Learn How to Create a Portfolio Website Using Adobe Flash

To take a quick peek at what you'll be creating, go to www.jwright.info | Download Source Files

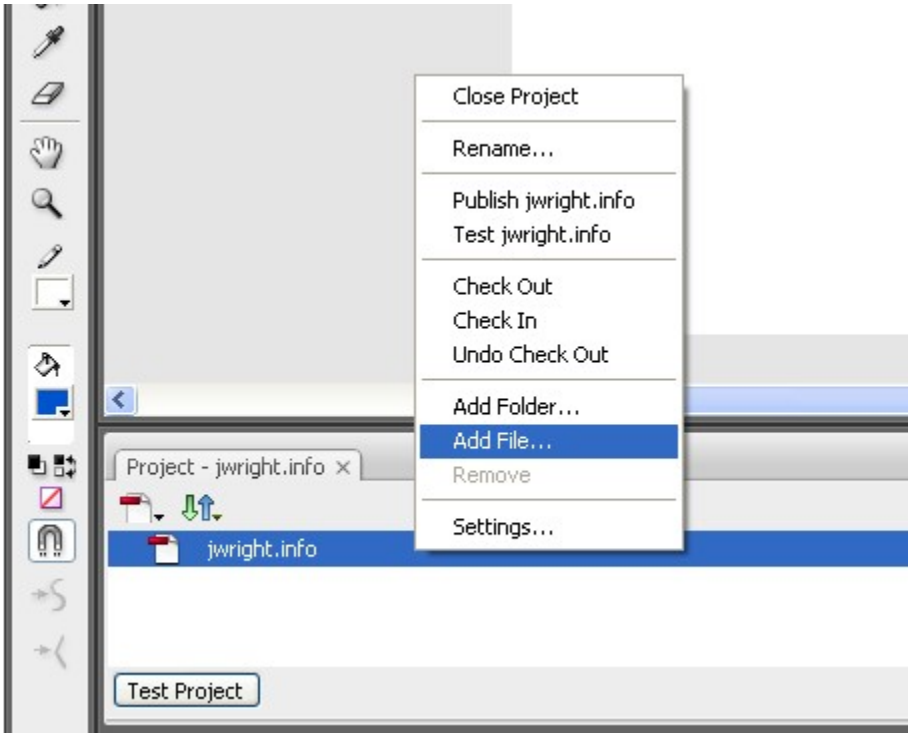
Getting Started: Setting up our Project

Open up Adobe Flash CS3 and create a new Flash Project. *File–>New–>Flash Project*.
Click *Ok*. Name your project *jwright.info.flp*.

Tip: I like to create a folder as one central location for all of my files (e.g.
a folder named jwright.info) to keep everything together.

Next, select *File–>New–>Flash File (ActionScript 3.0)*. Click *Ok*.
It's a good idea to go ahead and save your project. Let's do so. *File–>Save* (or Ctrl+S). Name the new Flash file *jwright.info.fla*.

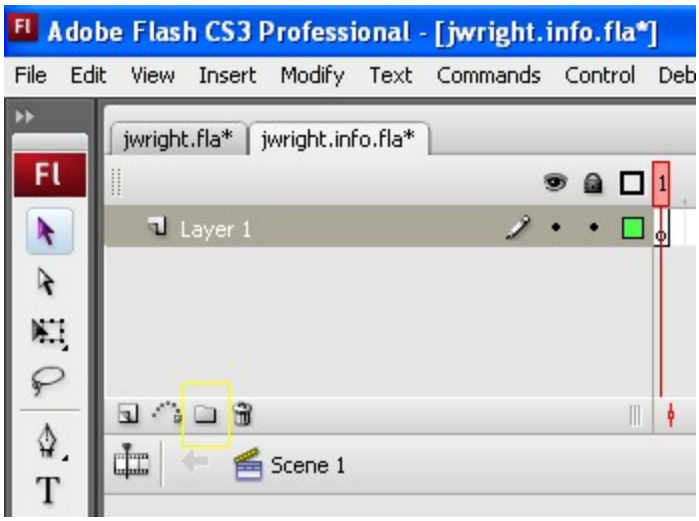
Now we can add our Flash file to our project. Right click on the project file and click *Add File…* as shown below. If you do not see you Project Panel, open it up by selecting *Window–>Project* (or Shift+F8) from the main toolbar.



Add the file *jwright.info.fla* to the project.

Setting up the Timeline

Let's start by setting up the layers that make up our site. Click the *Insert Layer Folder* button for the timeline. Rename this folder *Pages* by right clicking the folder and choosing *Properties* or by double clicking the folder.



Follow the similar procedure for adding a new folder for each page we will need: *Start, Home, Profile, Resume, Contact, Portfolio,* and *Respect*. Also, create a folder called *Master*.

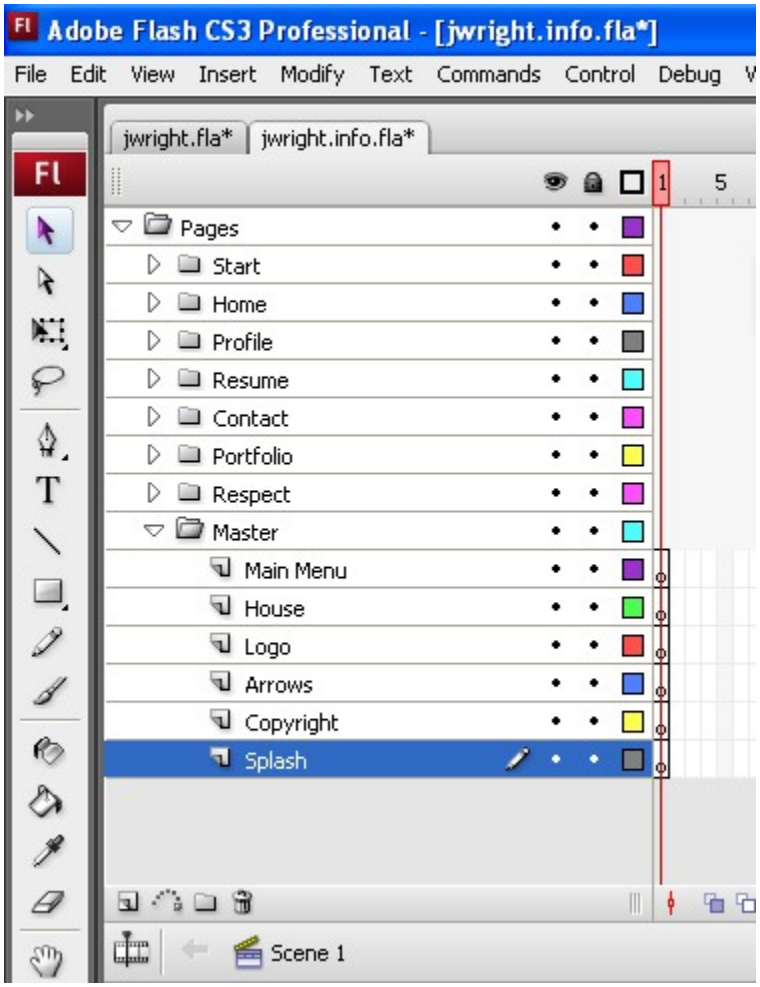
Here's a look at what you should now have in your timeline. Ensure that each page folder is a subfolder of *Pages*. You can do this by dragging the layer into the proper place. (You may delete the default layer *Layer1*, if it's still listed.)

Oh boy, we're rockin' now! Let's insert a new layer in the *Master* folder. Click the *Insert Layer* button found near the the *Inser Layer Folder* button.

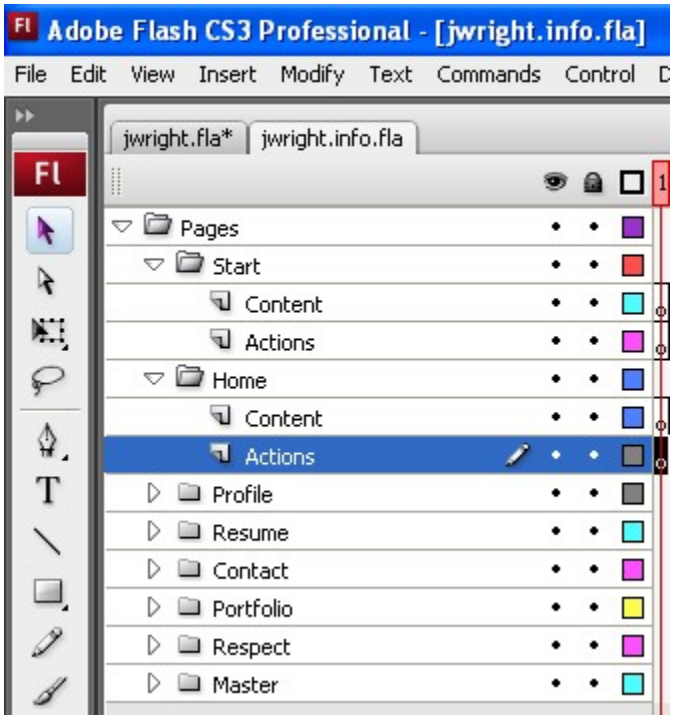
Name the new layer *Splash*. Again, be sure that the *Splash* layer is inside of the *Master* folder.

Follow the same steps to add the rest of our *Master* layers: *Main Menu, House, Logo, Arrows, Copyright,* and of course *Splash*.

Here's a stunning view of our timeline so far.

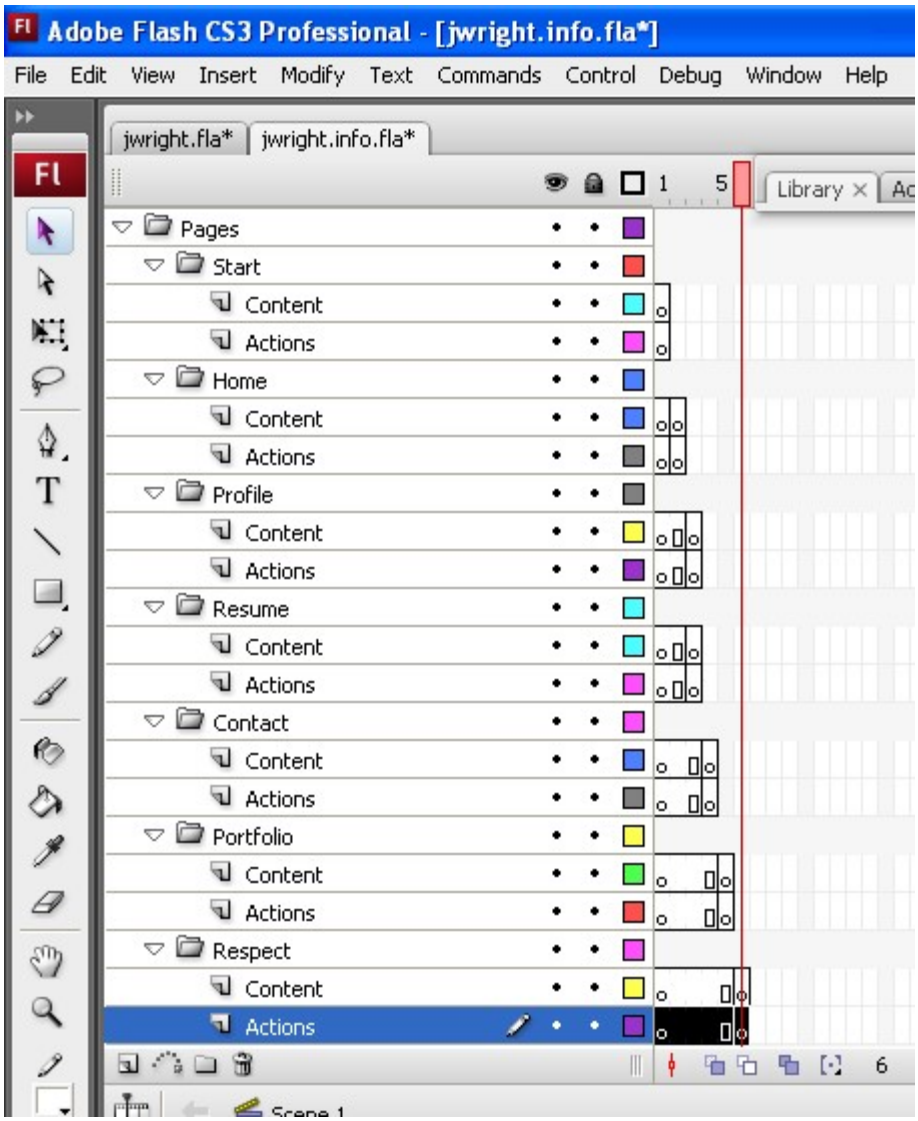


Okay, now in each page folder (excluding *Master*) insert two layers named *Content* and *Actions* like so.



Now, we want to add the keyframes for our website. Essentially, each page has a keyframe used to display the page we need. Then we can navigate to each requested page via frames.

Insert a keyframe on each page layer in sequence. Example: The *Start* page's keyframe goes on frame 1; The *Home* page's keyframe goes on frame 2; etc. Like so. To insert a keyframe, right click on the desired frame and choose *Insert Keyframe* (or select the frame and hit F6).



A couple more things before the real fun begins.

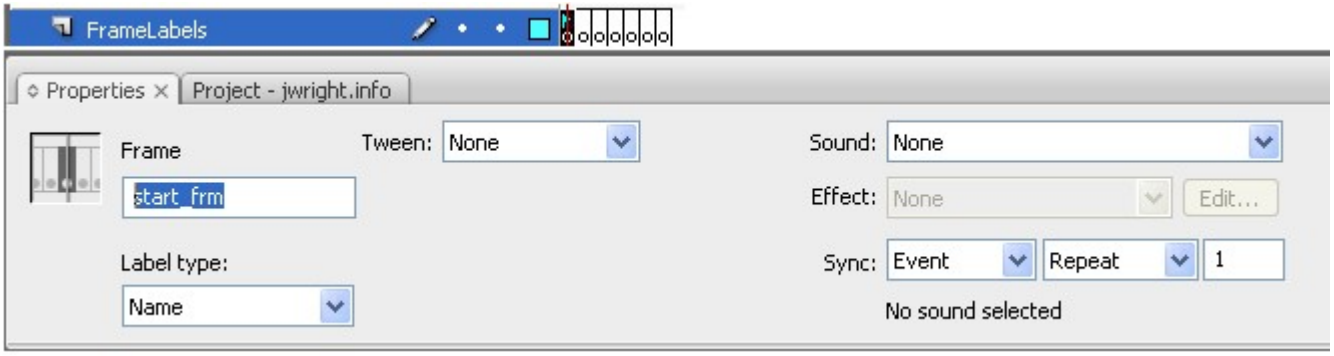
Create two more layers (not in any folder) named *Frame Labels* and *Actions*.

We'll place a lot of our ActionScript for the site in the *Actions* layer later on. We're going to use the *Frame Labels* layers for our site navigation. So, let's setup our Frame Labels.

Actually, let's first define a Frame Label. Simple, simple. A Frame Label is really just a way we can use an identifier for our page instead of always accessing a page by its frame number. It'll make more sense in a minute–bear with me.

Select the first seven frames on the *Frame Labels* layer. Then, convert them to keyframes (F6). Here's a look.

We need to give these Frame Labels a name. Select the first frame (frame 1) and give it a name in the *Properties* panel. Let's call it *start_frm*. This naming convention is consistent with our page layer names.

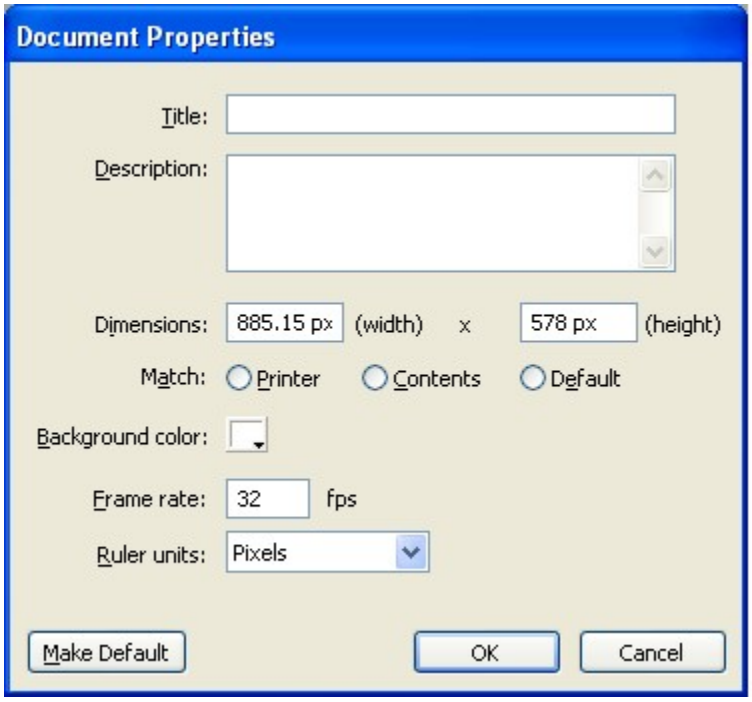


You should now see little flag in the first frame. Repeat these steps to name the rest of our Frame Labels. Here's a look at what you should have when all are named.

Well done ladies and gentlemen. Moving on…

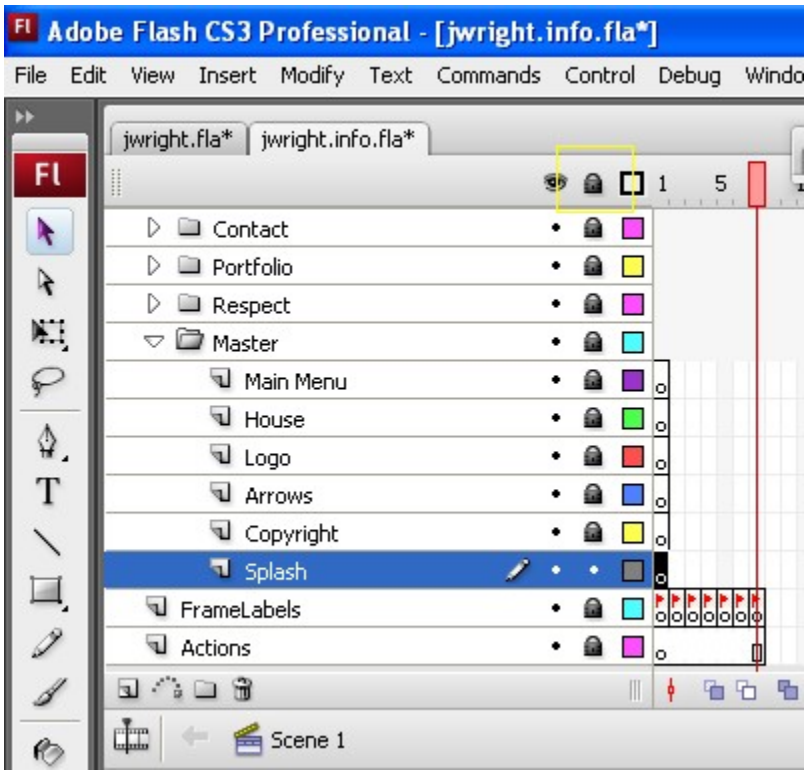
Creating the Main Layout

On the Properties panel, *Window–>Properties–>Properties* (or Ctrl+F3), click the size button. This will display the Document Properties form.



