Boutros AbiChedid, ***BEE, MSHP, MSEE***
boutrosabichedid@yahoo.com

This document tracks the time I spent to develop the Picture Gallery Flash movie. Flash CS3 authoring tool was used, along with Action Script 3.0, XML, XHTML, CSS, and JavaScript. Photoshop CS3 was also used to enhance the quality of the pictures and to resize them. Also, two open source code from Google were used (described below).

1. swfobject http://code.google.com/p/swfobject/

SWFObject is an easy-to-use and standards-friendly method to embed Flash content, which utilizes one small JavaScript file

documentation

Embedding Adobe Flash Player content using SWFObject 2

What is SWFObject?

SWFObject 2:

- Offers two optimized Flash Player embed methods; a markup based approach and a method that relies on JavaScript
- Offers a JavaScript API that aims to provide a complete tool set for embedding SWF files and retrieving Flash Player related information
- Utilizes only one small JavaScript file (10Kb / GZIPed: 3.9Kb)
- Is the successor of SWFObject 1.5, UFO and the Adobe Flash Player Detection Kit
- Intends to unify all existing Flash Player embed methods and provide a new standard for embedding Adobe Flash Player content

Why should you use SWFObject?

SWFObject 2:

- Is more optimized and flexible than any other Flash Player embed method around
- Offers one solution for everybody: It shouldn't matter if you are an HTML, Flash, or JavaScript developer, there should be something in it for everyone
- Breaks the cycle of being locked into vendor specific markup and promotes the use of web standards and alternative content
- Uses unobtrusive JavaScript and JavaScript best practices
- Is easy to use

The A List Apart article Flash Embedding Cage Match describes the full rationale behind SWFObject 2.

Why does SWFObject use JavaScript?

SWFObject 2 primarily uses JavaScript to overcome issues that cannot be solved by markup alone; it:

- Detects the Flash Player version and determines whether Flash content or alternative content should be shown, to avoid that outdated Flash plug-ins break Flash content
- Offers functionality to revert to alternative content in case of an outdated plug-in by means of a DOM manipulation (Note: if no Flash plug-in is installed the HTML `object` element automatically falls back to its nested alternative content)
- Offers the option to use Adobe Express Install to download the latest Flash Player

- Offers a JavaScript API to perform common Flash Player and Flash content related tasks

Should I use the static or dynamic publishing method?

SWFObject 2 offers two distinct methods to embed Flash Player content:

1. The **static publishing method** embeds both Flash content and alternative content using standards compliant markup, and uses JavaScript to resolve the issues that markup alone cannot solve
2. The **dynamic publishing method** is based on marked up alternative content and uses JavaScript to replace this content with Flash content if the minimal Flash Player version is installed and enough JavaScript support is available (similar like previous versions of SWFObject and UFO)

The advantages of the **static publishing method** are:

1. The actual authoring of standards compliant markup is promoted
2. Best embed performance
3. The mechanism of embedding Flash content does not rely on a scripting language, so your Flash content can reach a significant bigger audience:
    - If you have the Flash plug-in installed, but have JavaScript disabled or a use a browser that doesn't support JavaScript, you will still be able to see your Flash content
    - Flash will now also run on a device like Sony PSP, which has very poor JavaScript support
    - Automated tools like RSS readers are able to pick up Flash content

The advantages of the **dynamic publishing method** are:

1. It integrates very well with scripted applications and enables the use of dynamic variables (flashvars)
2. It avoids *click-to-activate mechanisms* to activate *active content* in Internet Explorer 6/7 and Opera 9+. Please note that Microsoft has phased out most active content from its Internet Explorer browsers

How to embed Flash Player content using SWFObject *static publishing*

STEP 1: Embed both Flash content and alternative content using standards compliant markup

SWFObject's base markup uses the nested-objects method (with proprietary Internet Explorer conditional comments) http://www.alistapart.com/articles/flashembedcagematch/ to ensure the most optimal cross-browser support by means of markup only, while being standards compliant and supporting alternative content http://www.bobbyvandersluis.com/flashembed/testsuite/ :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>SWFObject - step 1</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  </head>
  <body>
    <div>

      <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780" height="420">
        <param name="movie" value="myContent.swf" />
        <!--[if !IE]>-->
        <object type="application/x-shockwave-flash" data="myContent.swf" width="780" height="420">
        <!--<![endif]-->
          <p>Alternative content</p>
        <!--[if !IE]>-->
        </object>
        <!--<![endif]-->
      </object>

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

NOTE: The nested-objects method requires a double `object` definition (the outer `object` targeting Internet Explorer and the inner `object` targeting all other browsers), so you need to define your `object` attributes and nested `param` elements twice.

