer 7 on Microsoft Windows Vista uses the same icon to notify the page has an XML/RSS feed discovered.

Now let's look at a basic RSS feed, from the RMX, for the LA Flash user group:

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>LA Flash // Video Feed // RMX - Rich Media Exchange</title>
    <link>http://laflash.org/link</link>
    <description>LA Flash, an authorized Adobe User Group</description>
    <language>en-us</language>
    <pubDate>Thu, 14 Jun 2007 07:00:00 GMT</pubDate>
    <lastBuildDate>Fri, 15 Jun 2007 07:00:00 GMT</lastBuildDate>
    <generator>RMX - RichMediaX.com</generator>
    <managingEditor>editor@laflash.richmediax.com</managingEditor>
    <webMaster>webmaster@richmediax.com</webMaster>
    <item>
      <title>Adobe AIR (Adobe Integrated Runtime)</title>
      <link>http://laflash.org/news/2007/air.htm</link>
      <description>Chris Charlton shows off some Adobe AIR.</description>
      <pubDate>Wed, 20 June 2007 11:39:21 GMT</pubDate>
      <guid>http://laflash.richmediax.com/news/2007/air.htm</guid>
    </item>
    <item>
      <title>Flex for Flash Designers</title>
      <link>http://laflash.org/news/2007/flex_for_designers.htm</link>
      <description>Flash production tips for Flex Developers.</description>
      <pubDate>Wed, 16 May 2007 10:30:00 GMT</pubDate>
      <guid>http://laflash.org/news/2007/flex_for_flash_designers.htm</guid>
    </item>
    <item>
      <title>AS3 for Beginners Nanocamp preview</title>
      <link>http://laflash.org/news/2007/as3_begin_nanocamp_preview.htm</link>
      <description>R Blank previews a Nanocamp, AS3 for Beginners.</description>
      <pubDate>Wed, 18 Apr 2007 10:39:21 GMT</pubDate>
      <guid>http://laflash.org/news/2007/as3_begin_nanocamp_preview.htm</guid>
    </item>
  </channel>
</rss>
```

Some initial dissection of the feed shows the RSS standard root node, `<rss version="2.0">`, with a `<channel>` child node. The `<channel>` node carries children nodes that describe the feed's publisher

and its syndicated content. Feed content is listed in the multiple `<item>` nodes that follow the channel's metadata. The nodes shown in the example feed are all the nodes you should start with. Additional nodes open feeds to software features and search-friendly syndication. Before covering `<item>` nodes, I'll run through the `<channel>` meta info for you:

- `<title>`: The title to display for the feed. Feed readers usually render this title in the list of feeds and sometimes near the top of a rendered feed.
- `<link>`: The corresponding URL for the channel, not the feed itself. Many feed readers use this URL to take you back to the web site of the feed publisher.
- `<description>`: A body of text providing a description of the feed itself or where it originated from.
- `<language>`: The language the feed is presented in. You cannot assume there's a default language, so it's best to make sure you include this node.
- `<pubDate>`: The publishing date of the feed. It's nice if you can use the original publishing date the feed first launched, but nobody will hate you if this isn't exact.
- `<lastBuildDate>`: The last build date is used by software to check if there's a published update to your feed since it was last checked.
- `<generator>`: The name of the service or software that generated the feed.

There are over a dozen nodes for a channel, but most are optional. There are nodes that pertain to tags or keywords, copyright, editor, even an image for the channel. Apple iTunes uses the channel image as the logo of the feed. If you care to review RSS's other channel nodes, you may do so at <http://cyber.law.harvard.edu/rss/rss.html>. Once you fill these nodes in with channel info, people and software will know who's responsible for the feed and apply your preferences. If you plan to publish content that may be explicit or mature, like foul language, violence, or nudity, you will want your feeds to adopt the rating system for the channel `<rating>` node. Now that you have channel info, it's time for the feed's content.

Remember, a feed's content is listed in the multiple `<item>` nodes that appear under the channel info. There's no limit to how many items can be listed in a feed. Honestly, the feed can be as long as you care to have it; the number of items depends on your feed's focus. Short feeds list around 10 to 20 items, while longer feeds can list hundreds of items. Only if you plan to use a feed as an archive would you use a lot of nodes, but RSS isn't exciting when listing stagnant content. As ActionScript developers, we really don't care about how large a feed is, since our E4X features are processed quickly on the client side.

You'll want to use RSS for new or updated content. Sites offer feeds for the latest content on their network or content in a category or tag. RMX uses RSS to list the latest content of an entire user group or any tag a user chooses. Items in a feed offer content meta info, like title and description, as well as tags to publish dates, keywords, and links to the content.

Take a look at this `<item>` node:

```
<item>
<title>Adobe AIR (Adobe Integrated Runtime)</title>
<link>http://laflash.richmediax.com/news/2007/air.htm</link>
<description>Chris Charlton shows
off some Adobe AIR.</description>
```

```

<pubDate>Wed, 20 June 2007 11:39:21 GMT</pubDate>
<guid>http://laflash.richmediax.com/news/2007/air.htm</guid>
</item>

```

Similarly, in the <channel> node you see a <title> and <description> subelement for each <item>. The <link> subelement provides a URI to that piece of content. You will notice the <guid> node has the same value as the <link> node; this is for simplicity. The <guid> subelement requires a unique value for each unique piece of content, so instead of preparing complex code to handle that uniqueness, you reuse the content's link since those would be inherently unique. When it comes to filling out the <guid> and <link> tags, for each <item> node, you use your cool permalinks. The permalink values give you two benefits in your feed. One benefit is the links are short and will generally help keep RSS feeds slightly smaller, since long links would add more file size. Another benefit is that you help perpetuate the simple formatted URLs you hand out to users for sharing. You finish filling the <item> nodes with a properly formatted publish date, the <pubDate>, per item.

## Attaching files to RSS feeds

Content doesn't have to be just text; you can use HTML or more XML to describe the content in a <description> subelement and attach files. Using an optional node for feed enclosures, these can be considered attachments, providing a link to a file or files. Enclosure-aware software sniffs for <enclosure> tags and downloads them as your preferences see fit. Feed enclosures are what gave podcasting its life, by linking to audio files people recorded. Take a look at the RSS feed example again, this time with feed enclosures:

```

<item>
<title>Adobe AIR (Adobe Integrated Runtime)</title>
<link>http://laflash.org/news/2007/air.htm</link>
<description>Chris Charlton shows
  off some Adobe AIR.</description>
<pubDate>Wed, 20 June 2007 11:39:21 GMT</pubDate>
<guid>http://laflash.org/news/2007/air.htm</guid>
<enclosure url="http://richmediax.com/video/laflash/adobe_air.flv"
length="6428422" type="video/x-flv" />
</item>
<item>
<title>Flex for Flash Designers</title>
<link>http://laflash.org/news/2007/
  flex_for_flash_designers.htm</link>
<description>Flash production tips for Flex
  Developers.</description>
<pubDate>Wed, 16 May 2007 10:30:00 GMT</pubDate>
<guid>http://laflash.org/news/2007/
  flex_for_flash_designers.htm</guid>
<enclosure url="http://richmediax.com/audio/laflash/flexforflash.mp3"
length="12216320" type="audio/mpeg" />
</item>
<item>
<title>AS3 for Beginners Nanocamp preview</title>
<link>http://laflash.org/news/2007/
  as3_begin_nanocamp_preview.htm</link>

```