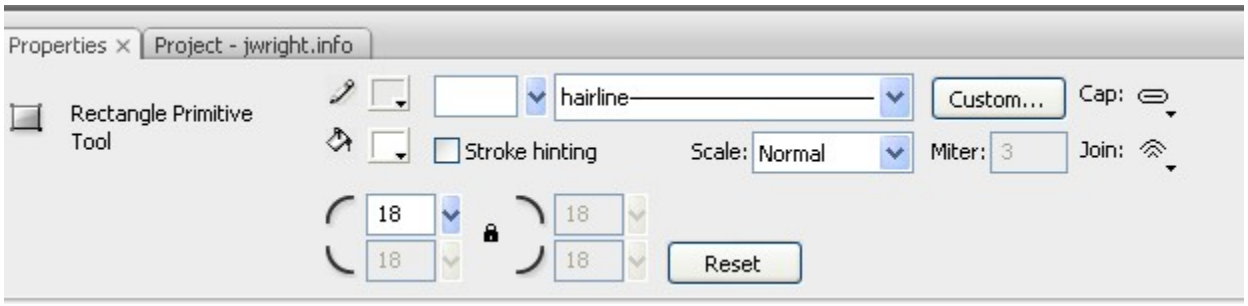
Set the *Dimensions* width to 885.15 px and the height to 578 px. Set the *Background color* to White (#FFFFFF), the *Frame rate* to 32 fps (frames per second), and the *Ruler units* to Pixels.

Ready for our first graphic? Me too. First, let's lock all of the layers but the one we need– the *Splash* layer.



Secondly, from the Tools toolbar, *Window–>Tools* (or Ctrl+F2), select the *Rectangle Primitive Tool* (or R). Change the stroke color to #EBEBEB, and
set the fill color to #FFFFFF via the color pickers on the Properties panel. Set
the corners rounded to 18.

Here's a shot of the properties panel for some help.

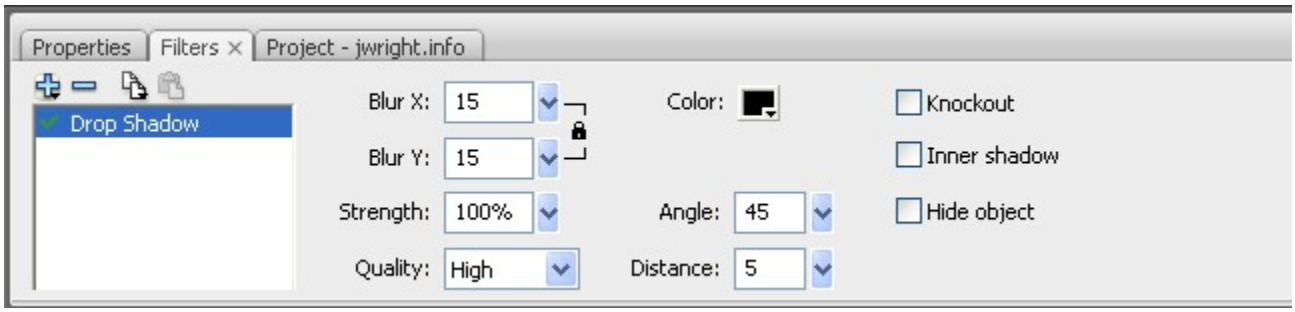


Now draw a primi rectangle 787.1 (width) by 468.4 (height) on the stage. Note: Ensure the *Splash* layer is selected.

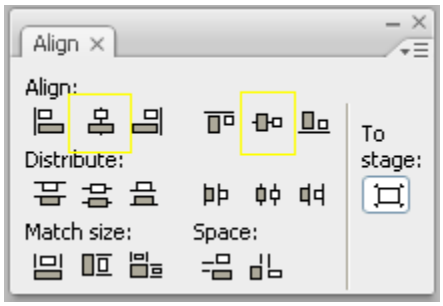
It's Movie Clip time. Select your primi and hit F8. This will pop up the Convert to Symbol form. Name it *SplashMc* and select Movie clip as the type. Click *Ok*.

This Mc looks pretty sad. Let's spice it up! Select *Window–>Properties–>Filter*. Select our *SplashMc* and hit the *Add Filter* button on the Filters panel and select *Drop Shadow*.

Next set the drop shadow's properties shown below.



Let's align this bad boy! Open up the align panel *Window–>Align* (or Ctrl+K). Select your Mc and click *Align horizontal center* and then *Align vertical center*.



Now that it's centered beautifully, let's move it down a little. About six hits on the down arrow on your keyboard. My X,Y coords for this Mc is now 442.6,307.2;

however, I've seen these X,Y's vary a little from computer to computer. Not sure why, but that's life I guess.

The copyright. First, lock the *Splash* layer and unlock the *Copyright* layer. Create a new Text Box using the text tool (T). Type in *Copyright © 2007-2008 jwright.info. All Rights Reserved.* Note: You can find the copyright symbol on your Windows start menu. *Start–>All Programs–>Accessories–>System Tools–>Character Map.* (That is, if you have a PC with Windows.)

Again, use the Align tool to center the copyright horizontally and then align on the bottom edge. So far, so good. Right? Here's our stage.

Next, our navigation arrows. Let's look at how to make an arrow button. I'm not going into this a great deal as I want to focus on the "programming a flash site" as opposed to "creating each little graphic for a flash site." Michael's got some great tutorials for graphics. I'll leave that to the pros.

To make a button, open up the Library (Ctrl+L). You should see the *SplashMc* we created earlier. Click the *New Symbol…* in the bottom left of the Library panel. Name the button *ForwardBtn* and be sure to select *Button*. Click *Ok*.

As a side note here, it's a good idea to create some folders in the library to organize everything. One for MovieClips, one for Buttons, etc.

Back on the stage now, place your arrow button center stage. Add a new keyframe in the *Over* frame. Make the button how you want it to look when you hover over it–I just changed my fill color to white. Then again for the *Down* frame.

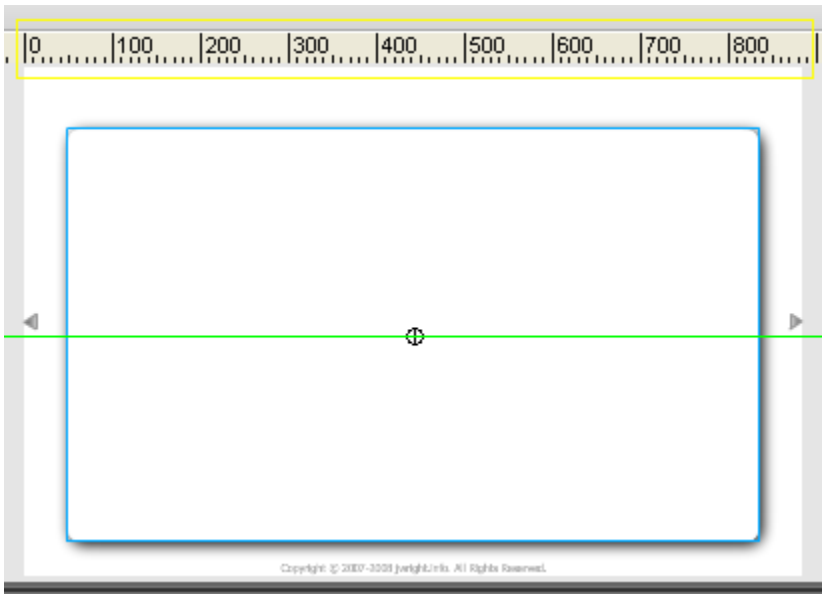
Follow the same steps for your back button. Be sure to name it *BackwardBtn*.

Let's go back to the main stage by clicking *Scene1* as shown below.

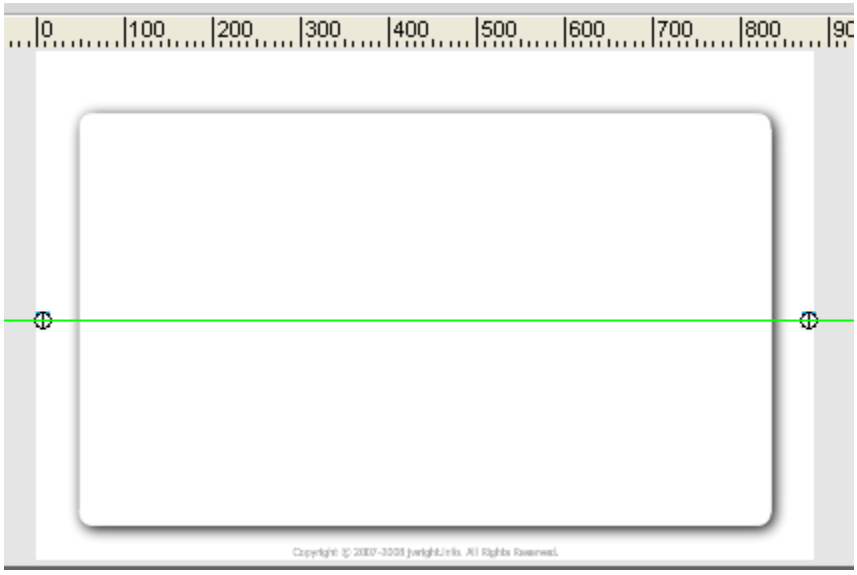
From the Library, drag your two buttons onto the *Arrows* layer. Using the Align tool, align the backward arrow on the left edge and the forward arrow on the right edge.

To align the arrows vertically we're going to use Rulers. We don't want to just "Align vertically" because our *SplashMc* is not aligned vertically. Remember?
We moved it down some? We want the arrows to be aligned with good ol' splashy.

Bring up the rulers by selecting *View–>Rulers* (or Ctrl+Alt+Shift+R). Select the *SplashMc* on the *Splash* layer (you may need to unlock it). Drag a ruler guide down from the horizontal ruler, intersecting the center point of splashy.



Now select both arrows and intersect them with your ruler guide, like so.



On to the logo. Use whatever logo you'd like. Place it on the *Logo* layer.

You may do the same for the little house button on the upper right. Place it on the *House* layer.

Our main layout so far.

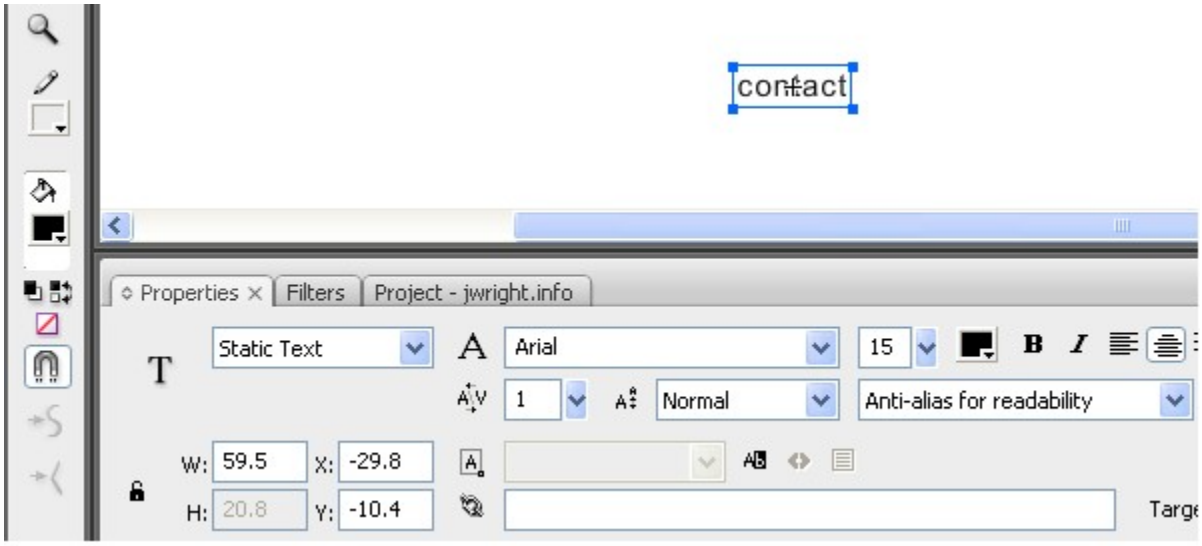
The Flying Menu

Although the main menu is definitely a part of our main layout, it deserves its own section. This, my friends, is the most difficult part of the site; however, it's just a bit tedious–nothing earth shattering. Let us begin!

To get started, create a new symbol (just like you did for the arrow buttons). Name it *MainMenuMc* and select *Movie clip* as the type. Click *Ok*.

Secondly, let's setup or Mc's timeline. Create a new folder called *Menu Items* and insert five new layers into the folder. Name them *Profile, Resume, Contact, Portfolio,* and *Respect* respectively (no pun intended).

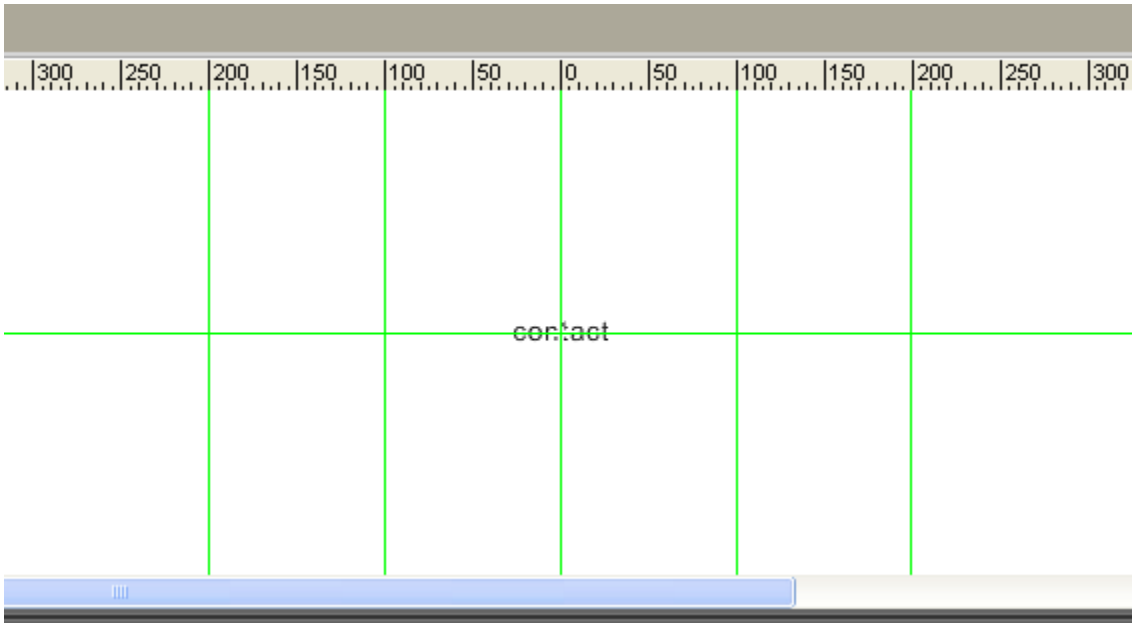
On the *Contact* layer, place a new Text Box with the text *contact*.
This text should be black, 15pt font, Arial, with a letter spacing of 1. See below.



Convert the Text Box to a button. Select it, hit F8, select type *Button*,
name it *ContactBtn* click *Ok*. Double click the new symbol to edit it.
Insert a new keyframe in the *Over* frame and change the text color to #999999.

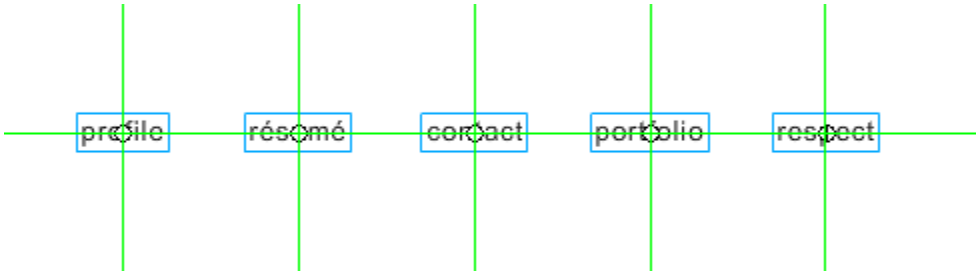
Now let's set up some guide lines. From the *ContactBtn*'s center point, drag two vertical guides to the left of the button. One spaced 100 units away and the other spaced 200 units away from the button's center point. Do the same to the right of the button. Also, drag a horizontal guide intersecting the button's center point.

Here's a clarifying example.