Required attributes:

- `classid` (outer `object` element only, value is always `clsid:D27CDB6E-AE6D-11cf-96B8-444553540000`)
- `type` (inner `object` element only, value is always `application/x-shockwave-flash`)
- `data` (inner `object` element only, defines the URL of a SWF)
- `width` (both `object` elements, defines the width of a SWF)
- `height` (both `object` elements, defines the height of a SWF)

Required `param` element:

- `movie` (outer `object` element only, defines the URL of a SWF)

NOTE: We advise not to use the `codebase` attribute to point to the URL of the Flash plugin installer on Adobe's servers, because this is illegal according to the specifications which restrict its access to the domain of the current document only. We recommend the use of alternative content with a subtle message that a user can have a richer experience by downloading the Flash plugin instead.

How can you use HTML to configure your Flash content?

You can add the following often-used optional attributes to the `object` element:

- `id`
- `name`
- `class`
- `align`

You can use the following optional Flash specific `param` elements (more info):

- `play`
- `loop`
- `menu`
- `quality`
- `scale`
- `salign`
- `wmode`
- `bgcolor`
- `base`
- `swliveconnect`
- `flashvars`
- `devicefont` (more info)
- `allowscriptaccess` (more info here and here)

- `seamlesstabbing` (more info)
- `allowfullscreen` (more info)
- `allownetworking` (more info)

Why should you use alternative content?

The `object` element allows you to nest alternative content inside of it, which will be displayed if Flash is not installed or supported. This content will also be picked up by search engines, making it a great tool for creating search-engine-friendly content. Summarizing, you should use alternative content when you like to create content that is accessible for people who browse the Web without plugins, create search-engine-friendly content or tell visitors that they can have a richer user experience by downloading the Flash plug-in.

STEP 2: Include the SWFObject JavaScript library in the head of your HTML page

The SWFObject library consists of one external JavaScript file. SWFObject will be executed as soon as it is read and will perform all DOM manipulations as soon as the DOM is loaded - for all browsers that support this, like IE, Firefox, Safari and Opera 9+ - or otherwise as soon as the onload event fires:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>SWFObject - step 2</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

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

  </head>
  <body>
    <div>
      <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780" height="420">
        <param name="movie" value="myContent.swf" />
        <!--[if !IE]>-->
        <object type="application/x-shockwave-flash" data="myContent.swf" width="780" height="420">
        <!--<![endif]-->
          <p>Alternative content</p>
        <!--[if !IE]>-->
        </object>
        <!--<![endif]-->
      </object>
    </div>
  </body>
</html>
```

STEP 3: Register your Flash content with the SWFObject library and tell SWFObject what to do with it

First add a unique `id` to the outer `object` tag that defines your Flash content. Second add the `swfobject.registerObject` method:

1. The first argument (String, required) specifies the `id` used in the markup.
2. The second argument (String, required) specifies the Flash player version your content is published for. It activates the Flash version detection for a SWF to determine whether to show Flash content or force alternative content by doing a DOM manipulation. While Flash version numbers normally consist of major.minor.release.build, SWFObject only looks at the first 3 numbers, so both "WIN 9,0,18,0" (IE) or "Shockwave Flash 9 r18" (all other browsers) will translate to "9.0.18". If you only want to test for a major version you can omit the minor and release numbers, like "9" instead of "9.0.0".
3. The third argument (String, optional) can be used to activate Adobe express install and specifies the URL of your express install SWF file. Express install displays a standardized Flash plugin download dialog instead of your Flash content when the required plugin version is not available. A default expressInstall.swf file is packaged with the project. It also contains the corresponding expressInstall.fla and AS files (in the SRC directory) to let you create your own custom express install experience. Please note that express install will only fire once (the first time that it is invoked), that it is only supported by Flash Player 6.0.65 or higher on Win or Mac platforms, and that it requires a minimal SWF size of 310x137px.
4. The fourth argument (JavaScript function, optional) can be used to define a callback function that is called on both success or failure of embedding a SWF file (see API documentation)

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

