CHAPTER 18

Developing Mashup AIR Applications

The term *mashup* has been used in the music industry for quite some time to define the result of producing a new sound by mixing two or more existing pieces together, so it's easy to guess that in the world of web applications the term means the ability of an application to combine data from more than one source.

Web application mashups were born at the same time as the new approach of Web 2.0, which, among its many objectives, was aimed at reusing different data sources to create hybrid applications. So, mashups owe their success to the growth of user-generated content on the Web. For example, with the increase of blogs on the Internet, bloggers wanted to offer services and multimedia elements in addition to classical web content, which is how services such as YouTube, Flickr, and Google Maps started to be integrated in millions of posts, adding value to the simple text. But all this wouldn't have been possible if it hadn't been for the increase of web services and XML data sources that could be used and invoked by the applications. Another great catalyst of mashup applications was the success of Ajax and technologies such as Flex, Ruby on Rails, and now Adobe AIR.

Useful mashup applications are born on the Web every day. Here are a few examples that you can try straightaway to get a better idea of what it means to combine different data sources in one application:

Kayak (http://www.kayak.com)
: A mashup that consumes Expedia, Traveloprice, and other travel web services to help you choose your flight or hotel at your convenience. In the trip ideas section (*http://www.kayak.com/h/buzz*), it uses Google Maps.

Flash Earth (http://www.flashearth.com)
: A zoomable mashup of satellite applications created with Flash. It consumes web services from many sources: NASA, Google Maps, Virtual Earth, Yahoo Maps, and OpenLayers—to name just a few.

Netvibes (http://www.netvibes.com)
: Netvibes is a personalized start page similar to Pageflakes, My Yahoo, iGoogle, and Microsoft Live. It is written in Ajax and organized into tabs, with each tab

containing user-defined modules. It integrates and consumes an RSS/Atom feed reader, local weather forecasts, a calendar supporting iCal, bookmarks, notes, to-do lists, and multiple searches. It also has support for POP3, IMAP4 email, and the webmail providers Gmail, Yahoo Mail, Hotmail, and AOL Mail. In addition, it supports Box.net web storage, del.icio.us, Meebo, Flickr photos, podcasts (via a built-in audio player), and more.

You can find a more complete and updated list of available mashups at *http://www.programmableweb.com/mashups*.

Mashup applications are based on the possibility of consuming remote data sources, and to create one, you need a good understanding of the APIs available. (You can find an updated list of APIs at *http://www.programmableweb.com/apis/directory*.) AIR offers even greater possibilities for creating mashup applications and widgets. With AIR, you can go beyond all the sandbox security of the browser and add advanced features to the application to interact with the file system or local storage with SQLite. This chapter demonstrates how to *integrate* the Flickr, Yahoo Maps, and Twitter web services to create desktop mashup applications with AIR.

18.1 Consuming Flickr Web Services

Problem

You want to consume Flickr (*http://www.flickr.com*) web services.

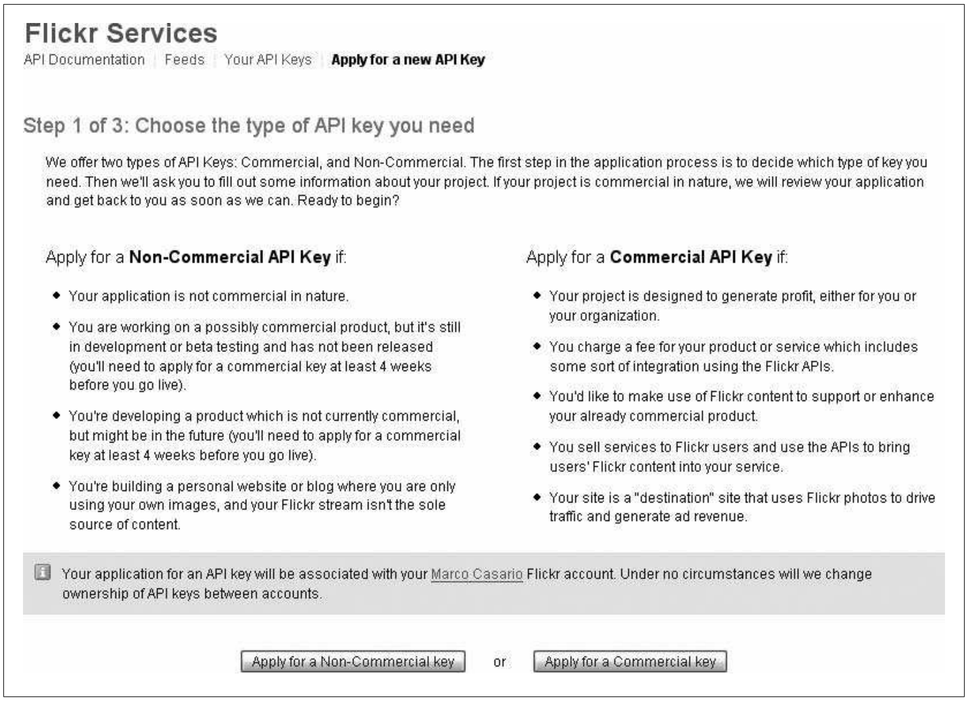
Solution

Useful to amateur and professional photographers alike, Flickr is a portal that provides photographic material. You can use it as a simple private archive of password-protected photographs or as an online portfolio that can be freely accessed by potential clients or whoever else is interested. Flickr's services, however, have limits. For example, because Flickr is a web application, it requires you use a web browser to access the service.

You can avoid this limitation with AIR.

Discussion

AIR applications allow you to access many functions that can't normally be accessed by traditional web developers, such as reading and writing files on the user's computer and creating multiwindow applications. By joining the simplicity of Flickr's services with the power and flexibility of the AIR software development kit (SDK), you can create very interesting mashup applications. The only limits to the integrations are your imagination and needs.


Flickr Services

API Documentation | Feeds | Your API Keys | **Apply for a new API Key**

Step 1 of 3: Choose the type of API key you need

We offer two types of API Keys: Commercial, and Non-Commercial. The first step in the application process is to decide which type of key you need. Then we'll ask you to fill out some information about your project. If your project is commercial in nature, we will review your application and get back to you as soon as we can. Ready to begin?

Apply for a **Non-Commercial API Key** if:

- Your application is not commercial in nature.
- You are working on a possibly commercial product, but it's still in development or beta testing and has not been released (you'll need to apply for a commercial key at least 4 weeks before you go live).
- You're developing a product which is not currently commercial, but might be in the future (you'll need to apply for a commercial key at least 4 weeks before you go live).
- You're building a personal website or blog where you are only using your own images, and your Flickr stream isn't the sole source of content.

Apply for a **Commercial API Key** if:

- Your project is designed to generate profit, either for you or your organization.
- You charge a fee for your product or service which includes some sort of integration using the Flickr APIs.
- You'd like to make use of Flickr content to support or enhance your already commercial product.
- You sell services to Flickr users and use the APIs to bring users' Flickr content into your service.
- Your site is a "destination" site that uses Flickr photos to drive traffic and generate ad revenue.

Your application for an API key will be associated with your Marco Casario Flickr account. Under no circumstances will we change ownership of API keys between accounts.

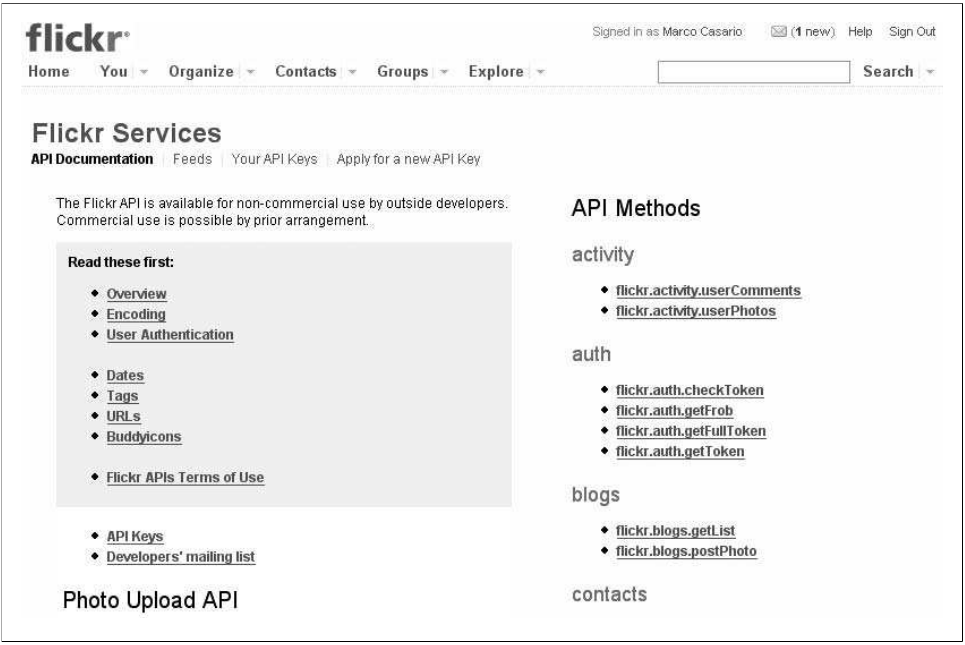
Figure 18-1. Flickr wizard to obtain your free application key

To access Flickr services, you first must register and request a free application key (*http://www.flickr.com/services/api/keys/apply/*). The key consists of two codes: an access key and a shared secret key; the combined use of the two keys allows you to access all the services provided by the portal. To get the application key, you'll be asked to fill out a simple online form (Figure 18-1), specifying the type of application you are creating, the technologies involved, and whether the final product will be commercial.