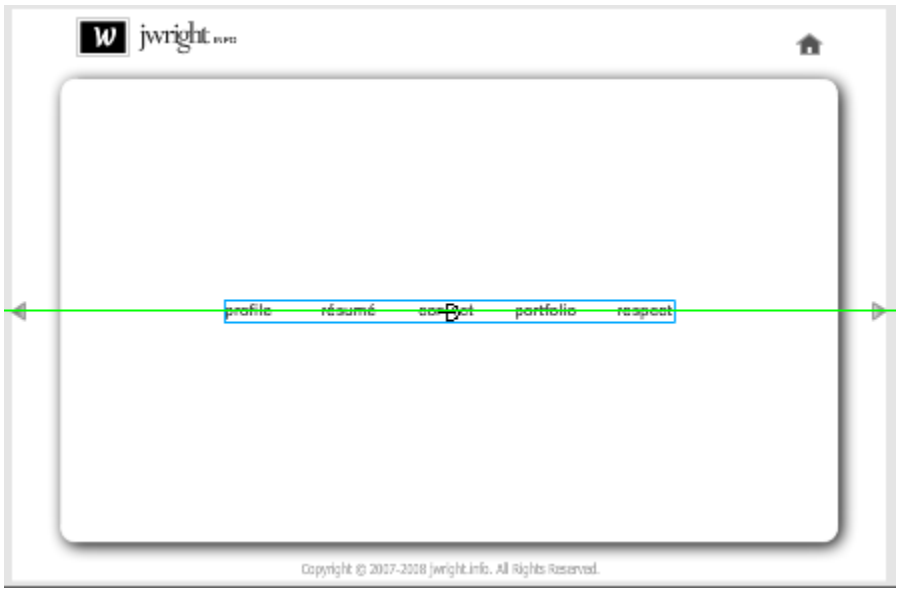


Next, create the other four buttons just like the *ContactBtn* we just made: *ProfileBtn, ResumeBtn, PortfolioBtn,* and *RespectBtn*. Then, place those

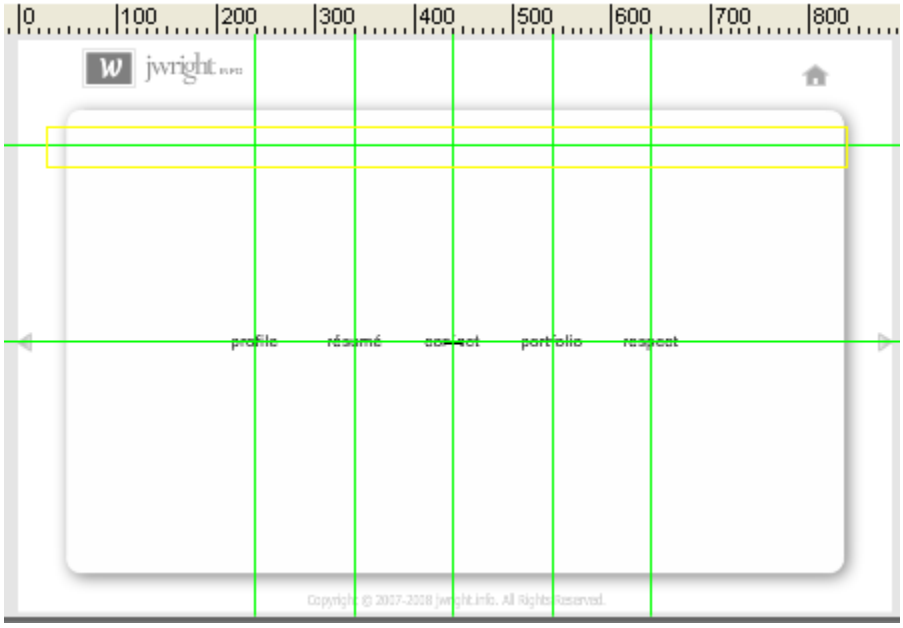
buttons on the intersecting guides like the following. Be sure to place each button on the proper corresponding layer.



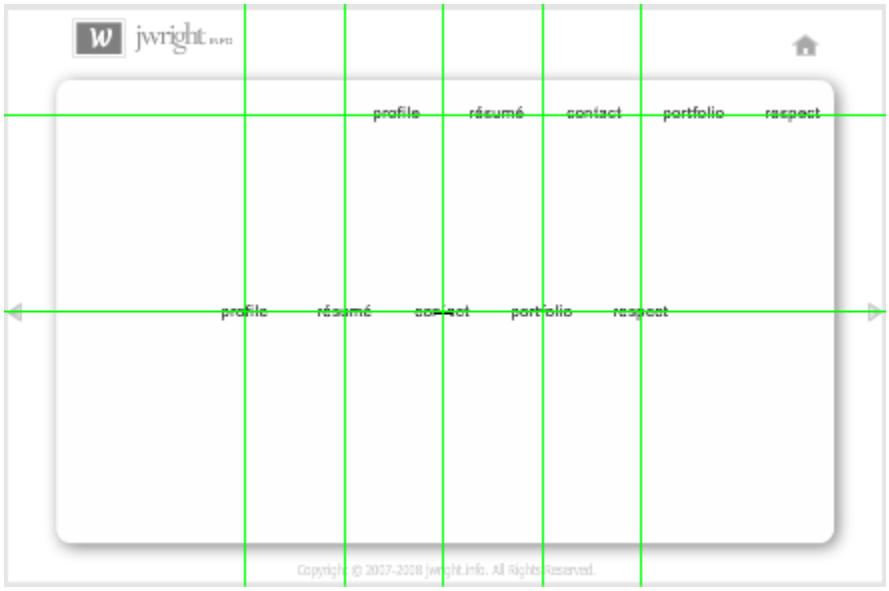
No, we're not done with the menu yet, but we need a reference point for the rest of it. So, let's go ahead and place our menu (from the Library) on our main stage on the *Main Menu* layer–centered vertically and centered horizontally in respect to splashy.



Double click the *MainMenuMc* to return to edit mode. Drag a horizontal guide to where you want the menu to fly to.

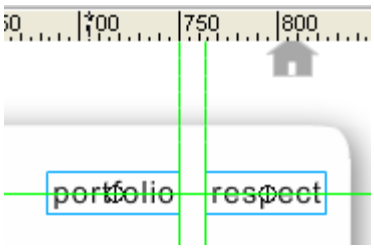


Next, insert a new layer and call it *Dummy*. Copy and paste all five buttons into this dumb layer. Align them horizontally on the guide we just made. Trust me on this one. Here's a look see.

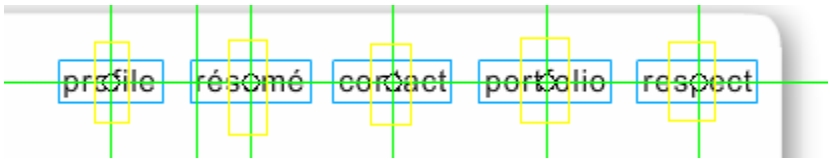


This is going to be how we set up our destination point for our menu items when they fly into place. If you can think of a better way, awesome. Go for it.

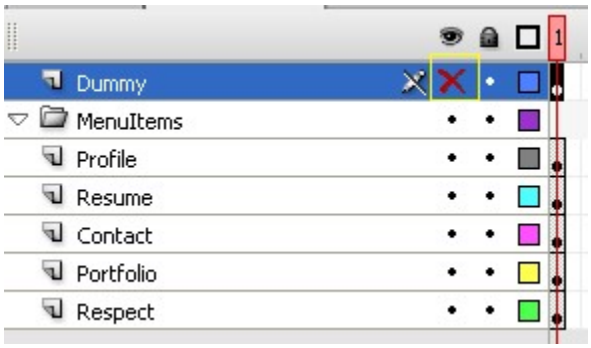
Space each button about 3 little units away from the button to its right. Use the guide lines to help.



Now intersect some guides on each dummy menu item's center point. Be careful not to confuse these guides with the ones for our real menu.

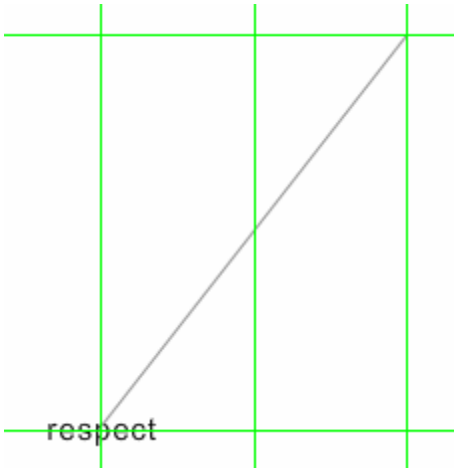


Okay, now hide the dummy layer. We could delete it now as we won't need it anymore, but just in case we need to return to it–it's there. Hide by selecting the layer and clicking the dot on the dummy layer under the *Show/Hide All Layers* button.



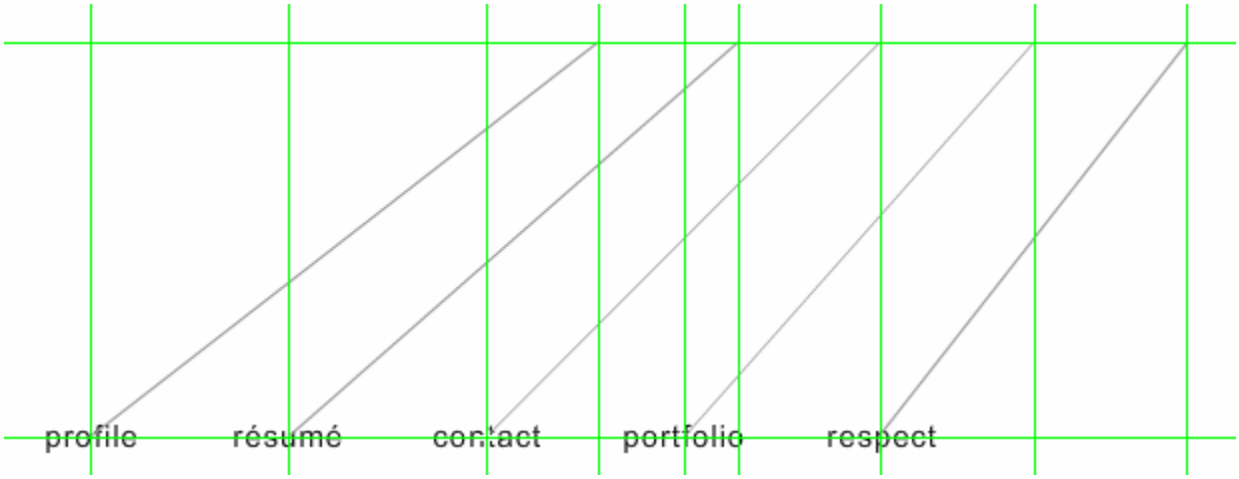
On to the flight plan. Let's draw some motion guides. First, select the *Respect* layer. Click the *Add Motion Guide* button.

Using the Line tool (N), draw a line from the *RespectBtn*'s center point to it's corresponding flight destination. Be sure to draw the guide on the the *Repect* layer's motion guide layer.



Follow the previous steps for the rest of the buttons.

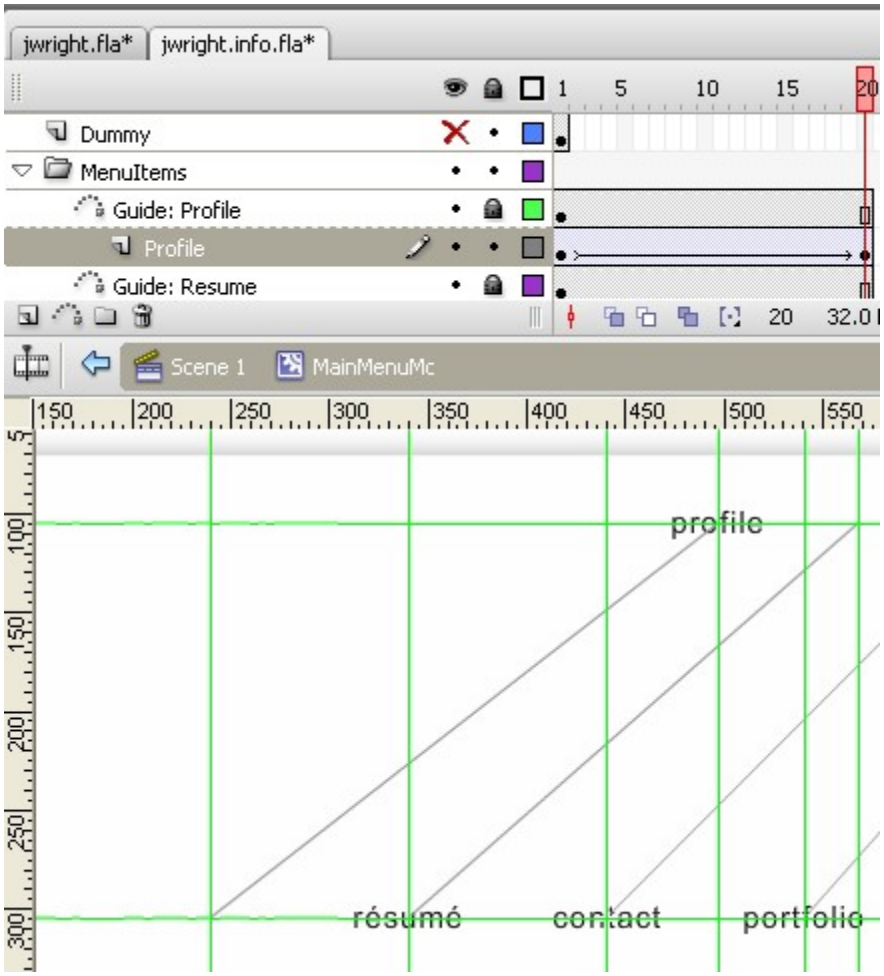
Here's what you should end up with.



Go ahead and lock each guide layer so that we don't accidentally select it.

Now for some animation! All right. Insert a frame (not keyframe) on frame 20 for each layer (excluding *Dummy*). Right click on frame 20 on the profile layer and select *Create Motion Tween*. Carefully select the *ProfileBtn* button and drag it up to its destination point.

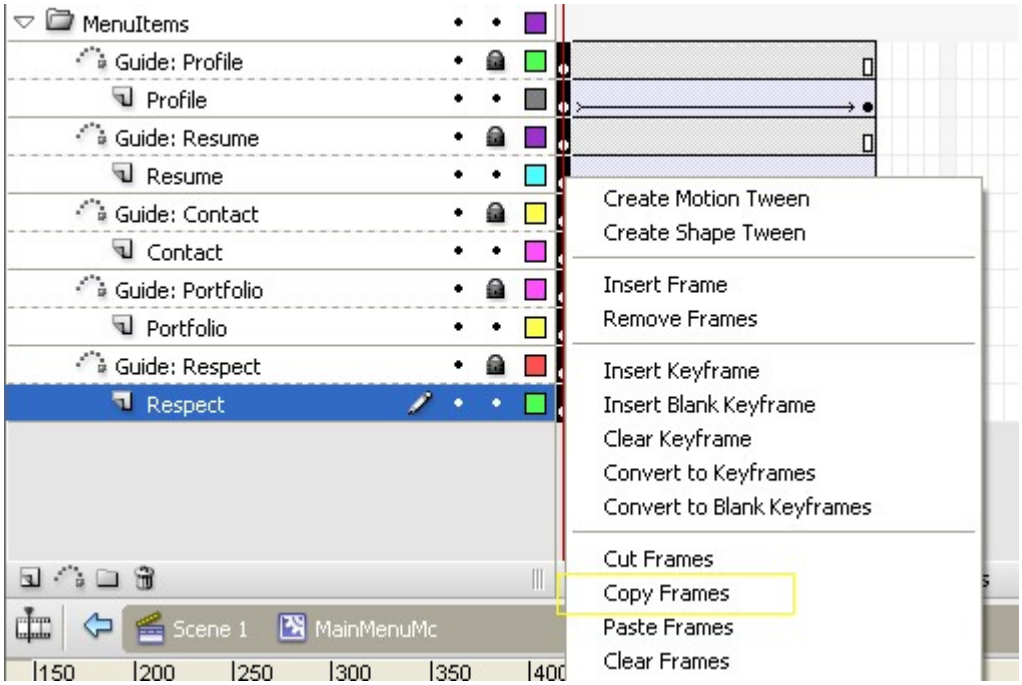
This will create your flying tween. Here's the results.



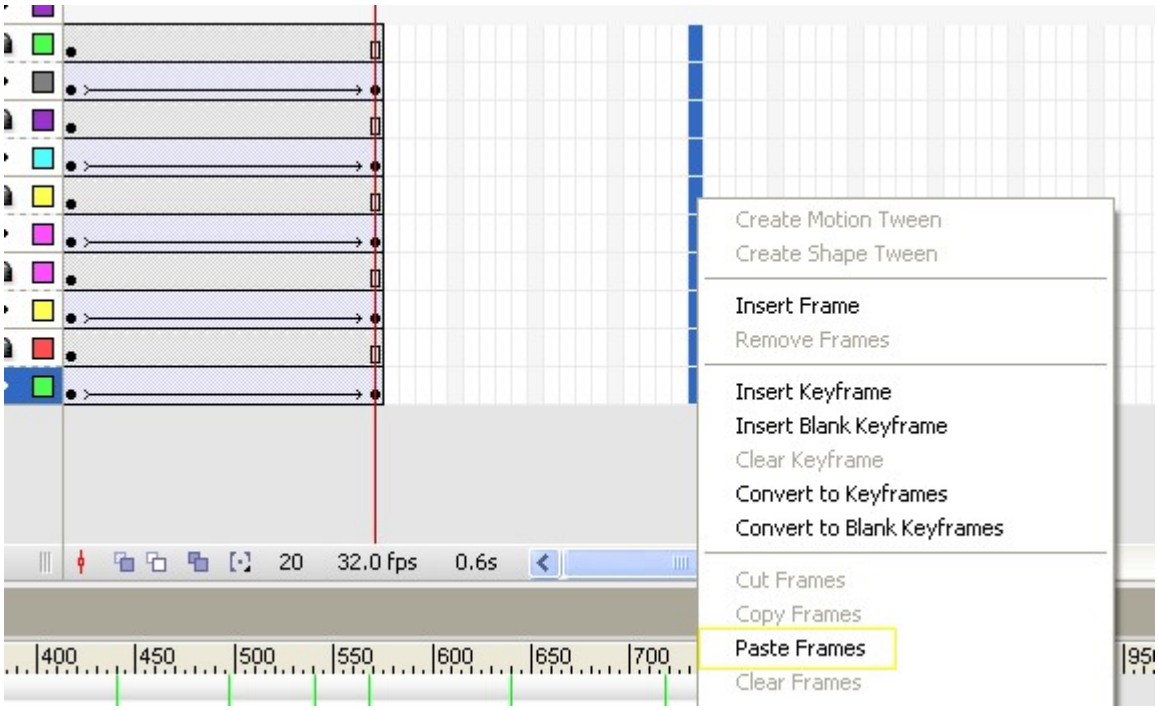
You can drag the red timeline slider back and forth (or use the ‘,’ (comma) and ‘.’ (period) keys) to see your tween in action.

Proceed to do the same for the rest of your menu item buttons.