```
    <title>SWFObject - step 3</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <script type="text/javascript" src="swfobject.js"></script>

    <script type="text/javascript">
    swfobject.registerObject("myId", "9.0.115", "expressInstall.swf");
    </script>

  </head>
  <body>
    <div>

      <object id="myId" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="780" height="420">

        <param name="movie" value="myContent.swf" />
        <!--[if !IE]>-->
        <object type="application/x-shockwave-flash" data="myContent.swf" width="780" height="420">
        <!--<![endif]-->
          <p>Alternative content</p>
        <!--[if !IE]>-->
        </object>
        <!--<![endif]-->
      </object>
    </div>
  </body>
</html>
```

TIPS

- Use the SWFObject HTML and JavaScript generator to help you author your code
- Just repeat steps 1 and 3 to embed multiple SWF files into one HTML page
- The easiest way to reference the active `object` element is by using the JavaScript API: `swfobject.getObjectById(objectIdStr)

How to embed Flash Player content using SWFObject *dynamic publishing*

STEP 1: Create alternative content using standards compliant markup

SWFObject's dynamic embed method follows the principle of progressive enhancement and replaces alternative HTML content for Flash content when enough JavaScript and Flash plug-in support is available. First define your alternative content and label it with an `id`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>SWFObject dynamic embed - step 1</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  </head>
  <body>

    <div id="myContent">
      <p>Alternative content</p>
    </div>

  </body>
</html>
```

STEP 2: Include the SWFObject JavaScript library in the head of your HTML page

The SWFObject library consists of one external JavaScript file. SWFObject will be executed as soon as it is read and will perform all DOM manipulations as soon as the DOM is loaded - for all browsers that support this, like IE, Firefox, Safari and Opera 9+ - or otherwise as soon as the onload event fires:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>SWFObject dynamic embed - step 2</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

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

```
  </head>
  <body>
    <div id="myContent">
      <p>Alternative content</p>
    </div>
  </body>
</html>
```

STEP 3: Embed your SWF with JavaScript

`swfobject.embedSWF(swfUrl, id, width, height, version, expressInstallSwfurl, flashvars, params, attributes, callbackFn)` has five required and five optional arguments:

1. swfUrl (String, required) specifies the URL of your SWF
2. id (String, required) specifies the `id` of the HTML element (containing your alternative content) you would like to have replaced by your Flash content
3. width (String, required) specifies the width of your SWF
4. height (String, required) specifies the height of your SWF
5. version (String, required) specifies the Flash player version your SWF is published for (format is: "major.minor.release" or "major")
6. expressInstallSwfurl (String, optional) specifies the URL of your express install SWF and activates Adobe express install. Please note that express install will only fire once (the first time that it is invoked), that it is only supported by Flash Player 6.0.65 or higher on Win or Mac platforms, and that it requires a minimal SWF size of 310x137px.
7. flashvars (Object, optional) specifies your flashvars with name:value pairs
8. params (Object, optional) specifies your nested `object` element `params` with name:value pairs
9. attributes (Object, optional) specifies your `object`'s attributes with name:value pairs
10. callbackFn (JavaScript function, optional) can be used to define a callback function that is called on both success or failure of embedding a SWF file (see API documentation)

NOTE: You can omit the optional parameters, as long as you don't break the parameter order. If you don't want to use an optional parameter, but would like to use a following optional parameter, you can simply pass `false` as its value. For the `flashvars`, `params` and `attributes` JavaScript Objects, you can also pass an empty object instead: `{}`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>SWFObject dynamic embed - step 3</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <script type="text/javascript" src="swfobject.js"></script>

    <script type="text/javascript">
    swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0");
    </script>