Once you obtain the application key to use the service, you can begin using Flickr. The documentation (Figure 18-2), which is very easy to understand, is available online at *http://www.flickr.com/services/api*.

The online documentation lists which functions are available to developers and how to use them. You will also find a list of the libraries to access existing Flickr services for the main web and desktop programming languages, such as PHP, .NET, Java, Delphi, Perl, and obviously ActionScript. The libraries that have been created to access the Flickr web service enable you to interact with the functions of the portal without having to worry about communication details between the web services and your development environment.

The recommended library on the Flickr portal, and the one used in this example, is as3flickrlib, developed by Mike Chambers and available as a Google Code project (*http://code.google.com/p/as3flickrlib/downloads/list*). In addition, as3flickrlib requires



The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

Read these first:

- Overview
- Encoding
- User Authentication
- Dates
- Tags
- URLs
- Buddyicons
- Flickr APIs Terms of Use
- API Keys
- Developers' mailing list

Photo Upload API

API Methods

activity

- flickr.activity.userComments
- flickr.activity.userPhotos

auth

- flickr.auth.checkToken
- flickr.auth.getFrob
- flickr.auth.getFullToken
- flickr.auth.getToken

blogs

- flickr.blogs.getList
- flickr.blogs.postPhoto

contacts

Figure 18-2. Flickr API online documentation

the as3corelib library in order to manage HTTP queries to the web services and to parse the returned XML data. You can download as3corelib from *http://code.google.com/p/as3corelib/downloads/list*.

If the libraries don't offer the specific functions that your application needs, you can either create a new library to communicate with Flickr services or extend one of the existing libraries.

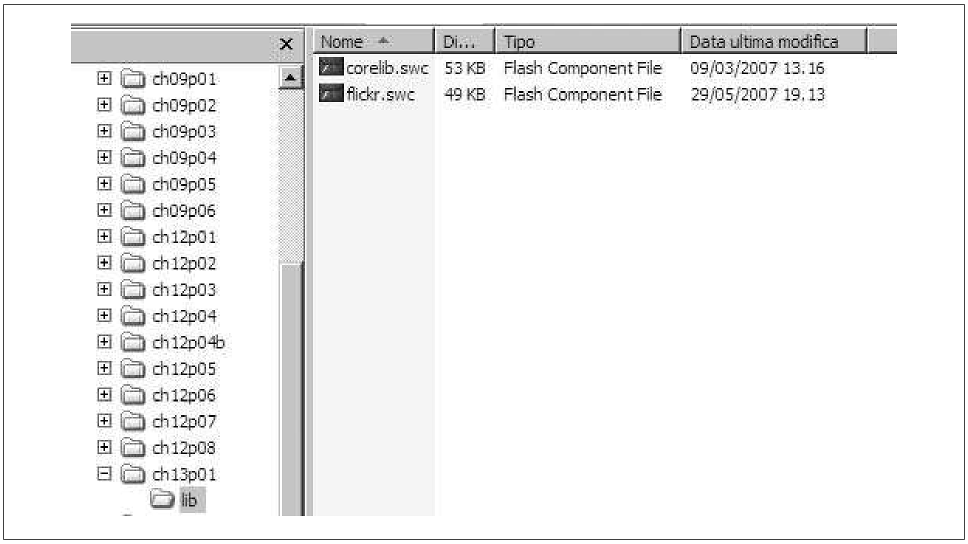
The downloads are compressed archives that contain ActionScript 3 sources, documents in HTML format regarding the available functions, and compressed libraries in SWC format (Figure 18-3). For AIR applications, you need the two libraries' SWC files. They should automatically decompress to the lib folder in your project folder, which contains the AIR application project.

ActionScript/Flash CS4

Once you obtain the necessary data from Flickr to consume its web services, you can begin creating the ActionScript class to control the mashup of the remote data.

From ASDoc in as3flickrlib (*http://as3flickrlib.googlecode.com/svn/trunk/docs/index.html*), take a look at the classes you'll be working with:

- `FlickrService`: Abstracts the Flickr API found at *http://www.flickr.com/services/api*
- `FlickrError`: Common errors that can happen during a call to a Flickr method
- `FlickrResultEvent`: Event class that contains information about events broadcast in response to data events from the Flickr API
- `PagedPhotoList`: A `ValueObject` for the Flickr API
- `Photo`: A `ValueObject` for the Flickr API
- `PhotoSize`: A `ValueObject` for the Flickr API
- `User`: A `ValueObject` for the Flickr API



Nome	Di...	Tipo	Data ultima modifica
corelib.swc	53 KB	Flash Component File	09/03/2007 13.16
flickr.swc	49 KB	Flash Component File	29/05/2007 19.13

Figure 18-3. ActionScript 3 required libraries

Here is the complete code of the `FlickrWS.as` class to launch a search and display the response data using Flickr APIs:

```
package com.oreilly.aircookbook.bonus
{
    import com.adobe.webapis.flickr.FlickrService;
    import com.adobe.webapis.flickr.PagedPhotoList;
    import com.adobe.webapis.flickr.Photo;
    import com.adobe.webapis.flickr.PhotoSize;
    import com.adobe.webapis.flickr.User;
    import com.adobe.webapis.flickr.events.FlickrResultEvent;

    import flash.display.Bitmap;
    import flash.display.Loader;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.net.URLRequest;
    import flash.system.Security;
    import mx.collections.ArrayCollection;

    [Bindable]
    public class FlickrWS
    {
        private const CROSSDOMAIN_ADDRESS:String = "http://api.flickr.com/crossdomain.xml";
        private const SECRET_KEY:String = "YOURSECRETKEY";
        private const SHARED_SHARED_KEY:String = "YOURSHAREDKEY";

        private const THUMBHEIGHT:Number = 175;
        private const THUMBWIDTH:Number = 155;

        public var fs:FlickrService;

        private var imgList:PagedPhotoList;

        private var userVO:User;

        private var activePhotoVO:Photo;

        private var _currImage:Number = 0;

        public function get currImage():Number
        {
            return _currImage;
        }

        public function set currImage(_currImage:Number):void
        {
            this._currImage = _currImage;
        }


        public function get thumbsDP():ArrayCollection
        {
            return _thumbsDP;
        }

        public function FlickrWS()
        {
            Security.loadPolicyFile( CROSSDOMAIN_ ADDRESS );
            fs = new FlickrService(SHARED_KEY);
            fs.secret = SECRET_KEY;

            fs.addEventListener( FlickrResultEvent.PHOTOS_GET_SIZES, onPhotoSize );
            fs.addEventListener( FlickrResultEvent.PEOPLE_GET_INFO, onUserInfo );
            fs.addEventListener( FlickrResultEvent.PHOTOS_SEARCH , onPhotoList );

        }

        private function onPhotoList( evt:FlickrResultEvent ):void
        {

            imgList = evt.data.photos as PagedPhotoList;
            if (imgList && imgList.total > 0)
            {
                loadNextImage();

            } else
            {
                trace("No results");
            }
        }

        private function onUserInfo( evt:FlickrResultEvent ):void
        {
            userVO = evt.data.user as User;

            checkReadyStatus();
        }

        private function onPhotoSize( evt:FlickrResultEvent ):void
        {
            if ( evt.success )
            {
                var availableSizeList:Array = evt.data.photoSizes;

                var availableSize:PhotoSize;

            for(var i:int = 0; i < availableSizeList.length; i++)
            {
                availableSize = availableSizeList[i];

            if ( availableSize.width > THUMBWIDTH ||
                 availableSize.height > THUMBHEIGHT )
            {
                break;
            }
        }

        if( availableSize.source != null || availableSize.source != "" )
        {

        if( imgLoader != null )
        {
            imgLoader.contentLoaderInfo.removeEventListener( Event.COMPLETE, onImageReady );
            imgLoader.contentLoaderInfo.removeEventListener( IOErrorEvent.IO_ERROR,
                                                             onImageError );
        }

        imgLoader = new Loader();

        imgLoader.contentLoaderInfo.addEventListener( Event.COMPLETE, onImageReady );
        imgLoader.contentLoaderInfo.addEventListener( IOErrorEvent.IO_ERROR, onImageError );

        imgLoader.load( new URLRequest( availableSize.source ) );

        } else
```

```
{
    loadNextImage();
}

} else
{
    loadNextImage();
}
}


private function checkReadyStatus():void
{
    if ( isImgReady )
    {
        var photoDataRow:Object = new Object();

        photoDataRow[ 'label' ] = imageTitle + " [" + imageAuthor + "]";

        photoDataRow[ 'source' ] = thumbnailImage;

        photoDataRow[ 'scaleContent' ] = true;

        photoDataRow[ 'photoVO' ] = activePhotoVO;

        _thumbsDP.addItem( photoDataRow );

        loadNextImage();

    } else
    {
        isImgReady = true;
    }
}


public function loadNextImage():void
{
    if (currImage >= imgList.photos.length)
    {
        return;
    }

        activePhotoVO = imgList.photos[ currImage ];

        isImgReady = false;

    if( activePhotoVO != null )
    {

    if( activePhotoVO.id != null )
    {

    fs.photos.getSizes( activePhotoVO.id );
```

```
        if( activePhotoVO.ownerId != null )
        {

            fs.people.getInfo( activePhotoVO.ownerId );

        }else
        {

            isImgReady = true;
        }
        } else
        {
            currImage++;

            loadNextImage();

            return;
        }
        } else
        {

        }

            currImage++;
        }


    private function onImageReady( evt:Event ):void
    {
        try
        {
            var imageAlias:Bitmap = Bitmap( imgLoader.content );

            checkReadyStatus();

        } catch (e:Error)
        {
         trace("Returned image was not a proper bitmap: " + imgLoader.loaderInfo.url);


        loadNextImage();
    }
    }


    private function onImageError( evt:IOErrorEvent ):void
    {
        trace( "Error loading image: " + imgLoader.loaderInfo.url );

        loadNextImage();
    }



    public function get thumbnailImage():Bitmap
```

```
            {
             return imgLoader.content as Bitmap;
            }


            public function get imageTitle():String
            {
             return activePhotoVO.title;
            }


            public function get imageAuthor():String
            {
             return userVO.fullname;
            }
            }
    }
```