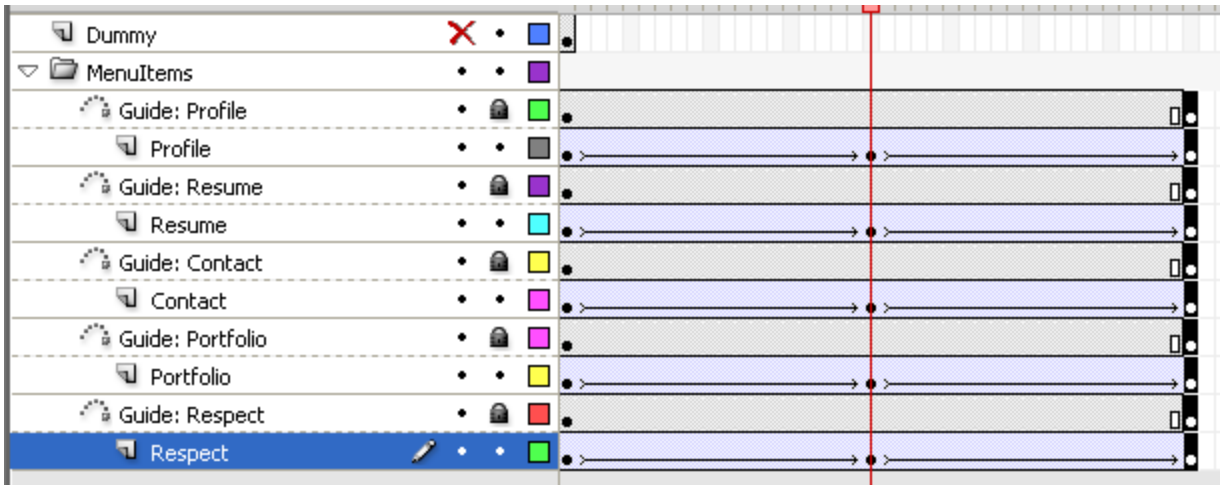
So that’s the up flight plan. What about it’s descent? Well, I’m glad you asked.
This is simple. Select frame one on each layer, right click on the selection. Choose *Copy Frames*.



Now select frame 20 for each layer and select *Paste Frames*, like so.



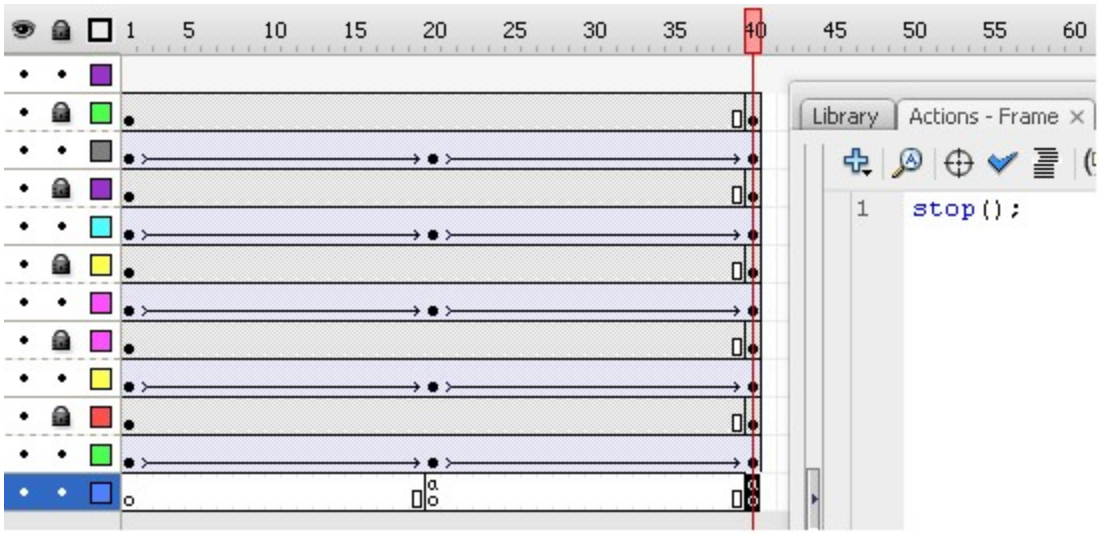
Well, that was the descent. Check it out using the slider to see your animated menu in all its glory! (Go ahead and delete the dummy layer if you'd like.)



Almost done with the menu!

Time for some ActionScript! We need to add some timeline control for our menu. Insert a new layer named *Timeline Control*. Select frame 20 in the new layer and hit F9. Like magic the ActionScript panel opens. Simple stuff here. Just type `stop();` This will stop the menu at the top of the flight plan when it's played.

Do the same for frame 40 so that the menu will stop at the bottom of its flight plan. Notice the little *a* denoting the frame has some ActionScript. Here's a glance.



We need a way to control the menu from the main timeline. So, the solution? We need to create two functions. First, add another layer in the *MainMenuMc* timeline called *Actions*.

Add the following ActionScript to the *Actions* layer.

```
function goHome():void
{
    gotoAndPlay(20);
}
```

```
function leaveHome():void
{
    gotoAndPlay(1);
}
```

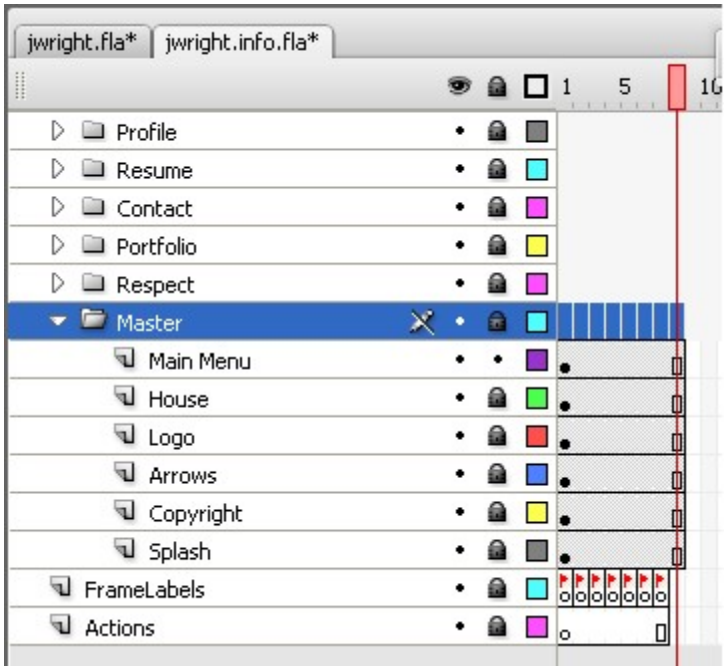
We'll be able to use these two functions to control our main menu.

Congrats! That's the end of our main menu. That wasn't too bad, right? Great job!

Navigation Control

Now that we have our flying menu, logo button, and our little house on the stage button, let's put 'em to good use. We'll be using a good bit more ActionScript (AS) from here on out. Excited?

The first thing we need to do is make sure that our *Master* layer displays throughout the timeline. To do so,
insert a frame into frame 8.



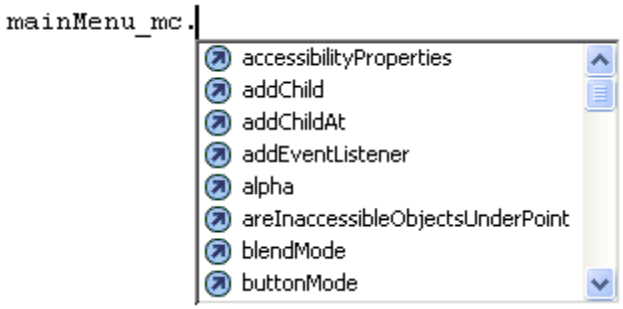
The next thing we need to do is to name our menu. Select the *MainMenuMc* symbol and set it's *Instance Name*
to *mainMenu_mc*. Now, let's add a couple lines of ActionScript to the *Actions* layer that'll run when the
flash file first loads.

```
////////////////////////////////////////////////////////////////////
// Startup.
////////////////////////////////////////////////////////////////////
stop();
mainMenu_mc.stop();
```

The Startup block serves no purpose but to identify the associated lines of code–essentially keeping our
code looking pretty and easy to find.

If you have not done so yet, go back into your *MainMenuMc* symbol and
assign *Instance Name*s to our menu item buttons. Try to stick with the same naming convention. Like, name the
ProfileBtn profile_btn, etc.

Notice how we use the *_btn* postfix for Buttons and the *_mc* postfix for Movie Clips. This will
tell the AS compiler to generate intellisense (What's that? See below.) for your object. Kinda like Visual Studio,
Dreamweaver, and other programming IDEs.



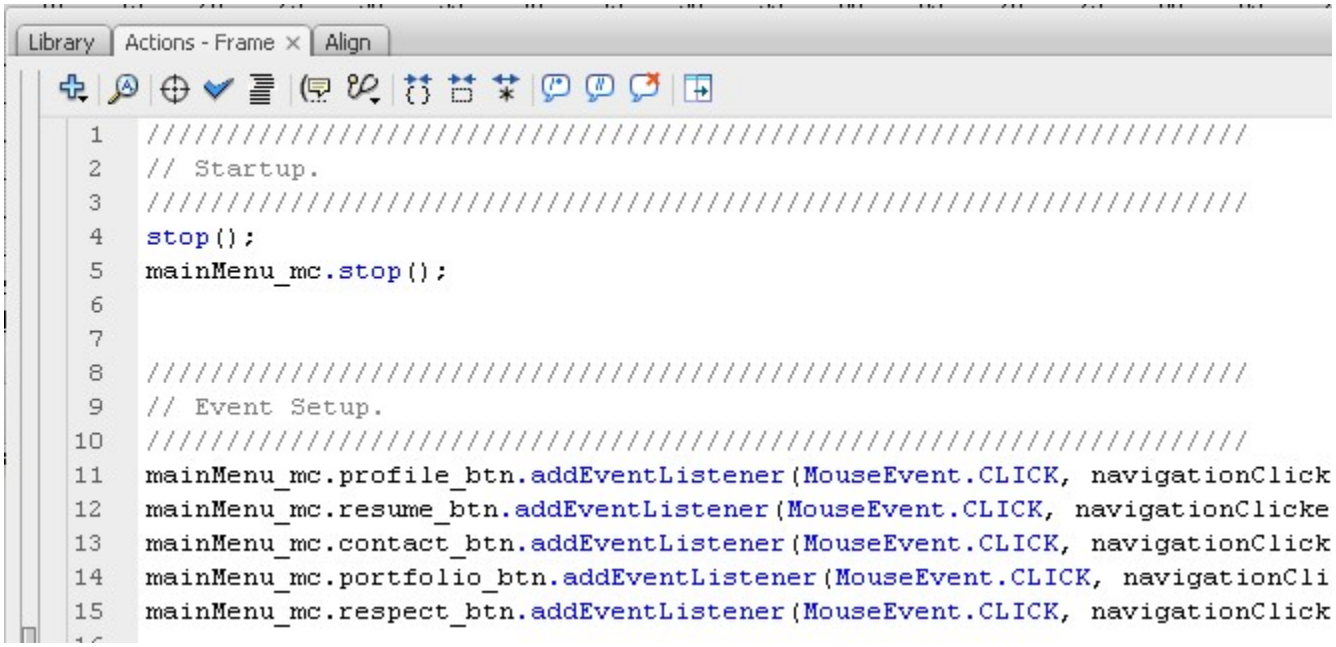
In order to handle the onclick events of our navigational buttons, we need to setup some event handlers.
Add the following lines of AS to the *Actions* layer. I'll explain the `navigationClicked` part in a second.

```
//////////////////////////////////////////////////////////////////////
// Event Setup.
//////////////////////////////////////////////////////////////////////
mainMenu_mc.profile_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
mainMenu_mc.resume_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
mainMenu_mc.contact_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
mainMenu_mc.portfolio_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
mainMenu_mc.respect_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
```

So what in the world did we just do? It's called event handling programming. I'm sure you're familiar with it, but it's basically just saying that function "X" gets called when the user clicks button "B"–pretty basic.

So, we have hooked up our main menu's navigational buttons to a function called `navigationClicked`
that we are about to define. Now, any time one of those buttons are clicked, this function will be executed.

Here's what you should have in your *Actions* layer for the main stage thus far.



I know it's getting a little more complicated, but hang in there and take your time.

Next, we need to define our `navigationClicked` function. Here's the stubbed out function. We'll add more to this function in a minute. Copy and paste this code under your other code in the *Actions* layer.

```
//////////////////////////////////////////////////////////////////
// Event Handlers.
//////////////////////////////////////////////////////////////////
function navigationClicked(Event:MouseEvent):void
{
}
```

This next batch of AS (the body of the `navigationClicked` function) is a little long; however, it's not too complicated. I'll try to explain it well as we go along.

We need to determine what page to go to based on what button was clicked. Remember the Frame Labels we discussed earlier? Here's where we're going to use them. Bascially, we're going to associate each button with a Frame Label via a common `switch` statement.

```
function navigationClicked(Event:MouseEvent):void
{
    //We'll use this to store the Frame Label's name.
    var frmLabel:String = '';

    //Determine what Frame Label to use based on which
    //button was clicked.
    switch (Event.target)
    {
        case mainMenu_mc.profile_btn :
            frmLabel = "profile_frm";
            break;
        case mainMenu_mc.resume_btn :
            frmLabel = "resume_frm";
            break;
        case mainMenu_mc.contact_btn :
            frmLabel = "contact_frm";
            break;
        case mainMenu_mc.portfolio_btn :
            frmLabel = "portfolio_frm";
            break;
        case mainMenu_mc.respect_btn :
            frmLabel = "respect_frm";
            break;
        case logo_btn :
            frmLabel = frmLabel = "home_frm";
            break;
        case home_btn :
            frmLabel = "home_frm";
            break;
    }
}
```

Okay, so now we know what Frame Label we need to use. (You may be asking what about our *LogoBtn* and *HouseBtn*. We'll get to that in a minute.)

In order to display the page we'll use the built-in AS function `gotoAndPlay(frame:Number)`. We have
a Frame Label which is a `string`; however, we need the frame's number. Let's write a little function to retrieve that number. Here ya go.

```
//////////////////////////////////////////////////////////////////
// Helper Functions.
//////////////////////////////////////////////////////////////////
function getFrame(frameName:String):Number
{
    var frame:Number = 1;

    //Loop through all Frame Labels to find our requested frame.
    for (var i = 0; i < currentLabels.length; i++)
    {
        if (currentLabels[i].name == frameName)
        {
            frame = currentLabels[i].frame;
            break;
        }
    }

    return frame;
}
```

All we are doing here is looping through all of our Frame Labels (via the built-in `currentLabels` property)
and comparing the name we found in our `switch` block to the label we're currently looping on.

We can now add the following two lines to the bottom of our `navigationClicked` function.

```
var frmGoto:Number = this.getFrame(frmLabel);
gotoAndPlay(frmGoto);
```

Here's our function so far.

```
function navigationClicked(Event:MouseEvent):void
{
    //We'll use this to store the Frame Label's name.
    var frmLabel:String = '';

    //Determine what Frame Label to use based on which
    //button was clicked.
    switch (Event.target)
    {
        case mainMenu_mc.profile_btn :
            frmLabel = "profile_frm";
            break;
        case mainMenu_mc.resume_btn :
            frmLabel = "resume_frm";
            break;
        case mainMenu_mc.contact_btn :
            frmLabel = "contact_frm";
            break;
        case mainMenu_mc.portfolio_btn :
            frmLabel = "portfolio_frm";
            break;
        case mainMenu_mc.respect_btn :
```

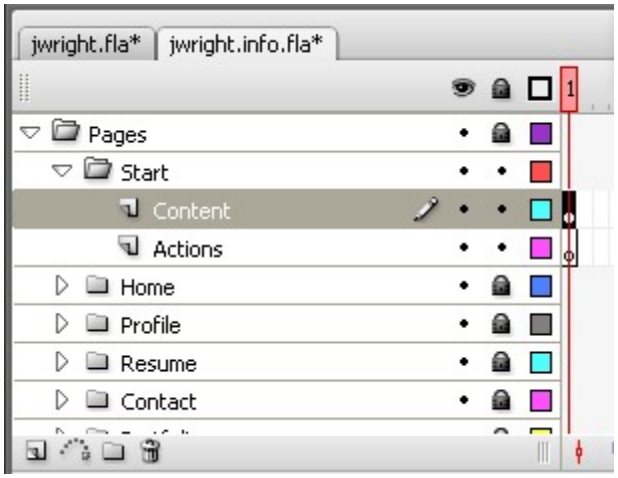
```
            frmLabel = "respect_frm";
            break;
        case logo_btn :
            frmLabel = frmLabel = "home_frm";
            break;
        case home_btn :
            frmLabel = "home_frm";
            break;
    }

    //Find the frame number based on our Frame Label.
    var frmGoto:Number = this.getFrame(frmLabel);

    //Go to the requested page.
    gotoAndPlay(frmGoto);
}
```

Well, let's test out what we have so far. Select *Debug–>Debug Movie* (Ctrl+Shift+Enter) from Flash CS3's main menu. This will compile and run your Flash file. If you've missed anything, the compiler will show you an error.

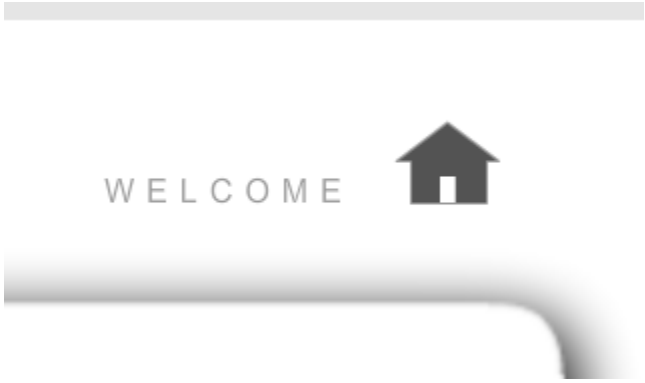
Here's what you should see.

Feel free to click some buttons, but you really won't be able to see the pages changing since all of our pages are blank. Let's skip ahead a little and change that. That way we'll know our menu is working.

First, select the frame in your *Content* layer under the *Start* page.



Secondly, place a Text Box, with text-aligned right, by the *HouseBtn* symbol, like so.



Next, follow those same steps to place a label (in the same place) on the *Content* layers of each page. Then change the text to represent each page.

Tip: You can select the Text Box you just made on your *Start* page, select the frame for the *Content* layer on
the *Home* page, and select *Edit–>Paste in Place* (or Ctrl+Shift+V). This will paste the Text Box in the exact same place as your *Start* page.

You will also need to add the following one line of AS to each of the *Actions* layers for each page– except the *Start*
page.