  </head>
  <body>
    <div id="myContent">
      <p>Alternative content</p>
    </div>
  </body>
</html>
```

How can you configure your Flash content?

You can add the following often-used optional attributes to the `object` element:

- `id` (NOTE: when undefined, the `object` element automatically inherits the `id` from the alternative content container element)
- `name`

- `styleclass` (used instead of `class`, because this is also an ECMA4 reserved keyword)
- `align`

You can use the following optional Flash specific `param` elements (more info):

- `play`
- `loop`
- `menu`
- `quality`
- `scale`
- `salign`
- `wmode`
- `bgcolor`
- `base`
- `swliveconnect`
- `flashvars`
- `devicefont` (more info)
- `allowscriptaccess` (more info here and here)
- `seamlesstabbing` (more info)
- `allowfullscreen` (more info)
- `allownetworking` (more info)

How do you use JavaScript Objects to define your flashvars, params and object's attributes?

You can best define JavaScript Objects using the Object literal notation, which looks like:

```
<script type="text/javascript">

var flashvars = {};
var params = {};
var attributes = {};

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf", flashvars, params,
attributes);

</script>
```

You can add your name:value pairs while you define an object (note: please make sure not to put a comma behind the last name:value pair inside an object)

```
<script type="text/javascript">

var flashvars = {
  name1: "hello",
  name2: "world",
  name3: "foobar"
};
var params = {
  menu: "false"
```

```
};
var attributes = {
  id: "myDynamicContent",
  name: "myDynamicContent"
};

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf", flashvars, params,
attributes);

</script>
```

Or add properties and values after its creation by using a dot notation:

```
<script type="text/javascript">

var flashvars = {};
flashvars.name1 = "hello";
flashvars.name2 = "world";
flashvars.name3 = "foobar";

var params = {};
params.menu = "false";

var attributes = {};
attributes.id = "myDynamicContent";
attributes.name = "myDynamicContent";

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf", flashvars, params,
attributes);

</script>
```

Which can also be written as (the less readable shorthand version for the die-hard scripter who love one-liners):

```
<script type="text/javascript">

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf",
{name1:"hello",name2:"world",name3:"foobar"}, {menu:"false"}, {id:"myDynamicContent",name:"myDynamicContent"});

</script>
```

If you don't want to use an optional argument you can define it as `false` or as an empty Object (NOTE: from **SWFObject 2.1** onwards you can also use `null` or `0`):

```
<script type="text/javascript">

var flashvars = false;
var params = {};
var attributes = {
  id: "myDynamicContent",
  name: "myDynamicContent"
};

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf", flashvars, params,
attributes);

</script>
```

The flashvars Object is a shorthand notation that is there for your ease of use. You could potentially ignore it and specify your flashvars via the params Object:

```
<script type="text/javascript">

var flashvars = false;
var params = {
  menu: "false",
  flashvars: "name1=hello&name2=world&name3=foobar"
};
var attributes = {
  id: "myDynamicContent",
  name: "myDynamicContent"
```

```
};

swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0","expressInstall.swf", flashvars, params,
attributes);