The ActionScript class uses the Flickr library and imports all the classes it needs with a series of imports that point to the `com.adobe.webapis.flickr` package. It defines private constants that contain your API keys and the URL of the cross-domain file:

```
private const CROSSDOMAIN_ADDRESS:String = "http://api.flickr.com/crossdomain.xml";
private const SECRET_KEY:String = "YOURSECRETKEY";
private const SHARED_KEY:String = "YOURSHAREDKEY";
```

The class constructor creates event listeners to control the response data of a search, which includes the information returned on the images and the users:

```
fs.addEventListener( FlickrResultEvent.PHOTOS_GET_SIZES, onPhotoSize );
fs.addEventListener( FlickrResultEvent.PEOPLE_GET_INFO, onUserInfo );
fs.addEventListener( FlickrResultEvent.PHOTOS_SEARCH , onPhotoList );
```

The data provider is populated in the `onPhotoSize` event listener, which is triggered by the `Result` event of the Flickr call, and then the data provider is passed into the `tileList` control to display the images:

```
var photoDataRow:Object = new Object();

photoDataRow[ 'label' ] = imageTitle + " [" + imageAuthor + "]";

photoDataRow[ 'source' ] = thumbnailImage;

photoDataRow[ 'scaleContent' ] = true;

photoDataRow[ 'photoVO' ] = activePhotoVO;

_thumbsDP.addItem( photoDataRow );


loadNextImage();
```

A `photoDataRow` object is created, holding the returned image and metaproperties such as `label`, `scaleContent`, and `photoVO`. This is added to the `_thumbsDP ArrayCollection` using the `addItem` method.

Then the `loadNextImage` method is called, which repeats the actions for each element in the response data.

The ActionScript class can now easily be implemented in an AIR application. The following is the MXML code that uses the class you just created:

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

<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
 initialize="init()"
 showFlexChrome="false"
 showStatusBar="false">

<mx:DefaultTileListEffect id="TLEffect"
fadeOutDuration="350"
fadeInDuration="350"
moveDuration="1000" />

<mx:Script>
<![CDATA[

    import mx.controls.Alert;
    import flash.events.MouseEvent;
    import com.oreilly.aircookbook.bonus.FlickrWS;

    [Bindable]
    private var fsClass:FlickrWS;

    private function init():void
    {
        fsClass = new FlickrWS();
        searchBtn.addEventListener( MouseEvent.CLICK, startNewSearch );
    }

    private function startNewSearch( evt:MouseEvent ):void
    {

        if( searchTxt.text == "" )
        {
            return;
        }

        fsClass.thumbsDP.removeAll();

        fsClass.currImage = 0;

        var tagSearch:String;
        var generalSearch:String;

        if( searchCbx.selectedLabel == "tag" )
        {
            tagSearch = searchTxt.text;
            generalSearch = "";

        }else {
            tagSearch = "";
            generalSearch = searchTxt.text;
        }

        fsClass.fs.photos.search("", tagSearch, "any", generalSearch, null, null, null, null,
                                 -1, "", 20, 1, "interestingness-desc");

    }
]]>
</mx:Script>


<mx:TextInput id="searchTxt" />

<mx:Button label="Search" id="searchBtn" />

<mx:TileList id="photoTL" dataProvider="{fsClass.thumbsDP}"
 columnCount="4" rowCount="3" rowHeight="155" columnWidth="175"
 itemRenderer="com.oreilly.aircookbook.bonus.FlickrThumbnail"
 itemsChangeEffect ="{TLEffect}"
 dragEnabled="true"
 dropEnabled="true"
 dragMoveEnabled="true"  />

<mx:ComboBox id="searchCbx">
<mx:dataProvider>
<mx:Object label="Tag Search" data="tag" />
<mx:Object label="Text Search" data="text" />
</mx:dataProvider>
</mx:ComboBox>
</mx:WindowedApplication>
```

The application has a ComboBox control to allow the user to launch the search by using the text or a tag (inserted by the user) as a parameter in a TextInput control. The search on the Flickr web services actually occurs when the startNewSearch method is invoked, triggered by the click event of the Button control. The search method of the Flickr SWC library is invoked, and the parameters it needs to obtain the images from the Flickr web service are sent to it:

```
fsClass.fs.photos.search("", tagSearch, "any", generalSearch, null, null, null, null,
-1, "", 20, 1, "interestingness-desc");
```

The request data is controlled by the onPhotoList method of the ActionScript class, which populated the thumbsDP data provider typed as ArrayCollection. This value is passed onto the TileList control, creating a data binding:

```
<mx:TileList id="photoTL" dataProvider="{fsClass.thumbsDP}"
 columnCount="4" rowCount="3" rowHeight="155" columnWidth="175"
 itemRenderer="com.oreilly.aircookbook.bonus.FlickrThumbnail"
 itemsChangeEffect ="{TLEffect}"
 dragEnabled="true"
 dropEnabled="true"
 dragMoveEnabled="true"  />
```

This control uses an item renderer to display the image and text as an element of the list. The renderer is passed to it with the `itemRenderer` property, where an external component, called `FlickrThumbnail`, is specified in `com.oreilly.aircookbook.bonus`.

The `TileList` control also makes it possible to drag and drop items in it by applying the specified effect to the movements at the beginning of the MXML application:

```
<mx:DefaultTileListEffect id="TLEffect"
fadeOutDuration="350"
fadeInDuration="350"
moveDuration="1000" />
```

This is the code of the `FlickrThumbnail.mxml` component, which exploits the data property that is propagated by the `TileList` in the item renderer and contains the data provider and all its properties:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center"
    verticalGap="0"
    borderStyle="none" backgroundColor="white" >

    <mx:Image id="image" width="60" height="60" source="{data.source}"/>
    <mx:Label text="{data.label}" width="120" textAlign="center"/>


</mx:VBox>
```

This simple mashup implements and consumes only one method from the Flickr APIs. You could improve it by adding remote features and by exploiting AIR SDKs to allow users to save images locally or create a local database with SQLite to store their favorite images.

HTML/JavaScript

You can also consume Flickr services by using HTML and JavaScript. Before you begin, it's worth consulting the API explorer tool (*http://www.flickr.com/services/api/explore*), with which you can query any method of Flickr's APIs, send parameters to it, and display the response. The syntax is `http://www.flickr.com/services/api/explore/?method=METHODNAME`. For example, the following URL invokes the `search` method of the APIs, and, in the page that opens, you can send the parameters by filling in the text inputs:

`http://www.flickr.com/services/api/explore/?method=flickr.photos.search`

What the API explorer actually does is build the web page to pass the parameters of the method using the reflection approach and build the REST URL of that method.

Once you get familiar with Flickr's APIs, you can start interacting with them. By using a Flickr *badge*, for example, you can create a web page showing your photos by importing the badge from *http://www.flickr.com/badge_new.gne*. Then again, you can create your own Flickr badge to insert in your AIR application with a few lines of JavaScript code.