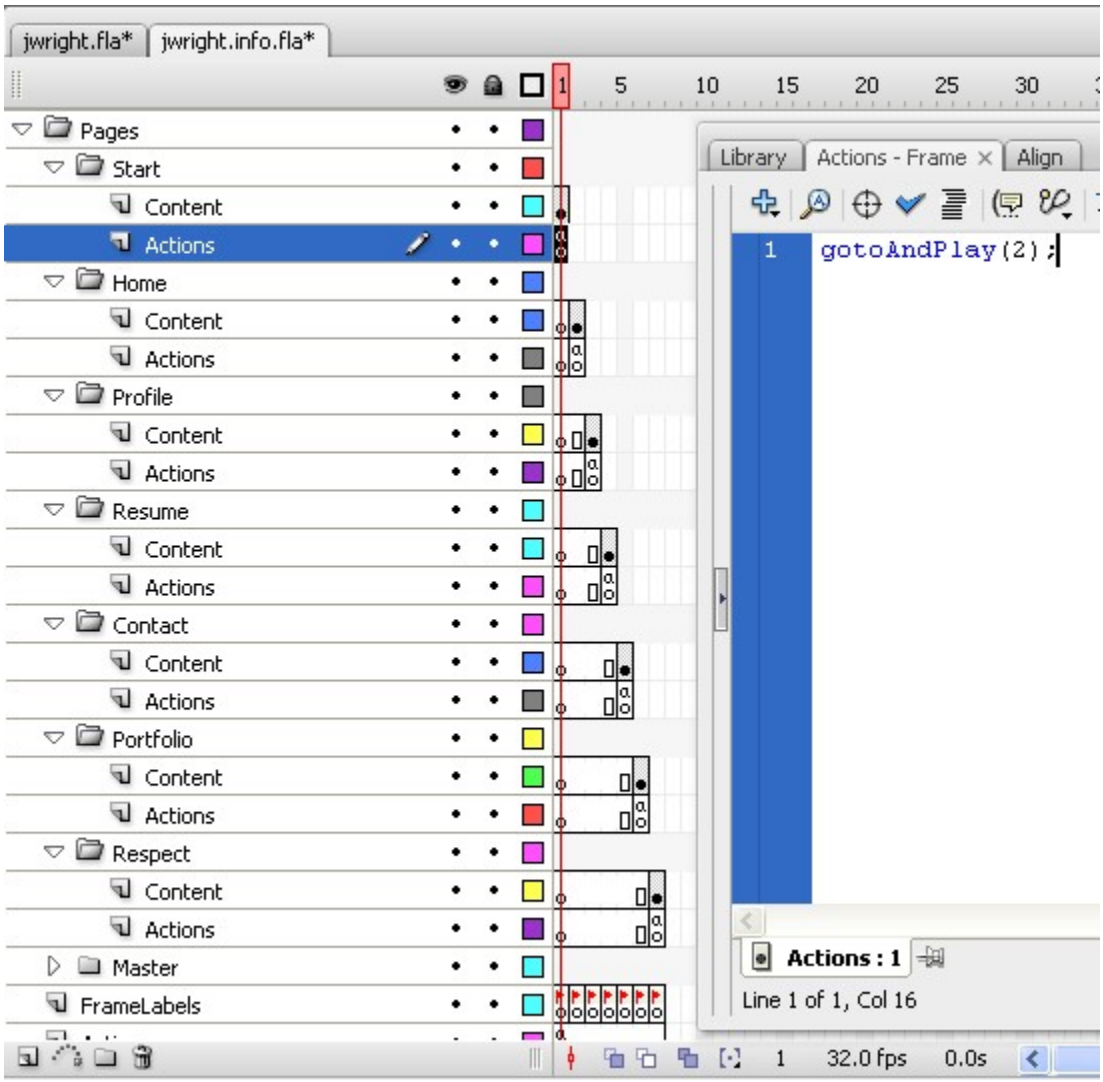
```
stop();
```

This will ensure that your Flash movie will stop on the requested frame. If not, we would be hitting the requested
frame and then the movie would continue to play out the rest of the movie.

As for the *Start* page, we need to add the following line of AS. We're bascially going to
just skip the start page. The reason is simple. I may want to, at a later time, change my home page to something
different and make the current home page a welcome page. But, for now we'll just use our home page as our welcome.
Basically, we're just trying to keep our site as maintainable as possible.

```
gotoAndPlay(2);
```

Again, notice the little ‘a’ on each frame containing AS code.



Before we debug again, let’s add a couple of lines of AS to our `navigationClicked` function.

We’re going to put our `gotoAndPlay(frmGoto);` line of AS inside of an `if` statement. Here’s the function now.

```
function navigationClicked(Event:MouseEvent):void
{
    //We'll use this to store the Frame Label's name.
    var frmLabel:String = '';

    //Determine what Frame Label to use based on which
    //button was clicked.
    switch (Event.target)
    {
        case mainMenu_mc.profile_btn :
            frmLabel = "profile_frm";
            break;
        case mainMenu_mc.resume_btn :
```

```
            frmLabel = "resume_frm";
            break;
        case mainMenu_mc.contact_btn :
            frmLabel = "contact_frm";
            break;
        case mainMenu_mc.portfolio_btn :
            frmLabel = "portfolio_frm";
            break;
        case mainMenu_mc.respect_btn :
            frmLabel = "respect_frm";
            break;
        case logo_btn :
            frmLabel = frmLabel = "home_frm";
            break;
        case home_btn :
            frmLabel = "home_frm";
            break;
    }

    //Find the frame number based on our Frame Label.
    var frmGoto:Number = this.getFrame(frmLabel);

    //Don't do anything if we are already on the requested page.
    if (currentFrame != frmGoto)
    {
        //Go to the requested page.
        gotoAndPlay(frmGoto);
    }
}
```

Go ahead and debug now to see your changes in action. Notice how the page label by the *HouseBtn* symbol changes as you go from page to page.

Next, let's set up our *LogoBtn* and *HouseBtn* buttons. After you assign an *Instance Name* to your buttons, all you need to do is attach the click event of those buttons to our `navigationClicked` function. Forget how? That's cool. Here's a refresher.

```
logo_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
home_btn.addEventListener(MouseEvent.CLICK, navigationClicked);
```

So now we need to add those guys to our `switch` statement. After you do that, feel free to debug again to check it out.

```
case logo_btn :
    frmLabel = "home_frm";
    break;
case home_btn :
    frmLabel = "home_frm";
    break;
```

How about that flying menu? It's time to put it to work! Remember those two functions we made, `goHome();` and `leaveHome();`? That's how we're going to control the menu from our `navigationClicked` function.
Let's have a look how.

All we need to do to implement our flying menu is make a couple of modificaitons to our `if` statement
in our `navigationClicked` function. Below are the changes. Be sure to read the commented lines to understand what we're trying to accomplish.

```
//Don't do anything if we are already on the requested page.
if (currentFrame != frmGoto)
{
    //Get and remember the Home page's frame number.
    var frmHome:Number = this.getFrame("home_frm");

    //If our requested page is the Home page, the flying menu
    //needs to go home.
    if(frmGoto == frmHome)
    {
        mainMenu_mc.goHome();
    }
    //Else, if we are on the Home page and are leaving, then
    //leave home.
    else if(currentFrame == frmHome)
    {
        mainMenu_mc.leaveHome();
    }

    //Go to the requested page.
    gotoAndPlay(frmGoto);
}
```

We've only got two buttons to go. Our *ForwardBtn* and *BackwardBtn* buttons.

To get started with those fellas, assign them *Instance Name*s and hook them up to our `navigationClicked` function as we've done several times previously.

It's a tad tricky to get our *forward_btn* and *backward_btn* buttons to cycle the pages appropriately. I think it will be much clearer if I just give you the code and have you read through the comments–instead of dissecting the functions line by line.

The main function here is `getSequencedFrame(forward:Boolean)`. This function accepts a `Boolean` value indicating if the next page is requested or the previous. Then, the function will determine the Frame Label of the requested page.

Here's the two functions you'll need. Just copy and paste them into your *Helper Functions* section.

```
function getFrameLabel(frame:Number):String
{
    var frmLabel:String = '';

    //Loop through all Frame Labels to find the requested Frame Label.
    for (var i = 0; i < currentLabels.length; i++)
    {
        if (currentLabels[i].frame == frame)
        {
            frmLabel = currentLabels[i].name;
            break;
        }
    }
```

```
    return frmLabel;
}

function getSequencedFrame(forward:Boolean):String
{
    //Used to remember the Frame Label of our page.
    var frmSequence:String = '';

    //If we're looking for the next page in sequence...
    if(forward)
    {
        //If our current page is that last page...
        if(currentFrame == this.getFrame("respect_frm"))
        {
            //...then, we need to go Home.
            frmSequence = "home_frm";
        }
        else
        {
            //...else, we just need to go to the next frame.
            frmSequence = this.getFrameLabel((currentFrame + 1));
        }
    }
    //...else, we're looking for the previous page in sequence.
    else
    {
        //If we're on the first page (remember, we skipped our Start page)...
        if(currentFrame == this.getFrame("home_frm") ||
           currentFrame == this.getFrame("start_frm"))
        {
            //...then, we need to go to the last page.
            frmSequence = "respect_frm";
        }
        else
        {
            //...else, we just need to go to the the previous frame.
            frmSequence = this.getFrameLabel((currentFrame - 1));
        }
    }

    return frmSequence;
}
```

Almost there! Just add this code to your `switch` statement and you're done.

```
case backward_btn :
    frmLabel = this.getSequencedFrame(false);
    reak;
case forward_btn :
    frmLabel = this.getSequencedFrame(true);
    break;
```

Congrats folks! You've just completed the hardest part of jwright.info.

From here on out, let's make each page its own section. That way it will be easier for you to stop and pick up again later if need be.

The Home Page

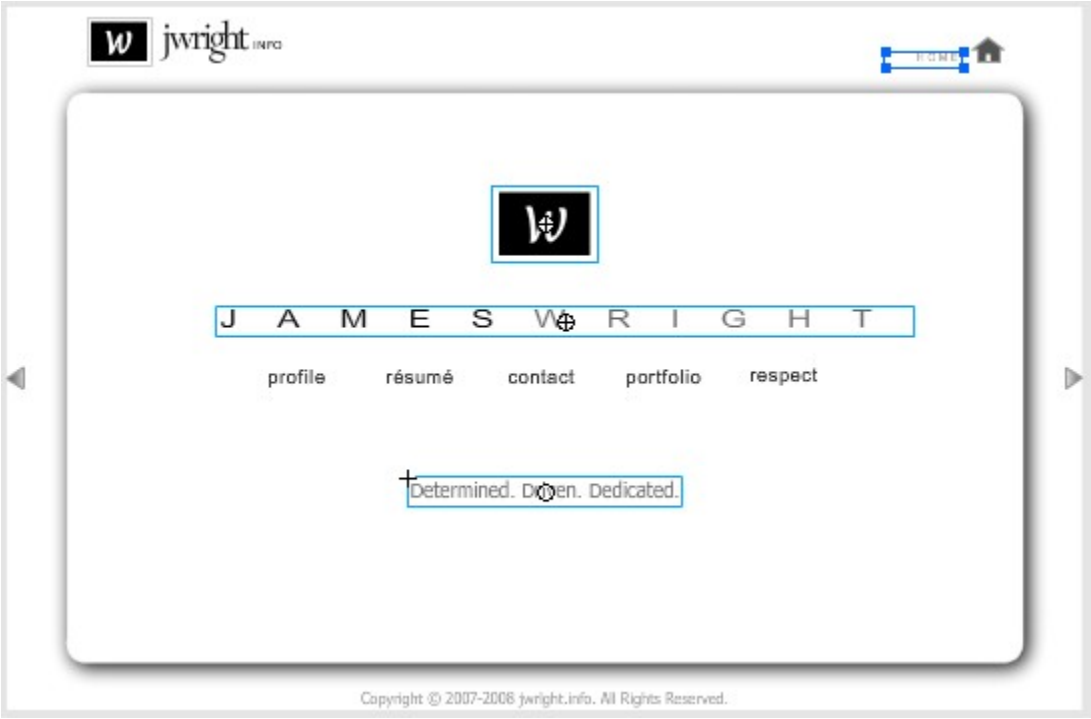
First things first. Let's spice up our *Home* page! Make sure that the following symbols go on the *Content* layer of our *Home* page layer.

Create a new Text Box and type in JAMESWRIGHT or use your own name for kicks. Use Arial, 28pt font, and
letter spacing of 33. Now, convert this Text Box to a Movie Clip called *NameMc*. Also, while we're there,
give it an *Instance Name* of *name_mc*.

You may use my logo if you'd like or your own–as long at it's a Movie Clip with an instance name of *logo_mc*.
Drop it onto the *Home* page as well.

I also made a little footer called *DetDriDetMc* with an *Instance Name* of *ddd_mc*.

Here's a look at our *Home* page.



Go ahead and test it out! You can debug like before, or hit Ctrl+Enter to run test mode.

Pretty cool, huh? Well it's about to get better. Let's add some fade transitions. Since we're going to be
using the same transitions on multiple pages and other Movie Clips, we need to put the transition code into its own class. Here's how.

Select *File–>New–>ActionScript File*. Click *Ok*. Save the file in the same directory as your Flash files named *utils.Fader*. Two things are crucial here. The location of the file and the naming of the file. It must be in the same directory because this is how your AS code will reference this file. Also, the
name signifies the package (a.k.a namespace) to be used.

Add the following code to your new *utils.Fader.as* file.

```
package utils
{
    //Import the needed packages.
    import fl.transitions.Transition;
    import fl.transitions.TransitionManager;
    import fl.transitions.Fade;
    import fl.transitions.easing.*;
    import flash.display.MovieClip;

    //Fader class definition.
    public class Fader
    {
        //This function will apply a "fade in" effect for any given Movie Clip.
        public static function fadeIn(mc:MovieClip):void
        {
            //Use the TransitionManager to fade in the movie clip.
            TransitionManager.start(mc, {type:Fade, direction:Transition.IN, durati
        }

        //This function will apply a "fade out" effect for any given Movie Clip.
        public static function fadeOut(mc:MovieClip):void
        {
            //Use the TransitionManager to fade in the movie clip.
            TransitionManager.start(mc, {type:Fade, direction:Transition.OUT, durat
        }
    }
}
```

Click here to read more about the *TransitionManager*.

Now, go ahead and add this AS file to your Flash project as we did in the beginning of the tutorial.

Next, add this code to your main *Actions* layer. This will allow you to be able to use your *Fader* class on the main timeline.

```
////////////////////////////////////////////////////////////////
// Includes and Imports.
////////////////////////////////////////////////////////////////
import utils.Fader;
```

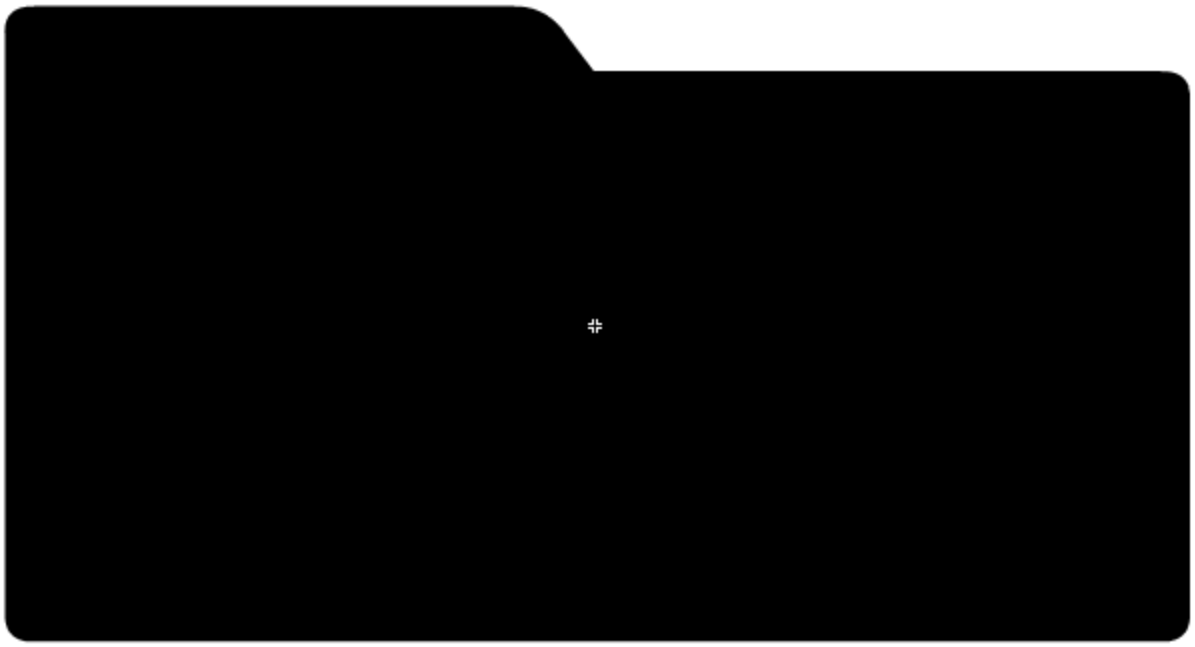
To implement your Fader class, simply add these line of AS to your *Actions* layer for your *Home* page.

```
utils.Fader.fadeIn(logo_mc);
utils.Fader.fadeIn(name_mc);
utils.Fader.fadeIn(ddd_mc);
```

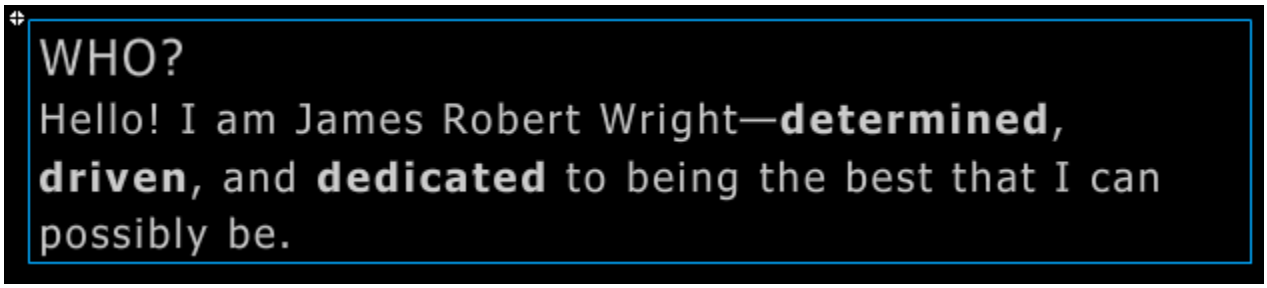
Great! Now you can test it and check out the cool fade effect! That's all for the *Home* page!