</script>
```

TIPS

- Use the SWFObject HTML and JavaScript generator to help you author your code
- Just repeat steps 1 and 3 to embed multiple SWF files into one HTML page

SWFObject 1.5 to SWFObject 2 migration tips

1. SWFObject 2 is NOT backwards compatible with SWFObject 1.5
2. It is now preferred to insert all script blocks in the head of your HTML page. Adding your scripts in the body of your page may have visual impact (e.g. flashes of replaced alternative content), because your JavaScript code will be executed in a later stage (the exact impact depends on your implementation)
3. The library itself is now written in lowercase: `swfobject` instead of `SWFObject`
4. Methods can only be accessed via the library (instead via a SWFObject instance in SWFObject 1.5)
5. The JavaScript API is entirely different and more elaborate
6. SWFObject 2 replaces your entire alternative HTML content block, including the referenced HTML container element, for Flash content when enough JavaScript and Flash support is available, while SWFObject 1.5 only replaces the content inside the referenced HTML container. When you don't specify an `id` attribute, the `object` element will automatically inherit the `id` of the HTML container element of your alternative content.

UFO to SWFObject 2 migration tips

1. SWFObject 2 replaces your entire alternative HTML content block, including the referenced HTML container element, for Flash content when enough JavaScript and Flash support is available, while UFO only replaces the content inside the referenced HTML container. When you don't specify an `id` attribute, the `object` element will automatically inherit the `id` of the HTML container element of your alternative content.
2. UFO's `setcontainercss` feature has not been incorporated in SWFObject 2, however it can easily be replicated by using the SWFObject JavaScript API, see: `swfobject.createCSS(selStr, declStr)`

Does SWFObject 2 support MIME type application/xhtml+xml?

SWFObject 2 does NOT support XML MIME types, which is a design decision.

There are a number of reasons why we are not supporting this:

- Only a very small (non-significant) amount of web authors is using it
- We are unsure if it is the direction to go. Internet Explorer does not support it and all other major browser vendors are aiming their arrows at a new standard way of HTML parsing (with HTML 5), which departs from the current W3C vision of parsing HTML as XML
- We save a considerate amount of file size and effort (testing, issue resolving) by not supporting it

2. **tweener** http://code.google.com/p/tweener/

A class for creating tweens in actionscript 2 and 3 - because there's infinity between 0 and 1.

Tweener (caurina.transitions.Tweener) is a Class used to create tweenings and other transitions via ActionScript code for projects built on the Flash platform. It's released and maintained for these versions:

- ActionScript 2.0, for Flash 7+ and Flash Lite 2.0+
- ActionScript 2.0, for Flash 8+
- ActionSctipt 3.0, for Flash 9+

Ported/inspired versions for other languages are also available:

- JavaScript version (ported by Yuichi Tateno)
- C++ version (ported by Wesley Marques)

In layman's terms, Tweener helps you move things around on the screen using only code, instead of the timeline.

The general idea of a tweening Class is that dynamic animation and transitions (created by code) are easier to maintain and control, and more stable than animation based on the regular Flash timeline, since you can control it by time rather than by frames.

Aimed both for designers and advanced developers, the Tweener syntax is created with **simplicity** of use in mind, while still allowing access to more advanced features. Because of this, it follows a 'one-line' design mentality when creating new tweenings, with no instancing required (as it's a static Class) and a set of optional parameters. Also, there are no initialization methods required by Tweener, other than the mandatory 'import' command.

Its fluid syntax allows it to be used to tween any numeric property of any object of any class, so it is not tied to specific properties of built-in Classes such as MovieClips or TextFields. This **flexibility** grants a wider control on how transitions are performed, and makes creating complex sequential transitions on any kind of object easier.

Small file overhead is also one of the main goals of Tweener - once included on SWF movies, Tweener currently takes 8.8kb (AS2 FL2), 9.2kb (AS2) or 10.4kb (AS3) of the total compiled file size. It can be compiled with the Flash IDE, MTASC, or Flex SDK (even with strict rules on), with no errors or warnings thrown during compilation.

Tweener is also the spiritual successor to MC Tween. However, it follows ActionScript's more strict OOP rules, and gets rid of the fixed parameter order syntax imposed by MC Tween. As a result, code written with Tweener is a lot more readable even for developers not versed on the Class.

Development wise, **modularity** is one of the main aspects of Tweener. The code is built in a way that new features such as transitions and special tweenings can be added (or removed) easily: for example, properties that are only acessible through methods and functions can be tweened by creating and registering new special properties. Expanding the feature set of the original Class can be done on a per-project basis, with no change to the original files.

From this page, you can download the latest stable (heavily tested) version of Tweener, check out a few examples with source, or read the documentation. There's also a mailing list for Tweener discussion.

Tweener Documentation http://hosted.zeh.com.br/tweener/docs/en-us/

Introduction

Tweener is a *class* for Actionscript 2 and 3, meaning it's a piece of code you can call and reuse on your own movies. This is just its documentation - for more information on the extension itself, please visit the official website at Google Code.

For a quick explanation on how to install Tweener, see Installation first.

If you are used to Actioncript 1, prototypes, or #includes, and don't know how to use a real class yet, please read the how to use a class page for a brief introduction on the issue.

If you are used to MC Tween and want to know the differences on the syntax between MC Tween and Tweener, also check Differences between MC Tween and Tweener.

Brief explanation

Tweener is a *static* class - that is, a class that allows you to run methods on it, or call its properties, but that never lets you create instances from it. This means that, with Tweener, you never create a new object - you simply tells Tweener to do something for you.

Tweener works on the idea that, instead of setting the value of a given property of a given object directly, as in `myMC._x = 10`, you can tell that property to create a *transition* to that value - by doing this transition or tweening, you can control your numeric data in a more fluid way, also by using *easing equations* in this process. Doing slides, fades, and all kinds of animation is the result of this kind of manipulation: by making a property or variable go to one value or the other fluidly, not immediately. Rich transitions and animations with simple code is the aim of Tweener.

With Tweener, you write your code by adding new tweenings or transitions to the movie, using the method addTween. Like this (AS2 version):