Flickr generates feeds in different formats and for different types of content, so your application can consume these feed links in various languages. You can, for example, consume the feeds to obtain recent public photos that have been uploaded on Flickr from *http://api.flickr.com/services/feeds/photos_public.gne* or your own public photos from *http://api.flickr.com/services/feeds/photos_public.gne?id=USER-NSID*.

Now all you have to do is choose the format to consume: RSS, ATOM, PHP, JSON, CSV, YAML, or SQL. The example uses the JavaScript Object Notation (JSON).

> According to Wikipedia, JSON is a lightweight computer data interchange format. It is a text-based, human-readable format to represent simple data structures and associative arrays (called *objects*). The JSON format is often used for transmitting structured data over a network connection in a process called *serialization*. Its main application is in Ajax web application programming, where it serves as an alternative to the XML format. Although JSON was based on a subset of JavaScript and is commonly used with that language, it is considered to be a language-independent data format. Code for parsing and generating JSON data is readily available for a large variety of programming languages. You can find a comprehensive listing of existing JSON bindings, organized by language, at *http://www.json.org*.

When you launch the URL *http://api.flickr.com/services/feeds/photos_public.gne?id=44124469126@N01&format=json* from your browser, the following JavaScript code is generated:

```
jsonFlickrFeed({
        "title": "Uploads from Marco Casario",
        "link": "http://www.flickr.com/photos/44124469126@N01/",
        "description": "",
        "modified": "2008-07-27T21:10:55Z",
        "generator": "http://www.flickr.com/",
        "items": [
      {
            "title": "STA72737",
            "link": "http://www.flickr.com/photos/44124469126@N01/2708076572/",
            "media": {"m":"http://farm4.static.flickr.com/3115/2708076572_505dc798eb_m.jpg"},
            "date_taken": "2008-07-27T23:10:55-08:00",
            "description": "&lt;p&gt;&lt;a
href=&quot;http://www.flickr.com/people/44124469126@N01/&quot;&gt;Marco
Casario&lt;/a&gt; posted a photo:&lt;/p&gt; &lt;p&gt;&lt;a
href=&quot;http://www.flickr.com/photos/44124469126@N01/2708076572/&quot;
title=&quot;STA72737&quot;&gt;&lt;img
src=&quot;http://farm4.static.flickr.com/3115/2708076572_505dc798eb_m.jpg&quot;
width=&quot;240&quot; height=&quot;180&quot; alt=&quot;STA72737&quot; /&gt;&lt;/a&gt;&lt;/p&gt; ",
            "published": "2008-07-27T21:10:55Z",
            "author": "nobody@flickr.com (Marco Casario)",
                "author_id": "44124469126@N01",
                "tags": "seychelles"
        },
        {
                "title": "STA72447",
                "link": "http://www.flickr.com/photos/44124469126@N01/2708076122/",
                "media": {"m":"http://farm4.static.flickr.com/3118/2708076122_e04bf714d7_m.jpg"},
                "date_taken": "2008-07-27T23:10:46-08:00",
                "description": "&lt;p&gt;&lt;a
href=&quot;http://www.flickr.com/people/44124469126@N01/&quot;&gt;Marco
Casario&lt;/a&gt; posted a photo:&lt;/p&gt; &lt;p&gt;&lt;a
href=&quot;http://www.flickr.com/photos/44124469126@N01/2708076122/&quot;
title=&quot;STA72447&quot;&gt;&lt;img
src=&quot;http://farm4.static.flickr.com/3118/2708076122_e04bf714d7_m.jpg&quot;
width=&quot;240&quot; height=&quot;180&quot; alt=&quot;STA72447&quot; /&gt;&lt;/a&gt;&lt;/p&gt; ",
                "published": "2008-07-27T21:10:46Z",
                "author": "nobody@flickr.com (Marco Casario)",
                "author_id": "44124469126@N01",
                "tags": "seychelles"
        },
....
}
]
})
```

You can use this feed to dynamically create an HTML page by using JavaScript only. The following is the JavaScript code, saved to a file called *FlickrBadge.js*, that creates the elements in an HTML page and displays the photos of a Flickr user:

```
function jsonFlickrFeed(feed)
{

  var flickrRegExp = /(http:\/\/farm4.static.flickr.com\/\d+\/\d+_[0-9a-z]+)_m\.jpg/;
  var htmlContent = "";
  var items = feed['items'];

  htmlContent += '<div id="flickr_badge_uber_wrapper">';
  htmlContent += '<p class="title"><nobr>Go to</nobr><a href="' + feed['link'] + '">' +
                 feed['title'] + '</a></p>';

  htmlContent += '<ul id="flickr_badge_source">';


  for (var i = 0; i < items.length; i++)
  {
    var data = flickrRegExp.exec(items[i]['description']);

    if (data != null)
    {
      var image = data[1] + '_s.jpg';

      htmlContent += '<li class="flickr_badge_image"><a href="' + items[i]['link'] + '"
                   id="flickr_www"><img src="'
          + image + '" /></a></li>';
    }
  }
  htmlContent += '</ul><p id="flickr_badge_source_txt"><nobr>Go to</nobr> <a href="' +
               feed['link'] + '">' + feed['title'] + '</a></p>'

  htmlContent += '</div>';

  document.writeln(htmlContent);
}
```

This script dynamically generates an HTML `<ul>` list of elements that repeats a `for` loop as many times as the number of items that are returned from the feed.

Most of the work is carried out by the regular expression, which returns the string that matches the URL of the Flickr image:

```
var flickrRegExp = /(http:\/\/farm4.static.flickr.com\/\d+\/\d+_[0-9a-z]+)_m\.jpg/;
```

The URL address *http://farm4.static.flickr.com* may change for your photos. Pay attention to the URL used by Flickr to store your public photos in the `media` property of the JSON feed:

```
"media": {"m":"http://farm4.static.flickr.com/3115/2708076572_505dc798eb_m.jpg"},
```

The response data of the method `flickrRegExp.exec(items[i]['description'])` actually creates the image list.

Now you can create the web page that calls the JSON feed and this simple JavaScript library:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<link rel="stylesheet" type="text/css" href="css/main.css"/>

<script type="text/javascript" src="FlickrBadge.js"></script>

<script
src='http://api.flickr.com/services/feeds/photos_public.gne?id=44124469126@N01&format=json'
type='text/javascript'></script>

<title>Flickr Badge</title>
</head>

<body>
</body>
</html>
```

This HTML page doesn't specify any content in its body. All the work is carried out by the JavaScript contained in the *FlickrBadge.js* file. The *main.css* file applies a style formation to the page with the following CSS syntax:

```
body {padding:0; font: 12px Arial, Helvetica, Sans serif; color:#666666;}
```

```
.flickr_badge_image {text-align:center !important;display: inline;list-style-type: none;}

.flickr_badge_image img {border: 1px solid black !important;}

#flickr_badge_uber_wrapper {width:120px;}
#flickr_www {display:block; text-align:center; padding:0 10px 0 10px !important;
font: 11px Arial, Helvetica, Sans serif !important; color:#3993ff !important;}

#flickr_badge_uber_wrapper a:hover,
#flickr_badge_uber_wrapper a:link,
#flickr_badge_uber_wrapper a:active,
#flickr_badge_uber_wrapper a:visited {text-decoration:none !important; background:inherit
!important;color:#3993ff;}
#flickr_badge_wrapper {background-color:#ffffff;border: solid 1px #000000}

#flickr_badge_source {padding:0 !important; font: 11px Arial, Helvetica, Sans serif
!important; color:#666666 !important;width: 190px; margin: 0; padding: 0;}
```

18.2 Consuming Yahoo Maps Web Services

Problem

You want to consume Yahoo Maps (*http://developer.yahoo.com/maps/*) web services.

Solution

Yahoo Maps web services allow you to integrate interactive map systems in AIR desktop applications. Specifically, the Yahoo Maps APIs enable you to work with map applications and add more features to them, adding value to your applications. You can find loads of useful services on the Web that consume Yahoo APIs to create rich user experiences:

Upcoming.org Events (http://www.androidtech.com/upcoming-events-map/upcoming-maps-2.php)
Robert Oschler uses the Yahoo Maps JS-Flash API to display Upcoming.org events along with traffic, ATMs, bars, and restaurants.

Instant Local Search and Traffic Plotter (http://theurer.cc/maps/y/)
This application combines Yahoo Maps' Ajax with three new Yahoo REST services: Geocoding, Local Search, and Traffic. It also comes with explanations on what needs to be done to write an Ajax API mashup, which is a great way to get started.

Local Events Browser (http://local.yahooapis.com/eb/)
An innovative group of Yahoo employees pushed Ajax to new levels with the Local Events Browser that blends multiple Yahoo APIs to put events on an interactive map with useful features such as a calendar, tag cloud, and built-in image search.

Flickr Maps (http://www.sodascope.com/FlickrMapsExt/)
For cities across the United States, Michael Hoch puts Flickr photos on Yahoo Maps using the Flex API. This application is another great example of how the

Yahoo Maps APIs give you full control over the look and feel of your mapping application.

You can find a list of Yahoo Maps mashups at *http://developer.yahoo.com/maps/applications.html*.

In this recipe, you will learn how to consume Yahoo Maps web services to create mashups in AIR.

Discussion

Yahoo provides two kind of APIs, according to the programming language you use:

- *The Ajax API* is designed to use DHTML and JavaScript to host maps. Yahoo provides the JavaScript functions to make mapping a breeze.
- *The ActionScript 3 Flash APIs* are designed to embed a map into your website or desktop application using the ActionScript 3 Maps API with Adobe Flash and Flex.

This dual possibility is a perfect match with AIR, which allows you to integrate Ajax as well as ActionScript applications.

Before beginning to write any code, you have to register at *http://developer.yahoo.com/wsregapp* to obtain an application ID.

Using Ajax, the first step to take is to import the Yahoo Maps Ajax API library in your web page by accessing the API's remote JavaScript script:

```
<script type="text/javascript"
 src="http://api.maps.yahoo.com/ajaxymap?v=3.8&appid=YOUR-APP-ID">
</script>
```

The Ajax API isn't on your server and isn't installed. Once the JavaScript file that is specified in the `<script>` tag is loaded, the web page is ready to use the maps.

At this point, you can create a container in the web page to display the map:

```
<body>
<div id="Ymap"></div>
OTHER CONTENT
 </body>
```

All you have to do now is add the functions and controls to the `Map` objects with some JavaScript code:

```
<style type="text/css">
    #YMap
    {
    height: 400px;
    width: 500px;
    }
</style>
<script type="text/javascript">

    var YMap
    function initYMap()
    {
        YMap = new YMap(document.getElementById('YMapDiv'));
        YMap.addPanControl();
        YMap.addTypeControl();
        YMap.addZoomLong();
        YMap.drawZoomAndCenter("New York", 5);
    }

     window.onload = initYMap;
 </script>

</head>

 <body>
 <div id="YMapDiv"></div>
</body>
```

This script, with just a few lines of code, creates a map that will be placed in the `YMapDiv` `div` container and adds three map controls:

- `Pan` to have north, south, east, and west directional controls.
- `Type` to add the ability to change between satellite, hybrid, and regular maps.
- `Zoom` to add a zoom feature. `Long` specifies a slider vs. + and – controls.

Finally, the `drawZoomAndCenter` specifies the map's starting location (New York City for the example) and zoom level.

By using the Yahoo Maps ActionScript 3 component, you can also embed maps in your Flash, Flex, or AIR application to add a map to a wide range of web and desktop applications. Once you've obtained an application ID, instead of using Ajax APIs where there is nothing to install, with ActionScript you have to download and install the component released as a SWC file from *http://developer.yahoo.com/flash/maps/getLatest.php*.

A SWC file is an archive file for Flash and Flex components that makes it easy to exchange components among Flash and Flex developers. To get it into your Flex 3 project, just drop it in your libs folder, or point to it in your project build path. Using Flash, install the provided MXP via the Adobe Extension Manager.

After you have installed the Yahoo Maps ActionScript 3 component, you have to instance a `YahooMap` object to embed a map:

```
import com.yahoo.maps.api.YahooMap;

var mapContainer:UIComponent = new UIComponent();
Ymap.init("YOUR-APP-ID", stage.stageWidth, stage.stageHeight);
```

```
    mapContainer.addChild(Ymap);
    addChild(mapContainer);
```

Like for any other mashup application, you can't start developing if you don't have a good idea of the APIs that are available. A good preparatory step is to refer to the class reference documentation to get familiar with the APIs. You can find the ActionScript 3 class reference documentation at *http://developer.yahoo.com/flash/maps/classreference/index.html* and the Maps Ajax API Reference Manual at *http://developer.yahoo.com/maps/ajax/V3.8/index.html*.

Another wise preparation that will save you time in the long run is to clarify the objectives of your mashup. You should have a good idea of the utility of the service that you are integrating and how the user will use it. If your ideas aren't clear, you can get more ideas by consulting the Yahoo Maps application gallery (*http://gallery.yahoo.com/maps*), which features working examples that you can use as a starting point for your own applications.

ActionScript/Flex

To create a mashup that uses the Yahoo Maps API, you have to import the Yahoo Maps ActionScript 3 component. If you are using Flash Professional CS4 as an IDE, all you have to do is install the provided MXP using the Adobe Extension Manager and drag the component onto the Stage. In Flex 3, you have to copy the SWC file in the libs folder of your Flex3 project or point to it within your project build path.

This example uses the functions to control the markers in a Yahoo Maps map and saves a search in a local XML file.

The first step is to create an MXML file, called `YMapsLocalSearch.as`, to begin instancing the `YahooMap` class. Insert a `Script` block with the `init` function:

```
    <mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import mx.controls.AdvancedDataGrid;
        import com.yahoo.maps.api.intl.MapLocales;
        import mx.collections.ArrayCollection;
        import mx.events.ResizeEvent;
        import com.yahoo.maps.webservices.geocoder.GeocoderResult;
        import com.yahoo.maps.webservices.geocoder.GeocoderResultSet;
        import com.yahoo.maps.webservices.geocoder.events.GeocoderEvent;
        import com.yahoo.maps.api.core.location.Address;
        import com.yahoo.maps.api.core.location.LatLon;
        import com.yahoo.maps.api.YahooMapEvent;
        import com.yahoo.maps.api.YahooMap;
        import com.yahoo.maps.api.markers.SimpleMarker;


        private  const  YAHOOID:String = " YOUR_APP_ID ";


        private var _yahooMap:YahooMap;
        private var _address:Address;
        private var file:File;
        private var stream:FileStream;

        [Bindable] private var _geocoderResults:ArrayCollection;

        private function init():void
        {

            // create YahooMap instance and listen for map initialize event
            _yahooMap = new YahooMap();
            _yahooMap.addEventListener(YahooMapEvent.MAP_INITIALIZE, handleMapInitialize);
            _yahooMap.init(YAHOOID, mapContainer.width, mapContainer.height);

            _yahooMap.addPanControl();
            _yahooMap.addScaleBar();
            _yahooMap.addTypeWidget();
            _yahooMap.addZoomWidget();

            mapContainer.addChild(_yahooMap);

            _geocoderResults = new ArrayCollection();
            gResultPanel.visible=false;

            file = File.desktopDirectory.resolvePath("Ymarkers.xml");
            stream = new FileStream();

            if (file.exists)
            {

            stream.openAsync(file, FileMode.READ);

            stream.addEventListener(ProgressEvent.PROGRESS,progressHandler);
            stream.addEventListener(Event.COMPLETE, completeHandler);
            return

            }

        }
```

The `addChild` method adds the map to the `DisplayList` of an instance of the `UIComponent` in the application's view:

```
    <mx:UIComponent id="mapContainer" width="100%" bottom="0" top="35"/>
```

A `TextInput` control and a `Button` control, which will make up the UI elements of the application, allow the user to insert an address, view it on the map, and save it in a local file. The event handler that responds to the click of the button is as follows:

```
    private function getTextInput(evt:MouseEvent):void
    {
        var searchStr:String = searchInput.text;

        searchInput.text = "Looking up address...";

        _address = new Address(searchStr);
        _address.addEventListener(GeocoderEvent.GEOCODER_SUCCESS, handleGeocodeSuccess);
        _address.geocode();
    }
```

When the `GEOCODER_SUCCESS` event, contained in the `GeocoderEvent` class, is triggered, it triggers the `handleGeocodeSuccess` handler, which loops through each result and adds an item to the suggest list (the `TextInput` control in the view):

```
    private function handleGeocodeSuccess(event:GeocoderEvent):void
    {
        var geocoderResults:GeocoderResultSet = _address.geocoderResultSet;
        // if we only have one result, set the location right away.

        if(geocoderResults.found == 1)
        {
            setLocation(geocoderResults.firstResult);
        }
        else if(geocoderResults.found > 1)
        {
            // loop through each result and add an item to the suggest list.
            _geocoderResults = new ArrayCollection();
            var len:int = results.length;
            for(var i:int=0; i<len; i++)
            {
                var result:GeocoderResult = results[i];

                var data:Object = {
                    label: result.getLineAddress(),
                    data:result
                }

                _geocoderResults.addItem(data);
            }

            //show the suggestions panel
            gResultPanel.visible=true;
        }else{
            // reset the text
            searchInput.text = "";
        }
    }
```

You specify the center `LatLon` point of the map in the `setLocation` method. A marker is dynamically created from the `SimpleMarker` class and placed at the address specified by the user in the `TextInput`. Finally, the `saveMarker` method is invoked, which saves the address to a local file:

```
    private function setLocation(geocoderResult:GeocoderResult):void
    {
    _yahooMap.zoomLevel = geocoderResult.zoomLevel;
    _yahooMap.centerLatLon = geocoderResult.latlon;
    searchInput.text = geocoderResult.getLineAddress();

    var marker:SimpleMarker = new SimpleMarker();
```

```
marker.address = new Address(geocoderResult.getLineAddress());

_yahooMap.markerManager.addMarker(marker);
saveMarker(geocoderResult.getLineAddress())
}
```

The Geocoder allows you to find the specific latitude and longitude for an address string, while the GeocoderEvent class represents the event object passed to the event listener for events dispatched by the Geocoder object.

The complete code for the Flex application is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
creationComplete="init()"
 layout="absolute"
  showFlexChrome="false"
showStatusBar="false">


<mx:Script>
<![CDATA[
    import mx.controls.Alert;
    import mx.controls.AdvancedDataGrid;
    import com.yahoo.maps.api.intl.MapLocales;
    import mx.collections.ArrayCollection;
    import mx.events.ResizeEvent;
    import com.yahoo.maps.webservices.geocoder.GeocoderResult;
    import com.yahoo.maps.webservices.geocoder.GeocoderResultSet;
    import com.yahoo.maps.webservices.geocoder.events.GeocoderEvent;
    import com.yahoo.maps.api.core.location.Address;
    import com.yahoo.maps.api.core.location.LatLon;
    import com.yahoo.maps.api.YahooMapEvent;
    import com.yahoo.maps.api.YahooMap;
    import com.yahoo.maps.api.markers.SimpleMarker;


    private  const  YAHOOID:String = " YOUR_APP_ID ";


    private var _yahooMap:YahooMap;
    private var _address:Address;
    private var file:File;
    private var stream:FileStream;

    [Bindable] private var _geocoderResults:ArrayCollection;

    private function init():void
    {

        // create YahooMap instance and listen for map initialize event
        _yahooMap = new YahooMap();
        _yahooMap.addEventListener(YahooMapEvent.MAP_INITIALIZE, handleMapInitialize);
        _yahooMap.init(YAHOOID, mapContainer.width, mapContainer.height);

        _yahooMap.addPanControl();
        _yahooMap.addScaleBar();
        _yahooMap.addTypeWidget();
        _yahooMap.addZoomWidget();

        mapContainer.addChild(_yahooMap);

        _geocoderResults = new ArrayCollection();
        gResultPanel.visible=false;

        file = File.desktopDirectory.resolvePath("Ymarkers.xml");
        stream = new FileStream();

        if (file.exists)
        {

        stream.openAsync(file, FileMode.READ);

        stream.addEventListener(ProgressEvent.PROGRESS,progressHandler);
        stream.addEventListener(Event.COMPLETE, completeHandler);
        return

        }

    }

    private function handleMapInitialize(event:YahooMapEvent):void
    {
        mapContainer.addEventListener(ResizeEvent.RESIZE, handleContainerResize);

        _yahooMap.zoomLevel = 13;
        _yahooMap.centerLatLon = new LatLon(40.81,-96.7);
    }

    private function handleContainerResize(event:ResizeEvent):void
    {
    _yahooMap.setSize( mapContainer.width, mapContainer.height );
    }

    private function getTextInput():void
    {
        var searchStr:String = searchInput.text;

        searchInput.text = "Looking up address...";

        _address = new Address(searchStr);
        _address.addEventListener(GeocoderEvent.GEOCODER_SUCCESS, handleGeocodeSuccess);
        _address.geocode();
    }

    private function handleGeocodeSuccess(event:GeocoderEvent):void
    {
        var geocoderResults:GeocoderResultSet = _address.geocoderResultSet;
        var results:Array = geocoderResults.results;

    // if we only have one result, set the location right away.
    if(geocoderResults.found == 1)
    {
        setLocation(geocoderResults.firstResult);
    }

    else if(geocoderResults.found > 1)
    {
        // loop through each result and add an item to the suggest list.
        _geocoderResults = new ArrayCollection();
        var len:int = results.length;
        for(var i:int=0; i<len; i++)
        {
            var result:GeocoderResult = results[i];

            var data:Object = {
            label: result.getLineAddress(),
            data:result
            }

            _geocoderResults.addItem(data);
        }

        //show the suggestions panel
        gResultPanel.visible=true;

    }else{

        // reset the text
        searchInput.text = "";
    }
    }

    private function setLocation(geocoderResult:GeocoderResult):void
    {
        _yahooMap.zoomLevel = geocoderResult.zoomLevel;
        _yahooMap.centerLatLon = geocoderResult.latlon;
        searchInput.text = geocoderResult.getLineAddress();

        var marker:SimpleMarker = new SimpleMarker();

        marker.address = new Address(geocoderResult.getLineAddress());
        /*
        marker.addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        marker.addEventListener(Event.REMOVED_FROM_STAGE, onRemoved);
        */
        _yahooMap.markerManager.addMarker(marker);
        saveMarker(geocoderResult.getLineAddress())
    }

    private function handleListChange():void
    {
        // get the selected geocoder result object
        var selectedResult:GeocoderResult = geocoderResultsList.selectedItem.data
                                                               as GeocoderResult;
        setLocation(selectedResult);

        _geocoderResults.removeAll();
        gResultPanel.visible=false;
    }

    private function saveMarker(add:String):void
    {
        var xmlData:XML = <marker/>;

        xmlData.address = add;
        xmlData.saveDate = new Date().toString()

        var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
        outputString += xmlData.toXMLString();
        outputString = outputString.replace(/\n/g, File.lineEnding);


        stream.open(file, FileMode.WRITE);
        stream.writeUTFBytes(outputString);

        stream.close();
    }

    private function progressHandler(event:ProgressEvent):void
    {
        // opening in progress ....
    }

    private function completeHandler(event:Event):void
    {

        var xmlData:XML = XML(stream.readUTFBytes(stream.bytesAvailable));
        stream.close();

        var searchStr:String = xmlData.address;

        searchInput.text = "Looking up address...";

        _address = new Address(searchStr);
        _address.addEventListener(GeocoderEvent.GEOCODER_SUCCESS, handleGeocodeSuccess);
        _address.geocode();
    }
]]>
</mx:Script>

<mx:UIComponent id="mapContainer" width="100%" bottom="0" top="35"/>

<mx:Canvas id="header" width="100%" height="35" backgroundColor="#E5E5E5"
backgroundAlpha="0.85">

<mx:TextInput id="searchInput" width="275" verticalCenter="0" left="15"
```

```
enter="getTextInput();"/>

<mx:Button label="Search and Save" cornerRadius="1" verticalCenter="0" left="288"
click="getTextInput()"/>

</mx:Canvas>

<mx:Panel width="343" height="200" layout="absolute" id="gResultPanel" title="Results"
left="35" top="45">

<mx:List left="0" right="0" top="0" bottom="0" id="geocoderResultsList"
dataProvider="{_geocoderResults}" change="handleListChange();"/>

</mx:Panel>

</mx:WindowedApplication>
```

This code is based on and extends the examples of mashups of Yahoo Maps, which are licensed under the BSD (revised) open source license. You can find other examples at *http://developer.yahoo.com/flash/maps/examples*.

JavaScript/HTML

In JavaScript, the steps to create the mashup example aren't that different. The only difference is in importing the remote script to instance the Yahoo Maps map:

```
<script src="http://api.maps.yahoo.com/ajaxymap?v=3.4&appid=YOUR_MAP_ID"
type="text/javascript"></script>
```

The web page will use an `iframe` control to load the map in the nonapplication sandbox. This HTML page will be loaded in the `iframe`, which is essentially the application root. Following is the code for the parent page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

    <script type="text/javascript" src="frameworks/AIRAliases.js" />
    <script type="text/javascript" src="frameworks/AIRIntrospector.js" />

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>AIR Cookbook: 18.2 Consuming Yahoo! Maps web services (JavaScript)</title>
</head>

<body>
<h1>AIR Cookbook: 18.2 Consuming Yahoo! Maps web services (JavaScript)</h1>

<iframe
id="yahoomap"
src="mapIframe.html"
```

```
sandboxRoot="http://SomeRemoteDomain.com/"
documentRoot="app:/" width="500" height="500"
/>