The Profile Page

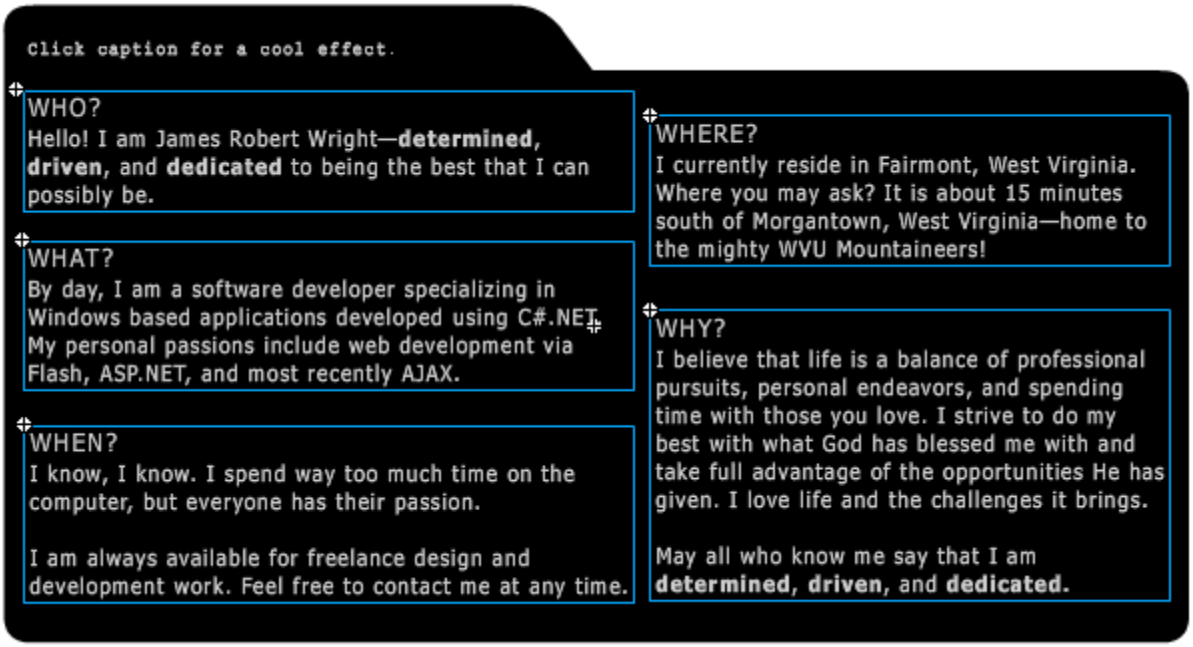
The first thing we need to do is create a new Movie Clip (Remember, you can do this from the Library panel). Then we want
to place a graphic on the MC that kinda looks like a tab from a tabbed document. Place the tab on its own layer called
Tab. Again, you can find this graphic and others in the package supplied with the tutorial.



Secondly, we need to create some buttons–just some simple buttons that tell our readers a little bit about us.
All you need to do is create a new button, as you've done before, and place some text in it. Nothing new here.
Here's a little snapshot of one of my buttons.



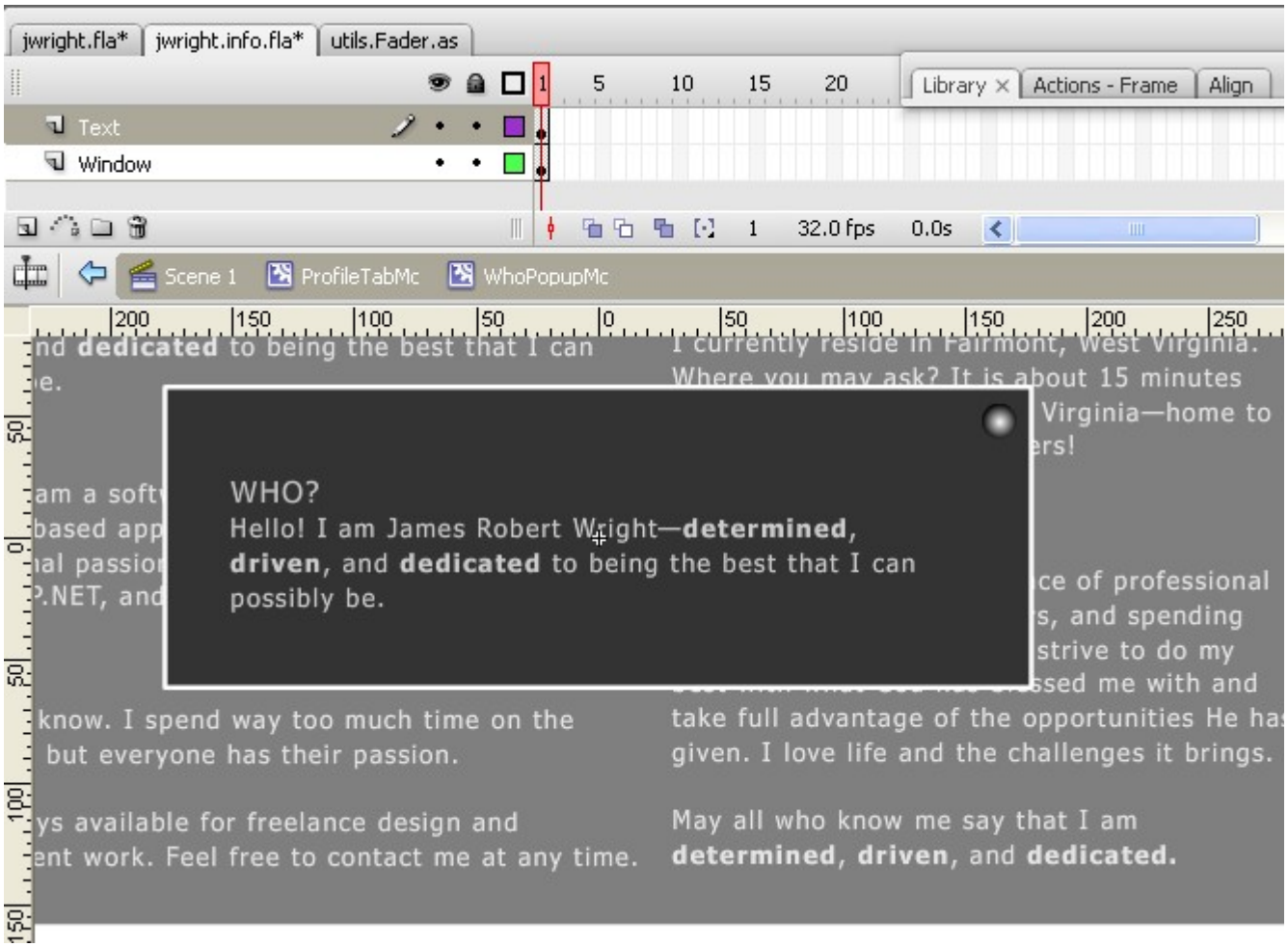
You can create whatever you want here, but if you want to stick with me here, create a button for each *Who, What,*
When, Where, and *Why* section that looks like the following. These buttons should be placed in their own
layer, named *Buttons*.



Go ahead and give each button an *Instance Name*. For example, I named my *WhoBtn* button *whoCap_btn*–since these buttons are like captions.

Now for some pretty cool effects! Let's create some "popups" for each button we just made. To do so, first create
a new movie clip called *WhoPopupMc*. These are real simple to make. Essentially, all you need is two layers:
Text and *Window*.

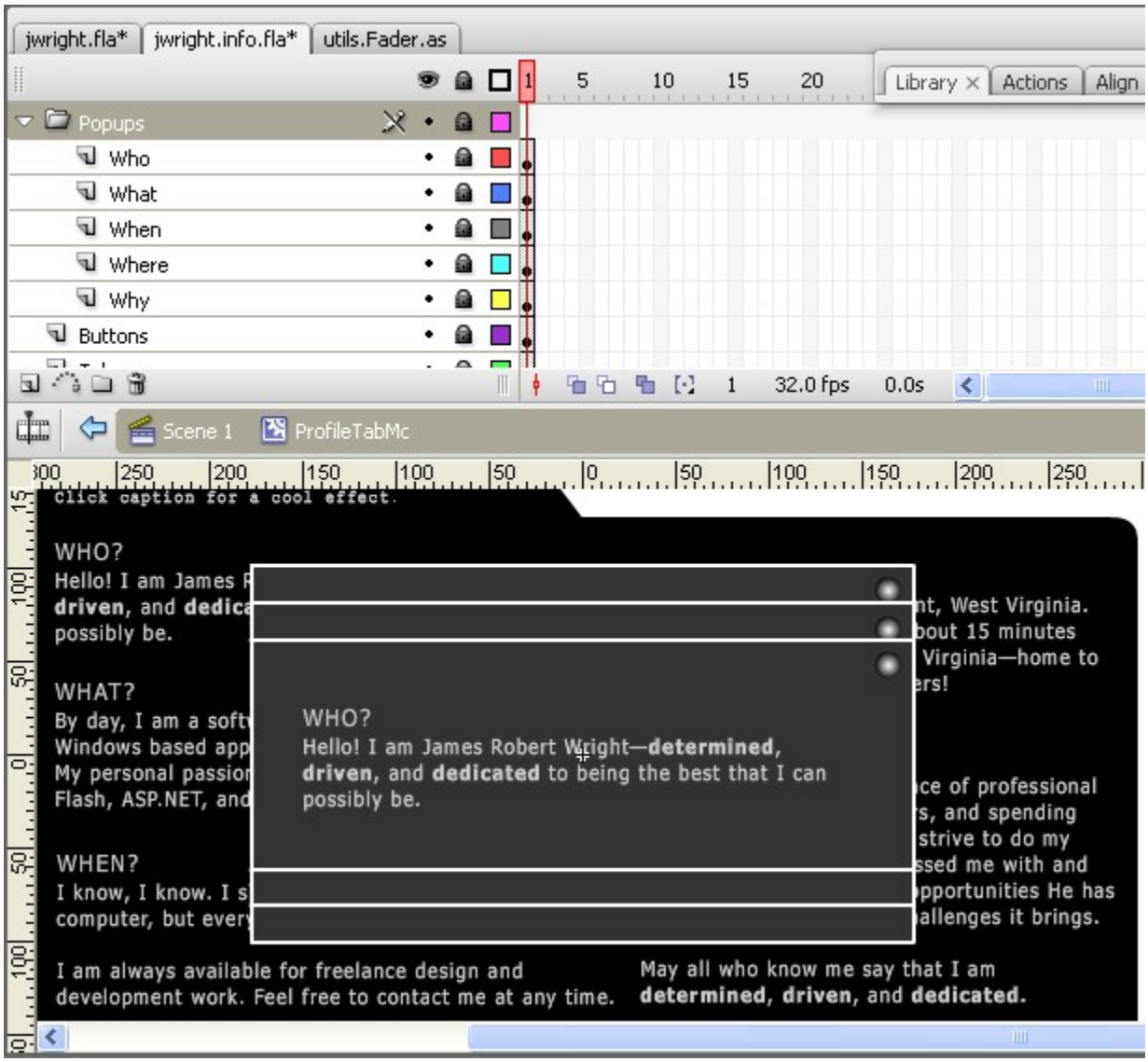
On the *Text* layer, simply place the *WhoBtn* you just made from the Library. Then, on the *Window* layer place a Rectangle, fill color #333333 and stroke color #FFFFFF, that is a little bigger than your button. You
will also need to create another little button used to close the popup. Feel free to make that button whatever
you like. I just used a little circle. Here's a look!



Be sure to give your *CloseBtn* an *Instance Name* called *close_btn*.

Now that you know how to do that, create a "popup" for each button.

Once you have created all of your popups, place each popup on its own layer on your *ProfileTabMc* symbol and assign *Instance Name*s to your popups. Here's what I ended up with.



Next, you'll need to insert another layer called *Actions* to your *ProfileTabMc* symbol. On this layer let's add some AS event handlers for our buttons. Here they are to save you some time and typing.

```
//////////////////////////////////////////////////////////////////
// Event Setup.
//////////////////////////////////////////////////////////////////
whoCap_btn.addEventListener(MouseEvent.CLICK, showPopup);
whatCap_btn.addEventListener(MouseEvent.CLICK, showPopup);
whenCap_btn.addEventListener(MouseEvent.CLICK, showPopup);
whereCap_btn.addEventListener(MouseEvent.CLICK, showPopup);
whyCap_btn.addEventListener(MouseEvent.CLICK, showPopup);
whoPup_mc.close_btn.addEventListener(MouseEvent.CLICK, closePopup);
whatPup_mc.close_btn.addEventListener(MouseEvent.CLICK, closePopup);
whenPup_mc.close_btn.addEventListener(MouseEvent.CLICK, closePopup);
wherePup_mc.close_btn.addEventListener(MouseEvent.CLICK, closePopup);
whyPup_mc.close_btn.addEventListener(MouseEvent.CLICK, closePopup);
```

Before we define our event handler functions, we need to add some AS to help us control which popup is
currently showing. So, how do we do that? Again, nothing new here. We're just going to write a

simple
`switch` statement to set a variable called `currentPopup`.

All that's going on here is we are determining which button was clicked and then setting a Movie Clip variable
to the popup we need to display. Here's the code.

```
//////////////////////////////////////////////////////////////////////
// Variables.
//////////////////////////////////////////////////////////////////////
var currentPup:MovieClip = null;

//////////////////////////////////////////////////////////////////////
// Helper Functions.
//////////////////////////////////////////////////////////////////////
function setCurrentPopup(e:Event):void
{
    //Set the current popup based on which caption was clicked.
    switch(e.target)
    {
        case whoCap_btn:
            currentPopup = whoPopup_mc;
            break;
        case whatCap_btn:
            currentPopup = whatPopup_mc;
            break;
        case whenCap_btn:
            currentPopup = whenPopup_mc;
            break;
        case whereCap_btn:
            currentPopup = wherePopup_mc;
            break;
        case whyCap_btn:
            currentPopup = whyPopup_mc;
            break;
    }
}
```

Next, we need to define our event handler functions `showPopup` and `closePopup`.

I know I say this alot, but, this shouldn't be too complicated. Just a couple basic event handlers. We'll also want to add some cool fading effects here as well, so we're going to import the `utils.Fader` class.

```
//////////////////////////////////////////////////////////////////////
// Includes and Imports.
//////////////////////////////////////////////////////////////////////
import utils.Fader;

//////////////////////////////////////////////////////////////////////
// Event Handlers.
//////////////////////////////////////////////////////////////////////
function closePopup(Event:MouseEvent):void
{
        //If the current popup has been set, fade it out.
    if(this.currentPopup != null)    utils.Fader.fadeOut(currentPopup);
}
```

```
function showPopup(Event:MouseEvent):void
{
    //Set the current popup.
    this.setCurrentPopup(Event);

        //If the current popup has been set, fade it in.
    if(this.currentPopup != null)   utils.Fader.fadeIn(currentPopup);
}
```

To see your changes, you'll need to go back to the main timeline and add your new *ProfileTabMc* symbol
to your *Content* layer in the *Profile* page section. You can use the Free Transform tool (Q) to make it fit to your stage accordingly if the size of your symbol isn't just right.

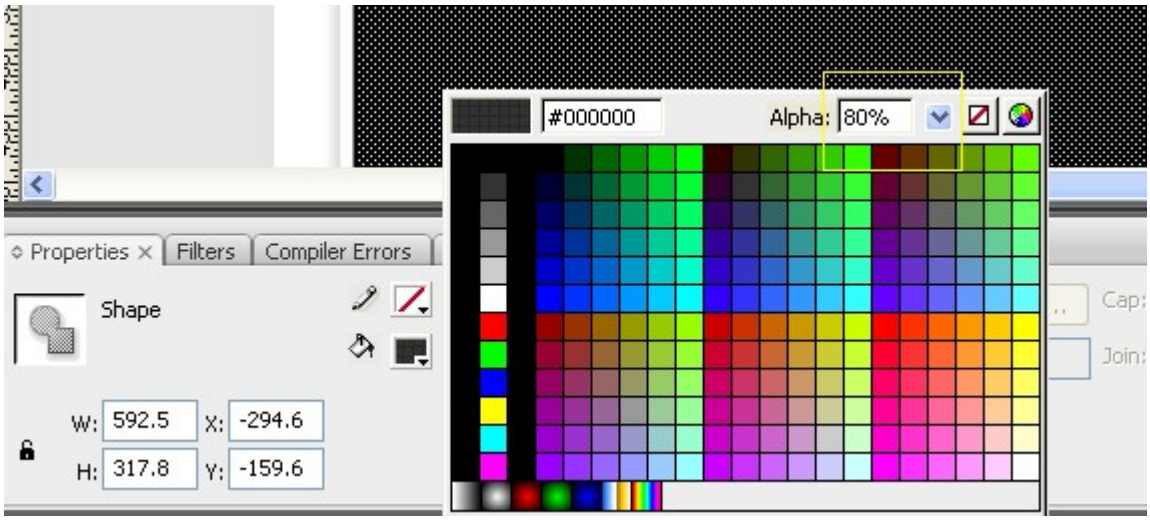
You can preview you profile page now if you'd like (Ctrl+Enter).

You'll probably notice that your popups start out being visible. To fix this issue let's add some more AS to
the *ProfileTabMc*'s *Actions* layer. This AS block will tell the popups to not be visible on startup. After you add this code to your MC, test it out and see the cool effects!