```
Tweener.addTween(myMovie, {_x:10, _y:10, time:1, transition:"linear"});
```

Or this (AS3 version):

```
Tweener.addTween(myMovie, {x:10, y:10, time:1, transition:"linear"});
```

This will move the object `myMovie` on screen, going to the position 10,10, in 1 second, using a linear animation (other transition types are available).

The final value is an absolute value, not a relative one. This means that, if you want to, say, move `myMovie` 10 pixels to the right (instead of sliding it to column 10), you would do this (in AS2):

```
Tweener.addTween(myMovie, {_x:myMovie._x+10, time:1, transition:"linear"});
```

Also notice you can chain tweenings sequentially, creating complex animations, by using delays. For example, to move `myMovie` to column 10, then to column -10, you would do (in AS2):

```
Tweener.addTween(myMovie, {_x:10, time:1, transition:"linear"});
Tweener.addTween(myMovie, {_x:-10, time:1, delay:1, transition:"linear"});
```

Notice the use of the delay property, that tells the animation to wait a bit (the same ammount of the previous transition) before starting. This specified delay property allows you to create complex animations that are not necessarily chained together sequentially, but start at a given time; you could create animations that act on different objects with arbritrary ammounts of time to wait for any of these transitions.

The cool thing with Tweener is that this addTween method allows you to use a plethora of different optional tweening parameters, giving you control to a lot of different transition properties.

The important thing to notice is that Tweener isn't just a class used to create just animations. It's, instead, a class used to create transitions - doing tweenings **on any numeric property of any object** - which can be used for many purposes, including visual animation. It's not restricted to the normal Actionscript classes like `MovieClip` or `TextField` either; you can use it to tween properties of your own classes and objects, like moving the `cameraX` and `cameraY` properties of your game rendering engine, the `currentTemperature` of your own `Thermometer` class object, the `travelPosition` of your train simulation movie, the `currentStrength` of each of your spectrum analyzer graph bars, and so on and so forth.

See the menu at the left for a listing of the available methods and other options. Don't forget to also check the online examples for a bunch of practical uses of Tweener, including their source files.

****

Total number of hours I spent on Picture gallery (details below) = 36.75 ***hrs***

****

****

Date	Time Spent (hrs)	Description (mainly)
Wednesday December 16, 2009	1.0	**Working o**n a personal Picture Gallery. Coded in Flash CS3/ ActionScript 3.0, XML and JavaScript. Photoshop CS3 was extensively

		used to enhance the quality of the images and also resize them.
Thursday December 17, 2009	6.75	**Continue-**working **o**n a personal Picture Gallery.
Friday December 18, 2009	3.0	**Continue-**working **o**n a personal Picture Gallery.
Saturday December 19, 2009	4.0	**Continue-**working **o**n a personal Picture Gallery.
Sunday December 20, 2009	6.0	**Continue-**working **o**n a personal Picture Gallery.
Monday December 21, 2009	5.75	**Continue-**working **o**n a personal Picture Gallery. Added a password protection code for the webpage.
Tuesday December 22, 2009	1.75	**Continue-**working **o**n a personal Picture Gallery. Wrote a Security Risks document about Private content for online presence.
Wednesday December 23, 2009	2.0	**Continue-**working **o**n a personal Picture Gallery. Made a Private and a Public Photo Gallery.
Thursday December 24, 2009	3.25	**Continue-**working **o**n the Picture Gallery.
Thursday December 24, 2009	3.0	**Finished-**working **o**n the Picture Gallery.
	Total = 36.75 hrs	