</body>
</html>
```

This technique allows you to go beyond the limitations of the security sandboxes. There are two security sandboxes in AIR: the nonapplication sandbox and the application sandbox. The *nonapplication sandbox* operates just like the browser and doesn't provide access to the AIR APIs. The *application sandbox* has restrictions on the JavaScript that can be executed, but it has full access to the AIR APIs. This would also directly impact the Yahoo Maps API.

You can learn more about the important issue of AIR security by reading the HTML Security FAQ (*http://labs.adobe.com/wiki/index.php/AIR:HTML_Security_FAQ*) and the "Ajax and Mashup Security" white paper from the Open Ajax Alliance (*http://www.openajax.org/whitepapers/ajax%20and%20mashup%20security.html*).

The core of the application is written in the *mapIframe.html* page, which is imported in the `iframe` and which displays a text input where the user can insert the address and a button. The script to instance the Yahoo Maps map is loaded in this page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script src="http://api.maps.yahoo.com/ajaxymap?v=3.4&appid=YOUR_YAHOO_APP_ID_HERE"
type="text/javascript />

<script type="text/javascript" src="mapIframe.js" />

<style>
    #map {
    width: 100%;
    height: 450;
    }
</style>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Yahoo! Maps</title>
</head>
<body onLoad="init();">

    <input id="location" type="text" value="Insert the address" />

    <input type="submit" name="go" id="go" value="Show destination" onclick="loadAddress()" />

    <div id="map"></div>

</body>
</html>
```

By using a technique known as *bridging*, it is possible to access the assets that are specified in the HTML root page, the one that uses the iframe. This way, it is possible to access the nonapplication sandbox to call functions in the application sandbox. By using the parentSandboxBridge property, the application sandbox can set up a bridge interface:

```
var parentBridge = new Object();
parentBridge.doSomething = doSomething;
parentBridge.doSomethingElse = doSomethingElse;
document.getElementById( 'yahoomap' ).contentWindow.parentSandboxBridge = parentBridge;
```

The yahoomap element is the id of the iframe in the root page:

```
<iframe
id="yahoomap"
src="mapIframe.html"
sandboxRoot="http://SomeRemoteDomain.com/"
documentRoot="app:/" width="500" height="500"
/>
```

This way, the nonapplication sandbox code can invoke the methods doSomething and doSomethingElse from the application sandbox using the following code:

```
window.parentSandboxBridge. doSomething ();
```

This is the code for the imported mapIframe.js script:

```
// const
var DEFAULT_TEXT = 'Insert the address';
var STARTADDR = 'Rome, Italy';

var map = null;

function init()
{
    map = new YMap( document.getElementById( 'map' ) );
    map.addTypeControl();
    map.addPanControl();
    map.addZoomShort();
    map.drawZoomAndCenter( STARTADDR, 5 );

    // Add an event to report to our Logger
    YEvent.Capture(map, EventsList.MouseClick, reportPosition);

}

function loadAddress()
{

    var geo = document.getElementById( 'location' ).value;

    if( geo != DEFAULT_TEXT )
    {
          map.drawZoomAndCenter( geo );
    }
}
```

```
 function reportPosition(_e, _c)
{
    // It is optional to specify the location of the Logger.
    // Do so by sending a YCoordPoint to the initPos function.
    var mapmapCoordCenter = map.convertLatLonXY(map.getCenterLatLon());
    YLog.initPos(mapCoordCenter); //call initPos to set the starting location
    // Printing to the Logger
    YLog.print("You Made a MouseClick!");
    YLog.print("Latitude:" + _c.Lat);
    YLog.print("Longitude:" + _c.Lon);

    YLog.print("Adding marker at....");
    YLog.print("Latitude:" + _c.Lat + "Longitude:" + _c.Lon);
 }
```

The map is loaded and positioned on the point chosen by the user in the text input in the `loadAddress` function. The map is placed in the new location with the `drawZoomAndCenter(geo)` method.

With a small specification that uses the application sandbox as a proxy between sandboxes, you can run this application using AIR.

18.3 Consuming Twitter APIs

Problem

You want to access the Twitter APIs web services.

Solution

Twitter was born from the simple idea of communicating quickly with friends via brief text messages, but its success also contributed to its transformation, and today the service is also used for commercial purposes to communicate new initiatives, software releases, and much more.

According to Wikipedia, Twitter is a free social networking and microblogging service that allows users to send updates (otherwise known as *tweets*), which are text-based posts of up to 140 characters. Updates are displayed on the user's profile page and delivered to other users who have signed up to receive them. The sender can restrict delivery to those in their circle of friends (delivery to everyone being the default). Users can receive updates via the Twitter website, instant messaging, SMS, RSS, or email, as well as through an application such as Twitterrific or Facebook. For SMS, four gateway numbers are currently available: short codes for the United States, Canada, and India, and a United Kingdom number for international use. Several third parties offer the ability to post and receive updates via email.

This is a sampling of AIR applications that consume Twitter's APIs:

- Twhirl (*http://www.twhirl.org*)
- Spaz (*http://funkatron.com/spaz*); developed using Ajax
- Snitter (*http://getsnitter.com*)
- Tweetr (*http://www.tweet-r.com*)

This recipe uses Twitter's APIs to create a simple AIR application with ActionScript and JavaScript that displays the most recent messages and account details.

Discussion

Twitter provides its own set of APIs to consume its web services and create mashups that use its services. This solution will use the simplest approach to consume Twitter's APIs, without carrying out the authentication procedure to retrieve information about protected users who aren't friends.

You can find more information to study Twitter's APIs at the following addresses:

- *http://apiwiki.twitter.com/Twitter*: The API home page
- *http://apiwiki.twitter.com/REST+API+Documentation*: API documentation, which is a reference for developers building tools that talk to Twitter
- *http://groups.google.com/group/twitter-development-talk*: Twitter Development Talk, where you can talk about your Twitter applications with other developers
- *http://code.google.com/p/twitterscript*:
  An ActionScript 3 library for accessing Twitter APIs.

ActionScript/Flex

An open source library that began as a Twitter project, the twitterscript ActionScript 3 library (*http://apiwiki.twitter.com*) is a great help when using the Twitter APIs with ActionScript.

It's quite easy to use this library. After you download `TwitterApi.swc` and import it into your Flash or Flex project, you can import and use its classes. For example, the `loadUserTimeline` method of the `Twitter` class allows you to load the timeline of a user by assigning it a nickname as a parameter:

```
tw = new Twitter();
tw.addEventListener(TwitterEvent.ON_USER_TIMELINE_RESULT, loadedUserTimeline );
tw.loadUserTimeline( nick );
```

The `nick` parameter contains the screen name of the Twitter user. The `ON_USER_TIMELINE_RESULT` event of the `TwitterEvent` class is triggered when the list of the timeline of messages is loaded. In the `loadedUserTimeline` event handler, you can load the messages in a variable, which will then be used as a data provider in the application:

```
private function loadedUserTimeline( evt:TwitterEvent ):void
{
    var timeline:Array = evt.data as Array;

    // populate messages list dataprovider
    var twitStatus:TwitterStatus;
    for each( twitStatus in timeline )
    {
        _timelineDP.addItem( twitStatus );
    }
    _logedUserProfile = twitStatus.user;
}
```

The `data` property of the `event` object contains an array of timeline messages. With a `for...each` loop, you can control all the `TwitterStatus` objects in the timeline messages.

The complete ActionScript class that pulls the latest messages from a Twitter user using a nickname is as follows:

```
package com.oreilly.aircookbook.bonus
{

    import mx.collections.ArrayCollection;
    import mx.controls.Alert;

    import twitter.api.Twitter;
    import twitter.api.data.TwitterStatus;
    import twitter.api.data.TwitterUser;
    import twitter.api.events.TwitterEvent;

    public class TwitterConn
    {
        private var tw:Twitter;

        private var _userDetails:TwitterUserVO;
        public function get userDetails():TwitterUserVO
        {
        return _userDetails;
        }

        private var _logedUserProfile:TwitterUser;
        public function get logedUserProfile():TwitterUser
        {
        return _logedUserProfile;
        }
        private var _timelineDP:ArrayCollection;

        public function get timelineDP():ArrayCollection
        {
        return _timelineDP;
        }
```

```
            public function TwitterConn()
            {
            tw = new Twitter();
            tw.addEventListener( TwitterEvent.ON_USER_TIMELINE_RESULT, loadedUserTimeline );
            _timelineDP = new ArrayCollection();
            }

            public function loadUserTimeline( nick:String ):void
            {
            if( nick == "" )
            return;

            tw.loadUserTimeline( nick );
            }

            // called when user timeline is returned from webservices
            private function loadedUserTimeline( evt:TwitterEvent ):void
            {
            // access timeline messages array
            var timeline:Array = evt.data as Array;

            // clear previous messages list contents
            _timelineDP.removeAll();

            // if no timeline found for specified user
            // exit function execution
            // user probably doesn't exists
            if(timeline == null || timeline.length == 0 )
            {
            return;
            }

            var twitStatus:TwitterStatus;
            for each( twitStatus in timeline )
            {
            _timelineDP.addItem( twitStatus );
            }

            _logedUserProfile = twitStatus.user;

            _userDetails = new TwitterUserVO();

            _userDetails.source = logedUserProfile.profileImageUrl;
            _userDetails.name = logedUserProfile.name;
            _userDetails.screenName = logedUserProfile.screenName;
            _userDetails.location = logedUserProfile.location;
            _userDetails.url = logedUserProfile.url;
            _userDetails.description = logedUserProfile.description;

            }
        }
    }
```

The class creates the instance of the `Twitter` class in the constructor and creates an event listener for the `TwitterEvent.ON_USER_TIMELINE_RESULT` event.