```

<description>R Blank previews his latest Nanocamp,
  AS3 for Beginners.</description>
<pubDate>Wed, 18 Apr 2007 10:39:21 GMT</pubDate>
<guid>http://laflash.org/news/2007/
  as3_begin_nanocamp_preview.htm</guid>
<enclosure url="http://richmediax.com/video/
  laflashes/as3_nanocamp.flv"
length="101010"
type="video/x-flv" />
</item>

```

You see the new `<enclosure>` tags have three attributes—`url`, `length`, and `type`. URLs point to the location of the enclosure, so they should be absolute paths, since people will be reading your feed on their computer, maybe even offline. `length` says how big the enclosure is in bytes, not time, and `type` gives the file or MIME type. FLV is pretty new for use in enclosures, so you'll want to know that you can use either of the MIME types in the preceding RSS snippet when referring to FLVs. You can essentially link to any kind of file, as there's no verification of any type; feed enclosures are just links to files, and the software is responsible for handling them. You can have multiple `<enclosure>` tags per item.

## XML namespaces and extensions

Since RSS is XML, shouldn't you be able to make up your own tags and add them to the feed at will? Yes and no. XML is easy and RSS is XML, so yes, you're able to add tags but not without following a couple of important but easy rules. When adding custom tags to an existing XML structure, you need to make sure your tags don't conflict with tags that are already defined in the structure. Imagine you want to add a `<name>` tag, but is it for the name of the web site, the user, the publisher, or even the author? Even worse, what if someone else adds tags that are the same as yours, resulting in conflicts and validation errors? The right way to make sure your tags stay your own is by giving them what's known as an **XML namespace**. You declare your custom namespace in an `xmlns` attribute of the root node in your XML document. You're allowed to generate as many custom tags and attributes as you need with your namespace used as a prefix followed by a desired suffix, separated by a colon, like in these examples:

```

<?xml version="1.0"?>
<rootNode xmlns:rmx="http://richmediax.com"
xmlns:myNamespace="http://someURL.com"
xmlns:namespacePrefix="http://someOtherURL.com">
...
<namespacePrefix:suffix>...</namespacePrefix:suffix>
<myNamespace:myTag>...</myNamespace:myTag>
<rmx:name rmx:type="usergroup">...</rmx:name>
...
</rootNode>

```

This is the standard way for adding nodes, or groups of nodes, to existing XML structures. In a sense, you're extending the structure, that's why custom namespaces are sometimes known as XML extensions. Hundreds of XML extensions exist on the Web; anything to do with documents or media, it's out there. I'll show you how to extend this feed with a popular XML extension, Media RSS (MRSS), developed by Yahoo! and used by Adobe in their Adobe Media Player as well as other feed readers.

Now how would we get at these custom nodes through ActionScript? With E4X, of course, but we need to register the custom namespace so we can hit those nodes right. Here's some sample code:

```
var rmx:NameSpace = rssObject.namespace('rmx');
var rmxUserGroupNameFromFeed:String = rssObject.rmx::name.toString();
trace(rmxUserGroupNameFromFeed); ➤
// Outputs "LA AIR"
```

As you see, we created an ActionScript NameSpace object called `rmx`, reading from our loaded RSS feed (`rssObject`). We use the variable name `rmx` to keep it the same as our feed and keep it simple for our code too. Next, we access the nodes of the custom namespace similarly to how we access regular nodes.

*Adobe released some free ActionScript 3 classes that parse XML feeds like RSS/ATOM/RDF, which were originally published on Adobe Labs (<http://labs.adobe.com>) and are now maintained over at Google Code (<http://code.google.com>). This class package is now part of a larger AS3 set called FlexLib. We recommend checking out the custom classes and components available at Google Code and at Adobe Exchange ([www.adobe.com/exchange](http://www.adobe.com/exchange)).*

For RMX, we knew we'd be publishing RSS feeds but needed some extra information embedded within the RSS feed `<item>` nodes, particularly geolocation and user group origin. Making up a custom namespace can be easy, since the URL doesn't have to exist to be valid, just as long as it's unique. Following is a sample of an RMX User Group video feed using our custom XML namespace and extension, `rmx`.

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:rmx="http://richmediax.com">
  <channel>
    <title>LA Flash // Video Feed // RMX - ➤
      Rich Media Exchange</title>
    <link>http://laflash.richmediax.com</link>
    <description>LA Flash, an authorized Adobe ➤
      User Group</description>
    <language>en-us</language>
    <pubDate>Thu, 14 June 2007 07:00:00 GMT</pubDate>
    <lastBuildDate>Fri, 15 Jun 2007 09:00:00 GMT</lastBuildDate>
    <generator>RMX - RichMediaX.com</generator>
    <managingEditor>editor@laflash.richmediax.com</managingEditor>
    <webMaster>webmaster@richmediax.com</webMaster>
    <rmx:usergroup name="LA Flash">
    <rmx:topics>
      <rmx:topic>Flash</rmx:topic>
      <rmx:topic>ActionScript</rmx:topic>
    </rmx:topics>
    <rmx:location>
    <rmx:city>Los Angeles</rmx:city>
    <rmx:state>CA</rmx:state>
    <rmx:country>United States</rmx:country>
```

```

</rmx:location>
</rmx:usergroup>
  <item>
    <title>Adobe AIR (Adobe Integrated Runtime)</title>
    <link>http://laflash.richmediax.com/news/2007/air.htm</link>
    <description>Chris Charlton shows ▶
      off some Adobe AIR.</description>
    <rmx:tags>Flash, AIR, ActionScript, ActionScript 3, AS3</rmx:tags>
    <pubDate>Wed, 20 June 2007 11:39:21 GMT</pubDate>
    <guid>http://laflash.org/news/2007/air.htm</guid>
    <enclosure url="http://richmediax.com/video/laflash/
      adobe_air.flv" ▶
      length="6428422" type="video/x-flv" />
    </item>
    <item>
      <title>Flex for Flash Designers</title>
      <link>http://laflash.richmediax.com/news/2007/▶
        flex_for_flash.htm</link>
      <description>Flash production tips for Flex ▶
        Developers.</description>
      <rmx:tags>Flash, Flex, MXML, AS3, ActionScript, ▶
        Design</rmx:tags>
      <pubDate>Wed, 16 May 2007 10:30:00 GMT</pubDate>
      <guid>http://laflash.org /news/2007/▶
        flex_for_flash.htm</guid>
      <enclosure url="http://richmediax.com/audio/laflash/
        flexforflash.mp3" ▶
        length="12216320" type="audio/mpeg" />
      </item>
      <item>
        <title>AS3 for Beginners Nanocamp preview</title>
        <link>http://laflash.richmediax.com/news/2007/▶
          as3_begin_nanocamp.htm</link>
        <description>R Blank previews AS3 for ▶
          Beginners.</description>
        <rmx:tags>ActionScript 3, AS3, Flash CS3, ▶
          Flash 9</rmx:tags>
        <pubDate>Wed, 18 Apr 2007 10:39:21 GMT</pubDate>
        <guid>http://laflash.org/news/2007/▶
          as3_begin_nanocamp.htm</guid>
        <enclosure url="http://richmediax.com/video/laflashes/▶
          as3_nanocamp.flv" ▶
          length="101010" type="video/flv" />
        </item>
      </channel>
    </rss>

```

The bold lines in the sample feed are custom tags that use our custom XML extension, or namespace. For example, `<rmx:tags>` is the custom element we use for content tags, which are totally valid.

## MRSS and syndicating to the Adobe Media Player

Media RSS is a documented proposal by Yahoo! to help extend today's RSS to provide better publishing information embedded in RSS for media such as video, audio, or any file type for that matter. When using RSS's <enclosure> tag, you can assign one or more files to an <item> element, but it may only be one video format the feed can offer this way. For common formats, like JPEG, PNG, GIF, and MP3, this option seems perfectly fine for that scenario. What if you need to promote one feed that broadcasted all the common formats, and when you have video you want it to cater the format to the user's preference or whatever the supported software would automatically choose? Software would grab an FLV video, or iPod video, or PSP video, or even a QuickTime HD version of the video—I think you get the picture.