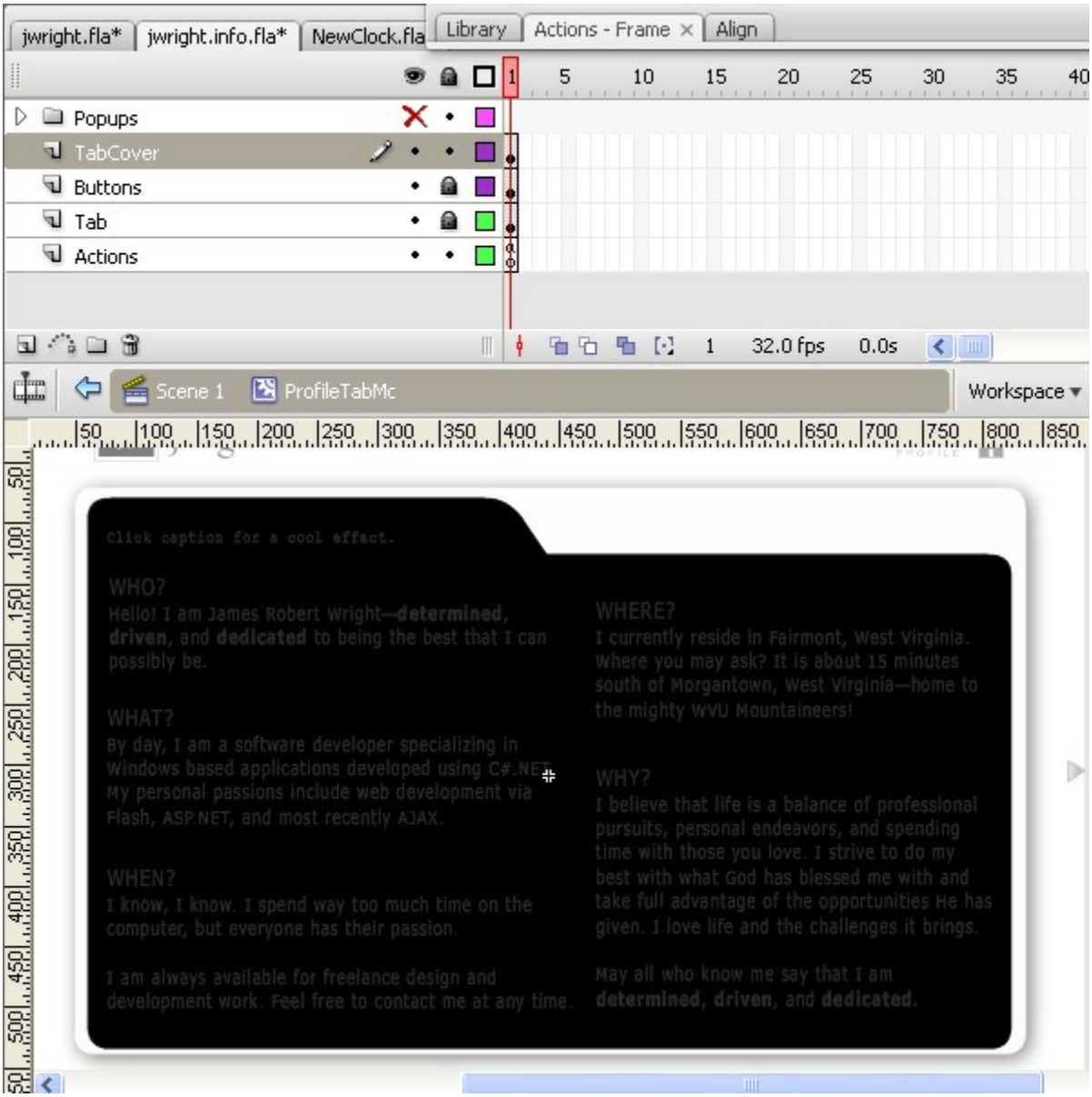
```
//////////////////////////////////////////////////////////////////
// Startup.
//////////////////////////////////////////////////////////////////
whoPopup_mc.visible = false;
whatPopup_mc.visible = false;
whenPopup_mc.visible = false;
wherePopup_mc.visible = false;
whyPopup_mc.visible = false;
```

You like? Cool. Well, only one more tweak before we move on to the next page. If you notice, when you click any
of your buttons, the corresponding popup displays. That's what it should do; however, even after a popup displays
you can still click another button. This is not a desired behavior. Let's come up with a way to stop this. Here's
what I have.

My proposal is to make a layer that will cover up the other buttons while a popup is being displayed. So, go back
into your *ProfileTabMc* symbol and copy the tab–just the tab part, not the buttons. Next, paste that tab into
a new symbol. Call it *TabCoverMc*. One small touch, change the graphic's Alpha property to 80% in fill color
picker.



After you create that symbol, drag it into your *ProfileTabMc* symbol on a new laber called *TabCover.* Be
sure that the layer is above the *Buttons* layer and below the *Popups* layer. Here's the result.



Now you can give the *TabCoverMc* symbol an *Instance Name* of *tabCover_mc*. We also want this symbol
to be hidden on startup. So, let's hide it just like our caption buttons.

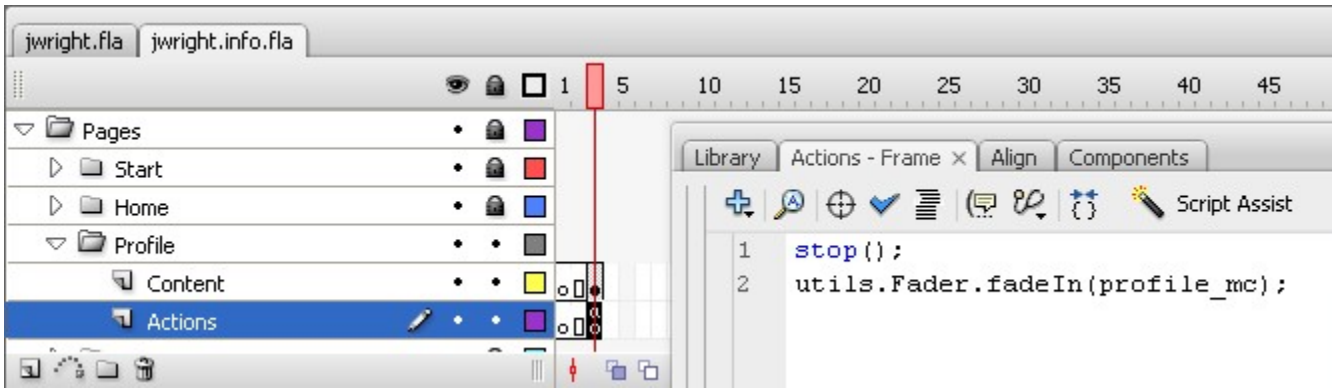
```
tabCover_mc.visible = false;
```

When the user clicks a caption button to see a popup, we can now cover up the other buttons with the new *TabCoverMc*
symbol in our `showPopup` and `closePopup` functions. Here are the modified functions.

```
//////////////////////////////////////////////////////////////////////
// Event Handlers.
//////////////////////////////////////////////////////////////////////
function closePopup(Event:MouseEvent):void
{
    //Fade out the tab cover.
    utils.Fader.fadeOut(tabCover_mc);
```

```
    //If the current popup has been set, fade it out.
    if(this.currentPopup != null)       utils.Fader.fadeOut(currentPopup);
}

function showPopup(Event:MouseEvent):void
{
    //Set the current popup.
    this.setCurrentPopup(Event);

    //Fade in the tab cover.
    utils.Fader.fadeIn(tabCover_mc);

    //If the current popup has been set, fade it in.
    if(this.currentPopup != null)       utils.Fader.fadeIn(currentPopup);
}
```

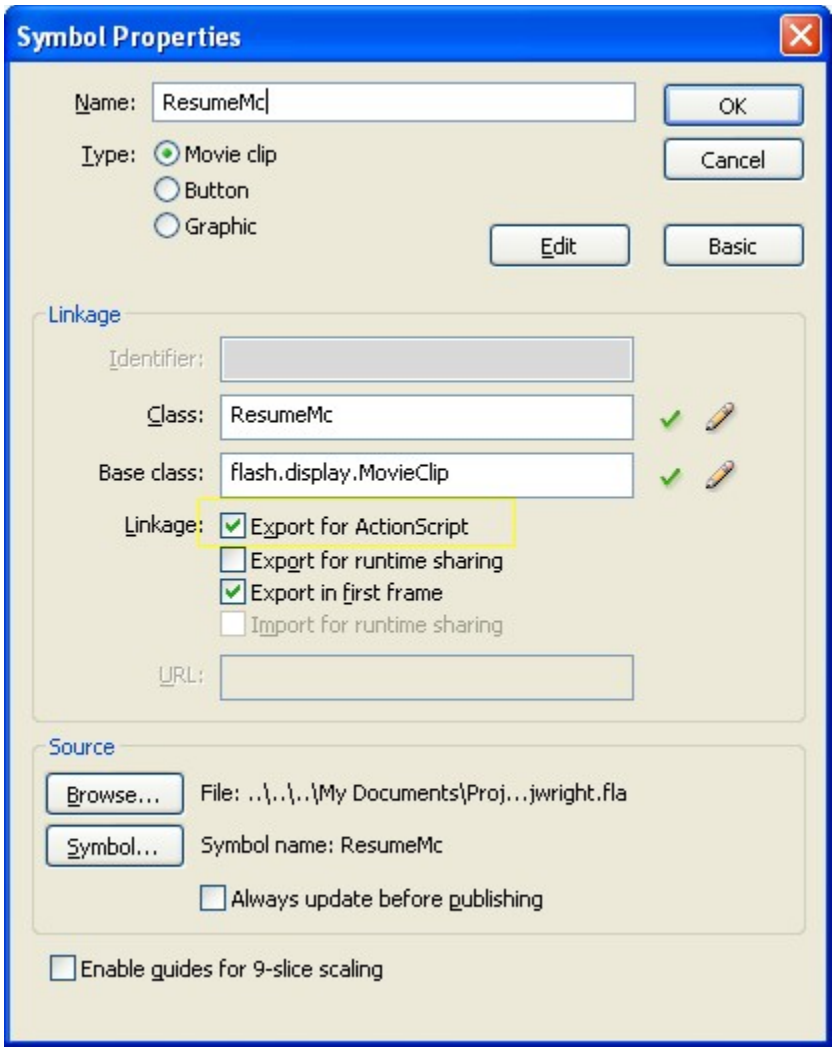
To finish off this section, add a fade effect to the *ProfileMc* symbol in your *Actions* layer. This will fade in your *ProfileMc* symbol when the page is displayed.



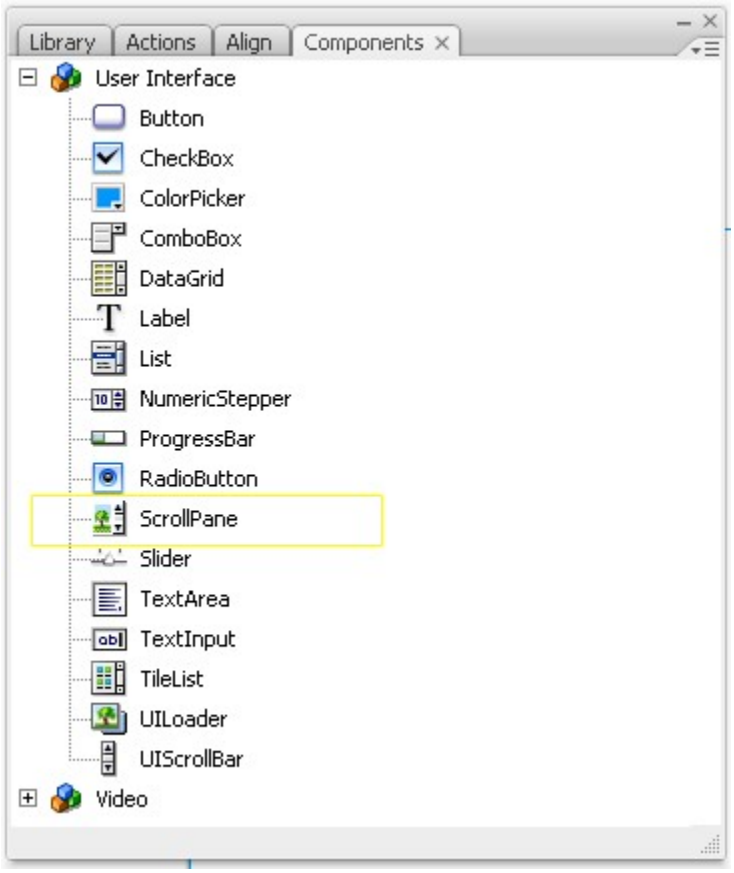
Take a look at the *Profile* page now. It's a beauty! Great job!

The Résumé Page

The *Résumé* page won't take long at all. First create a new Movie Clip call *ResumeMc*.
Inside of the MC, you'll need to either copy and paste my résumé or use your own. One little thing, make sure to check the Check Box marked *Export for ActionScript*.



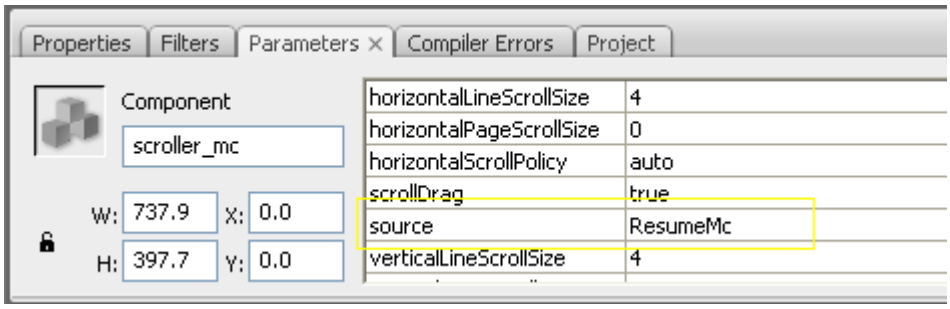
Well that was simple enough. Now, let's use it. Next you'll need to create another Movie Clip called *ResumeScollerMc*. Inside of this MC you'll need a ScrollPane from the Components panel. You can find that under *Window–>Components* (or Ctrl+F7). Once you locate the ScrollPane, drag one onto your *ResumeScrollPaneMc*.



This is a really neat control. Since our résumé is too long to fit on splashy, the ScrollPane component will
scroll our *ResumeMc* symbol up and down. All we need to do is to set a couple of the ScrollPane's parameters to tell it which MC we want it to scroll.

To view the ScrollPane's parameters, select *Window–>Properties–>Parameters*. Then select your ScrollPane
and you should see its parameters listed on the Parameters panel.

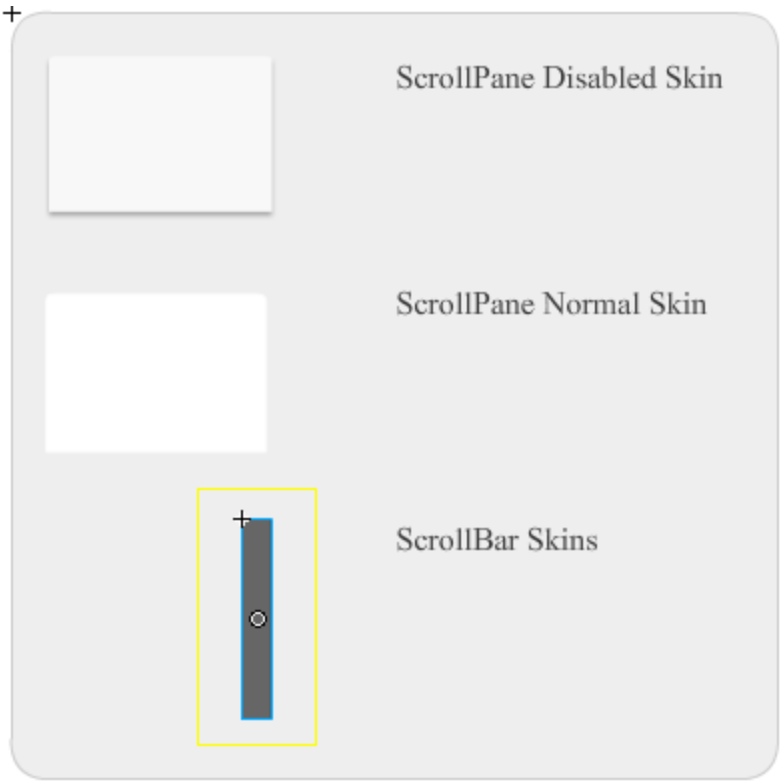
The only parameter you should need to set is the *source* parameter. Set that to the symbol we want to scroll–*ResumeMc*. This is the reason you needed to check *Export for ActionScript* when making the *ResumeMc* symbol. In a nutshell, since we're not adding this directly to our stage, the AS compiler needs
to know what MC to scroll.



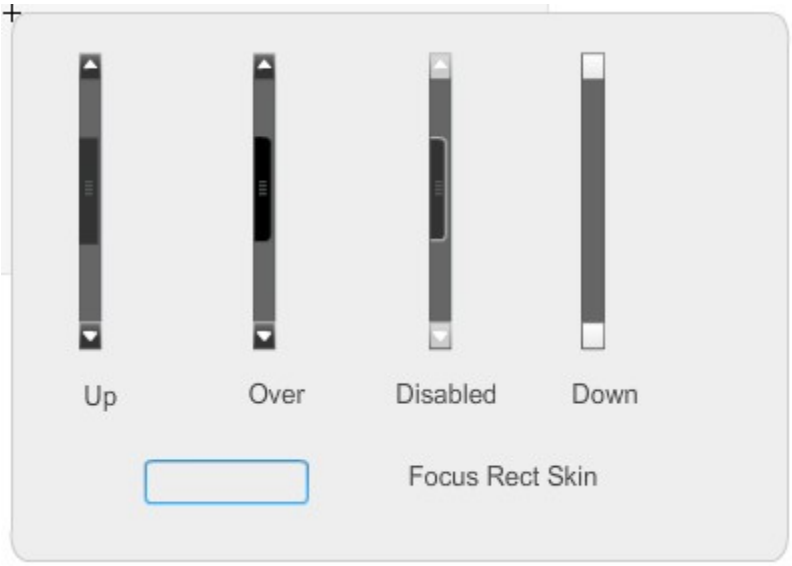
You may now drag the *ResumeScrollerMc* onto the main timeline on the *Content* layer of the *Resume* page. Also, add a fade effect to the *ResumeScrollerMc* symbol in your *Actions* layer. You can do this the same exact way you did the *ProfileMc* symbol's fade effect.

More than likely, you're going to notice that your scrollbar does not look like mine. If that's the case, you can
actually edit the ScrollPane component–just like a Movie Clip. Pretty awesome, huh?! Just locate the ScrollPane
in your Library (probably under *Component Assets*), right click, and choose *Edit*.

You should see something like the following. Double click the ScrollBar Skins symbol.



Now, you can edit any of the skins you see before you. Here's what mine look like after I've tweaked them.



That's it for the *Resume* page! On to the *Contact* page.

The Contact Page

The *Contact* page really does not need much explanation. You've already done this multiple times.