In the loadedUserTimeline event handler, called when a response user timeline is returned from web services, you check whether there is a timeline of messages for the specific screen name that was passed to the function; if a timeline doesn't exist, the function will exit. This check is carried out by verifying whether the timeline.length property is different from zero.

In the userDetails property, an ActionScript Value object is instanced and saved as TwitterUserVO.as, a simple class that transports data, to which you assign the properties contained in the TwitterStatus class and in its user: twitStatus.user property. This information belongs to the user for which you loaded the message timeline. The ActionScript Value object TwitterUserVO.as simply contains a list of properties:

```
package com.oreilly.aircookbook.ch16
{
    [Bindable]
    public class TwitterUserVO
    {

        public var source:String;
        public var name:String;
        public var location:String;
        public var url:String;
        public var description:String;
        public var screenName:String;

        public function TwitterUserVO()
        {
        }
        }
}
```

For the sake of brevity, in this Value object the properties have been declared as public. To follow object programming best practices, you should encapsulate the properties of a class to make it more secure and declare the properties as private so they can be exposed and be get and set with the public getter/setter methods.

The AIR application that defines the user interface elements is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
xmlns:mx="http://www.adobe.com/2006/mxml"
 width="550">

<mx:Script>
<![CDATA[
    import mx.core.IFlexDisplayObject;
    import com.oreilly.aircookbook.bonus.TwitterConn;
    import com.oreilly.aircookbook.bonus.TwitterAuthor;
    import mx.managers.PopUpManager;

    [Bindable]
    private var twclass:TwitterConn = new TwitterConn();
```

```
        private function showDetails():void
        {
           var myPopUp:IFlexDisplayObject = PopUpManager.createPopUp (this,
        TwitterAuthor, true);
            TwitterAuthor( myPopUp ).twitterDetails = twclass.userDetails;
            PopUpManager.centerPopUp( myPopUp );
        }

    ]]>
    </mx:Script>

    <mx:HBox>
    <mx:Label text="Insert your Twitter username: " />
    <mx:TextInput id="userTxt" />

    <mx:Button id="userBtn"
    label="Read latest message"
    click="twclass.loadUserTimeline(userTxt.text);authorBtn.visible = true" />

    </mx:HBox>

    <mx:List dataProvider="{twclass.timelineDP}"
    labelField="text"  width="496" height="166"/>

    <mx:LinkButton id="authorBtn" visible="false" label="See Twitter Account Details"
    click="showDetails()" />

    </mx:WindowedApplication>
```

The application has a `TextInput` control and a `Button` control that allow the users to insert their Twitter nicknames and download their timeline message lists. This list of messages is associated as a data provider to a `List` control through the `timelineDP` property of the ActionScript class that you created previously: `dataProvider= "{twclass.timelineDP}"`.

Finally, a `LinkButton` control allows you to open a pop-up window, controlled by the `PopUpManager` class, to which you pass the ActionScript `Value` object in the `userDetails` property:

```
private function showDetails():void
{
    var myPopUp:IFlexDisplayObject = PopUpManager.createPopUp (this, TwitterAuthor, true);
    TwitterAuthor( myPopUp ).twitterDetails = twclass.userDetails;
    PopUpManager.centerPopUp( myPopUp );
}
```

The second parameter that is passed to the `createPopUp` method of the `PopUpManager` class is an MXML component saved as `TwitterAuthor.mxml`. This component will be a subclass of the `TitleWindow` class, will take the `Value` object as a parameter, and will print its properties in the Flex controls:

```
<?xml version="1.0"  encoding="utf-8"?>
<mx:TitleWindow  xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"  title="Twitter Account Details"
```

```
width="320" height="240"
showCloseButton="true"
borderAlpha="1.0"
creationComplete="addEventListener( CloseEvent.CLOSE, closeMe );">

<mx:Script>
<![CDATA[
    import mx.managers.PopUpManager;
    import mx.events.CloseEvent;
    import com.oreilly.aircookbook.bonus.TwitterUserVO;

    [Bindable]
    public var twitterDetails:TwitterUserVO;
    private function closeMe( event:CloseEvent ):void
    {
        PopUpManager.removePopUp( this );
    }
]]>
</mx:Script>
<mx:Image source="{twitterDetails.source}"  x="10" y="12"/>
<mx:Label x="91" y="10" text="{twitterDetails.name}" />
<mx:Label x="89" y="36" text="{twitterDetails.screenName}" />
<mx:Label x="89" y="62" text="{twitterDetails.url}" />
<mx:Label x="89" y="88" text="{twitterDetails.location}" />
<mx:TextArea x="90" y="114" text="{twitterDetails.description}" />
</mx:TitleWindow>
```

JavaScript/HTML

For JavaScript, the TwitterJs library allows you to obtain a list of messages in your own web page. Created by remy sharp (*http://remysharp.com*), the TwitterJs library is available at *http://code.google.com/p/twitterjs/downloads/list*.

Download the zip file, extract it directly in the project folder, and import the file *twitterjs/src/twitter.js*.

The `getTwitters` method returns a list of messages. The parameters you can pass are as follows:

- `id` (String): This is the username.
- `count` (Int): This can be 1–20, and it defaults to 1. The maximum limit is 20.
- `prefix` (String): This is in the format `'%name% said'`. It defaults to blank.
- `clearContents` (Boolean): This removes the contents of the element specified in `cssIdOfContainer`; it defaults to `true`.
- `ignoreReplies` (Boolean): This skips over tweets starting with `@`, and it defaults to `false`.
- `template` (String): This is the HTML template to use for `li` elements. This defaults to a predefined template.
- `enableLinks` (Boolean): This “linkifies” the text, and it defaults to `true`.

- `timeout` (Int): This is how long before triggering `onTimeout`. It defaults to 10 seconds if `onTimeout` is set.
- `onTimeoutCancel` (Boolean): This completely cancels the Twitter call if it times out. It defaults to `false`.
- `onTimeout` (Function): This is the function to run when the timeout occurs. This function is bound to the element specified with it.

The example in AIR uses the same approach as the previous solution. One web page incorporates a second web page through an `iframe` container, which carries out the JavaScript call to the Twitter APIs.

This is the code of the parent web page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>

<script type="text/javascript" src="frameworks/AIRAliases.js" />
<script type="text/javascript" src="frameworks/AIRIntrospector.js" />


<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>AIR Cookbook: 18.3 Consuming Twitter APIs (JavaScript)</title>
</head>

<body>
<h1>AIR Cookbook: 18.3 Consuming Twitter APIs (JavaScript)</h1>


<iframe
id="twitter"
src="tweetrIframe.html"
sandboxRoot="http://SomeRemoteDomain.com/"
documentRoot="app:/" width="500" height="500"
/>

</body>
</html>
```

The core of the application is in the *tweetrIframe.html* page that is loaded in the `iframe`. In this web page, after importing the TwitterJs library, call the `getTwitters` method, which will return the message timeline.

This is the complete code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en">
<head>
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
 <title>Twitter Plugin</title>
<link rel="stylesheet" href="twitterjs/test/twitter.css" type="text/css" media="screen"
```

```
charset="utf-8" />

<script src="twitterjs/src/twitter.js" type="text/javascript" charset="utf-8"></script>
<script type="text/javascript" charset="utf-8">

var twitter_id = '765942';

function loadMsg()
{

var nick = document.getElementById( 'location' ).value;

if (nick == 'Insert your Twitter nickname' || nick == '') { nick = 'marcocasario'}

getTwitters('msg', {
id: nick,
prefix: '<img height="16" width="16" src="%profile_image_url%" />
<a href="http://twitter.com/%screen_name%">%name%</a> said: ',
clearContents: false, // leave the original message in place
count: 10,
withFriends: true,
ignoreReplies: false,
template: '<span class="twitterPrefix"><img height="16" width="16"
src="%user_profile_image_url%" />
<span class="twitterStatus">"%text%"</span> <em class="twitterTime">
<a href="http://twitter.com/%user_screen_name%/statuses/%id%">%time%</a></em>'
    });
}
</script>
  </head>
  <body id="page">

 <input id="location" type="text" value="Insert your Twitter nickname" />

<input type="submit" name="go" id="go" value="Show Tweets" onclick="loadMsg()" />

<div class="twitters" id="msg">
<p></p>
</div>

</body>
</html>
```

The page displays a text input where users can insert their screen names used in Twitter and a button that, once it is clicked, invokes the `loadMsg` function. This function takes the value in the text input and, in turn, launches the `getTwitters` method from the TwitterJs library. This method, according to the parameters that are specified for it, returns the message timeline for the Twitter user who is specified in the text input. If the text input is left empty, the default nickname is marcocasario (my personal nickname).