Yahoo! saw fit to include additional meta information like bit-rate choices, playback restrictions, content maturity ratings, and Creative Commons licensing. Here's an example from Yahoo! of a movie review feed, with a trailer, using a Creative Commons license:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss"
xmlns:creativecommons="http://backend.userland.com/↵
creativecommonsRssModule">
<channel>
<title>My Movie Review Site</title>
<link>http://www.foo.com</link>
<description>I review movies.</description>
<item>
<title>Movie Title: Is this a good movie?</title>
<link>http://www.foo.com/item1.htm</link>
<media:content url="http://www.foo.com/trailer.mov"
fileSize="12216320" type="video/quicktime" expression="sample"/>
<creativecommons:license>
http://www.creativecommons.org/licenses/by-nc/1.0
</creativecommons:license>
<media:rating>nonadult</media:rating>
</item>
</channel>
</rss>
```

Adobe thought MRSS was a good extension to adopt in their new Adobe Media Player application (see Figure 6-6). Adobe Media Player supports RSS subscriptions with XML extensions, like MRSS, for content downloading, playback restrictions, custom branding options, and interactive advertisements over your content. MRSS information is available at <http://search.yahoo.com/mrss>.

Adobe Media Player, formerly code-named Philo, is a new application that has custom branding and advertising options for content publishers and supports subscribing to channels via RSS, similar to iTunes and other recent media players. To veteran podcast fans, this is no big deal, but Adobe's offering fills a void for users, since before they couldn't enjoy watching FLV and SWF files on their desktop easily. Adobe can do this now in a much more engaging way since they released AIR, allowing web developers to make desktop applications with Ajax and the Flash Platform.

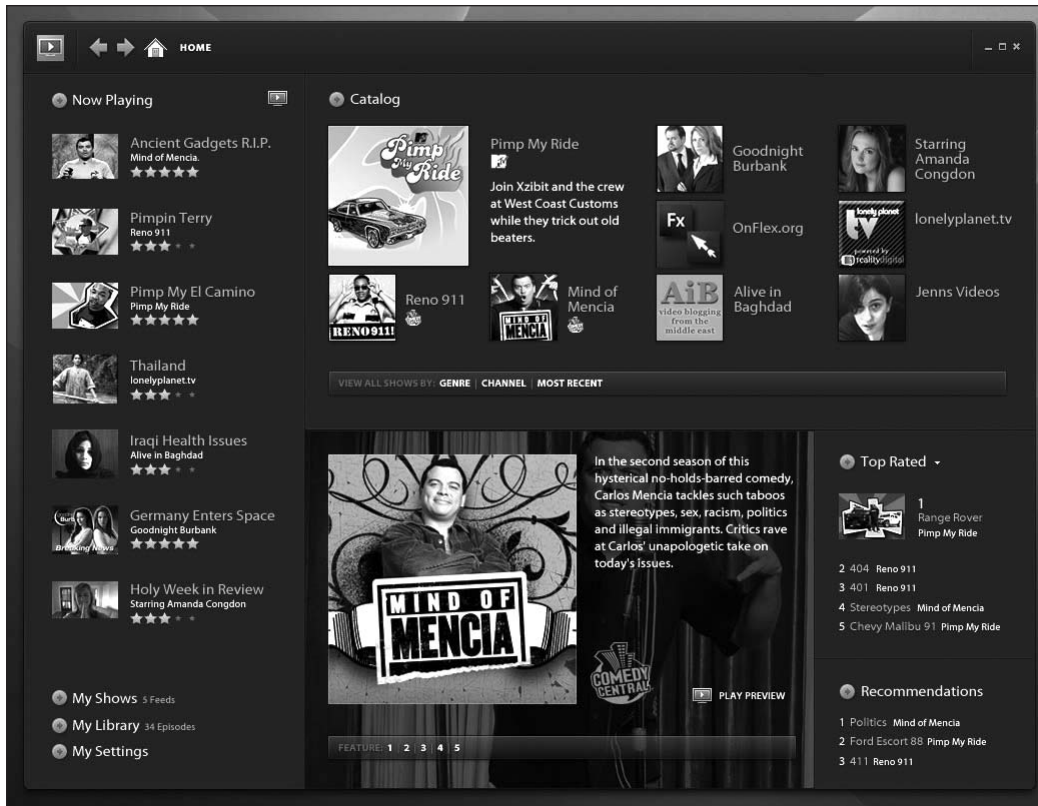


Figure 6-6. Adobe Media Player (alpha)

## Summary

This chapter analyzed MySQL as a key data source (relational databases) and how you can render that data in basic and extended XML to generate core Web 2.0 functionality like podcasts. By providing members with ways to take or share content from the RMX network through permalinks, send-to-friend e-mails, and embed code, we provide the basic functionality users expect in contemporary social media networks.

In the next chapter, Hasan will explain how we built some of the key navigational items for the RMX.