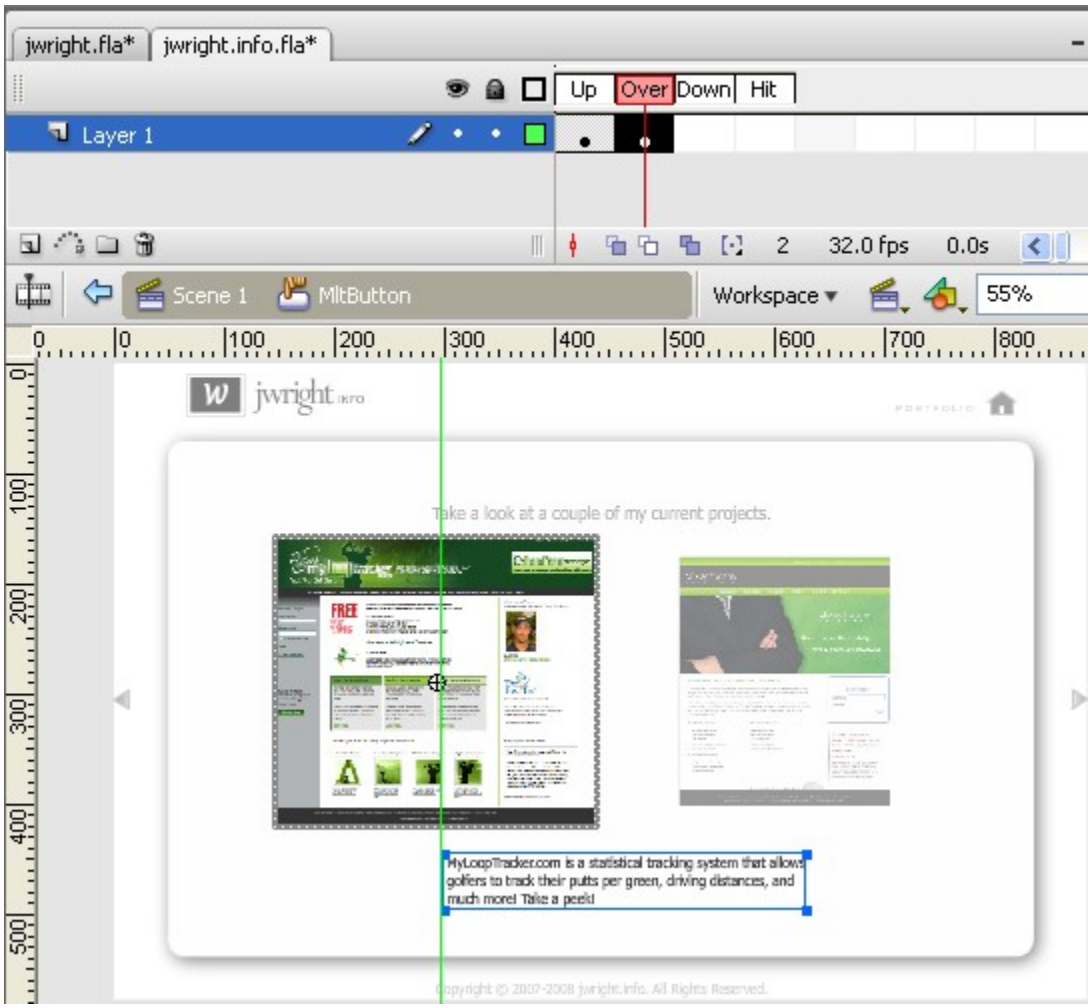
But, just for kicks, here you go. All you need to do is create a new Movie Clip called *ContactMc* that contains your contact information. Simple stuff really. Then you can give it an *Instance Name* and use the `utils.Fader` class to fade in the MC when the *Contact* page's frame is hit.

That's it!

The Portfolio Page

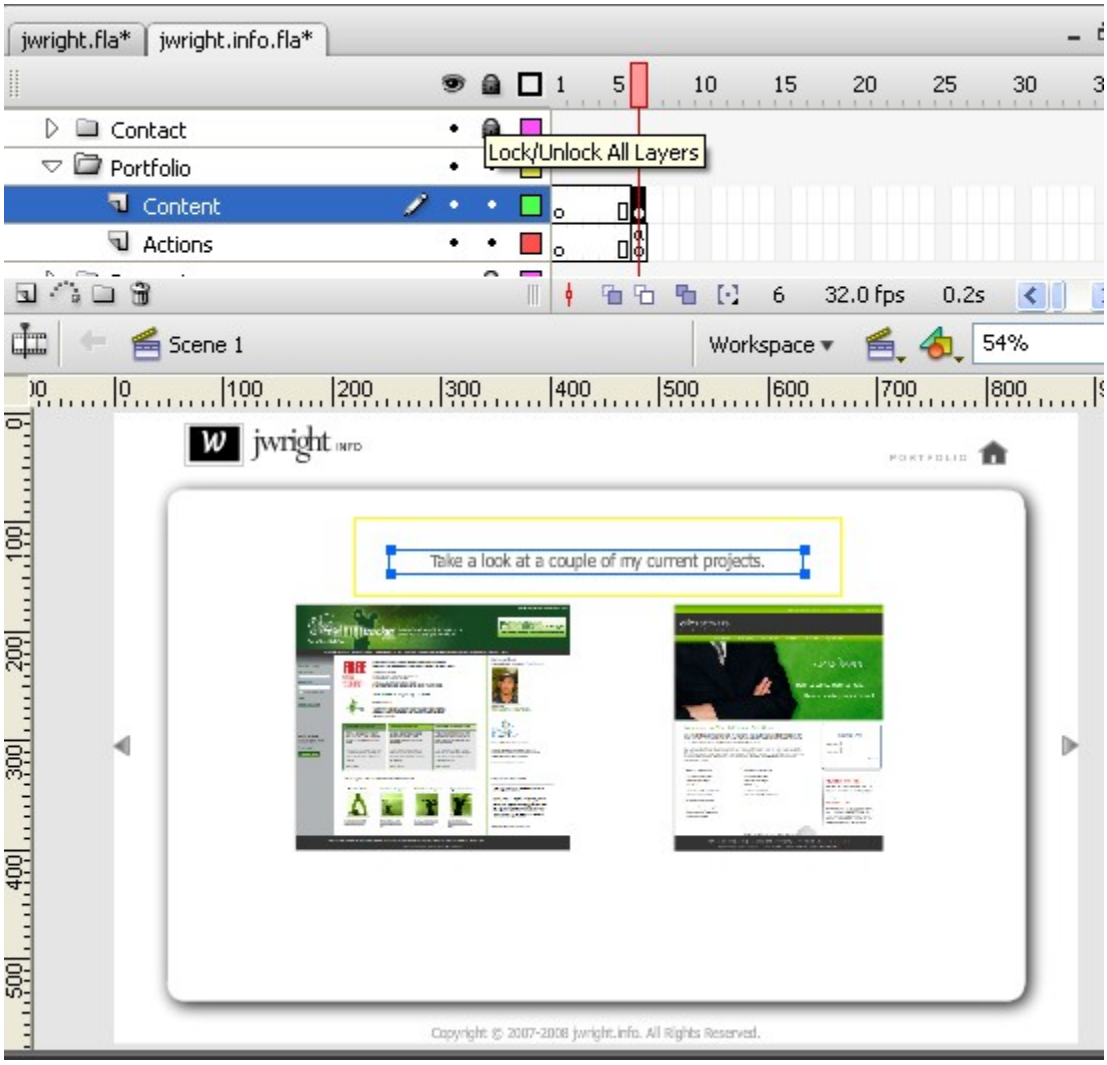
My *Portfolio* page definitely needs some work. But we'll walk through what we have anyway. It's basically just using two buttons that will open a new window.

All I did to create my buttons was take a screen shot of a couple websites I've been working on. Then I imported
them into my Library and dropped them on a button. Then, on the *Over* frame of the button, I made the
image a little bigger and added a caption at the bottom. Here's the first button for reference.



After you create your two buttons, simply drop them onto your main stage.

Next you can add a little heading if you'd like.



Select your three items–the heading and the two buttons. Next, hit F8 and convert those three things into one Movie Clip so we will be able to add our fade effect. I called my new MC *PortfolioMc*.

We're almost done! Let's add a little bit of AS to open up a new browser window when a button is clicked. Double click your *PortfolioMc* symbol to take it into Edit mode. Next, add a new layer called *Actions*. In that layer paste the following code.

```
css_btn.addEventListener(MouseEvent.CLICK, cssClicked);
mlt_btn.addEventListener(MouseEvent.CLICK, mltClicked);

function cssClicked(event:MouseEvent):void
{
    //Create a new URL request.
    var url:URLRequest = new URLRequest("http://www.clixsoftwaresolutions.com/");

    //Open up the URL in a new window.
    navigateToURL(url, "_blank");
}

function mltClicked(event:MouseEvent):void
{
```

```
    //Create a new URL request.
    var url:URLRequest = new URLRequest("http://www.mylooptracker.com/");

    //Open up the URL in a new window.
    navigateToURL(url, "_blank");
}
```

I know what you're thinking. Yes, we could've used another `switch` statement, but for just two buttons
I opted to just use two functions. Good catch though!

Finally, just add your `utils.Fade.fadeIn(portfolio_mc)` function call into the *Portfolio* pages's *Action* layer on the main timeline, and you're done!

Note: If you debug your movie now and try to click one of your buttons on the *Portfolio* page, you will
probably see the following warning. No worries, it's supposed to do that. If you test your movie (Ctrl+Enter) or
deploy your movie–it should work like a charm!

The Respect Page

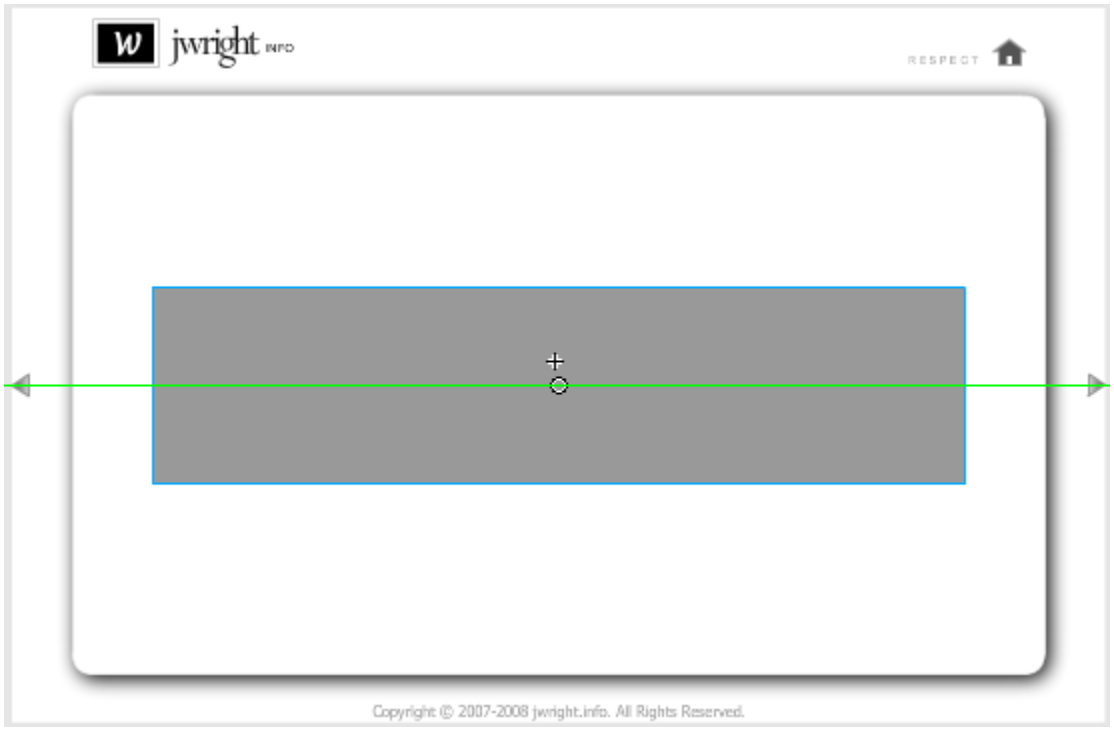
Ahhhhhh, at last. The final page. Well, in my opinion, I've saved the best for last. I love this little scroller,
and the best part is that it's not too hard to achieve this awesome effect.

Start out by making a new Movie Clip called *RespectStageMc*. On the first layer in our MC we want to introduce
something new to jwright.info–masking. You'll see the power of masking real soon.

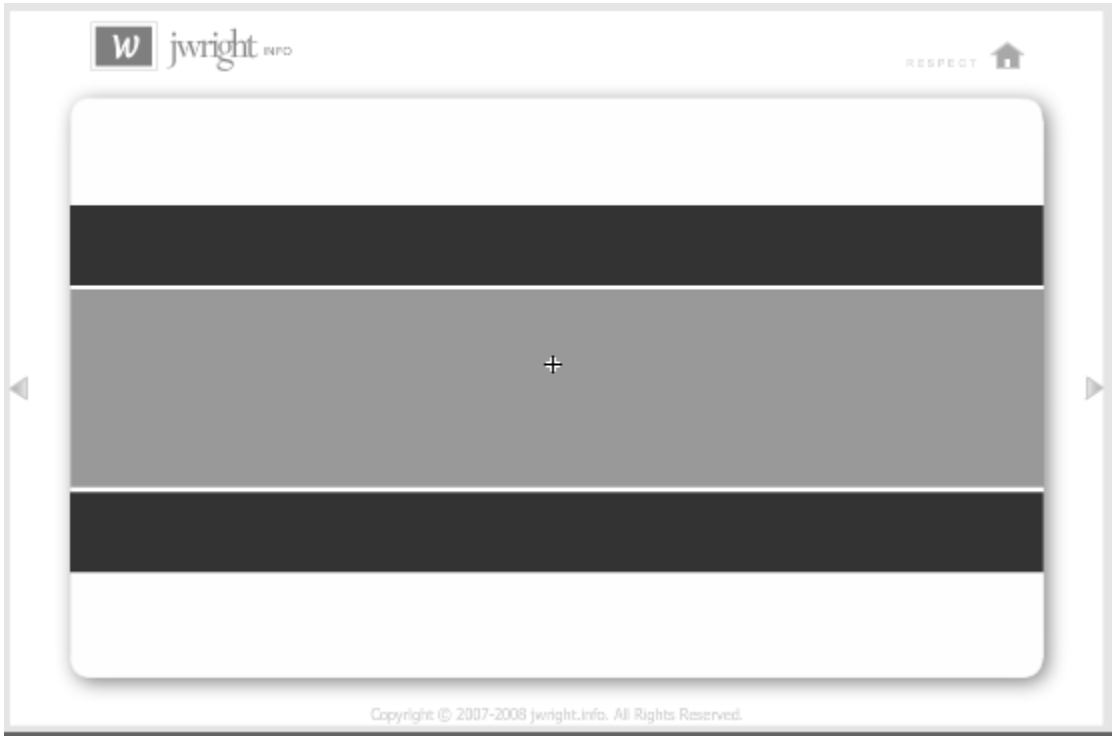
Secondly, place a rectangle centered on the page on a layer called *Scroll Area*. Set its fill color to #999999
with no stroke. Don't worry about size quite yet. Next, go back to your main timeline and add the MC to the *Content*
layer on your *Respect* page.

Now, you can go back into your MC and adjust the size of your rectangle to be the exact width of splashy.

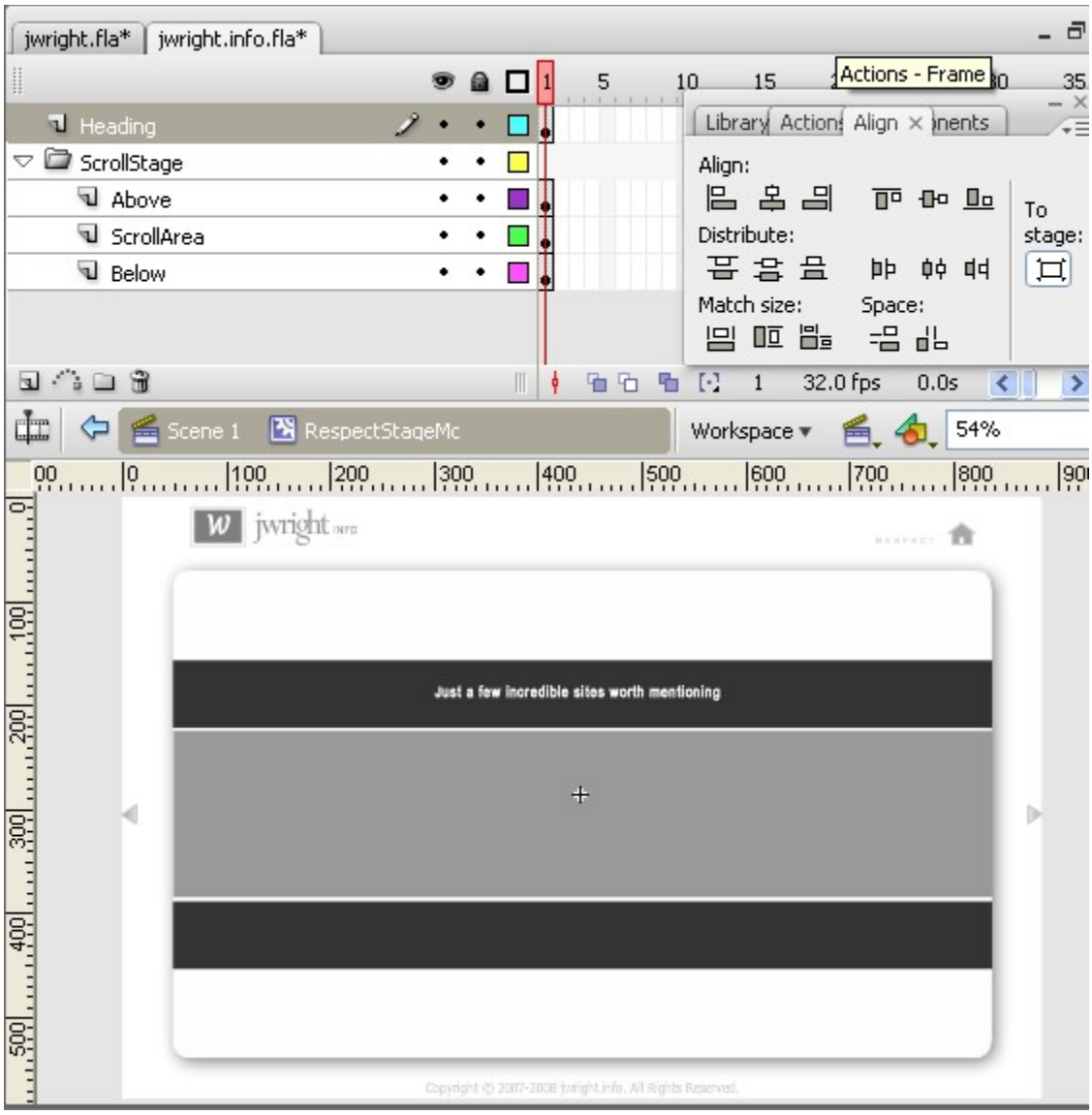
(Remember him?) Make sure that your MC is centered with your navigational arrow buttons before you resize anything.



Let's add a little more to our MC. Let's place a darker rectangle (#333333) above and below our *Scroll Area* layer.



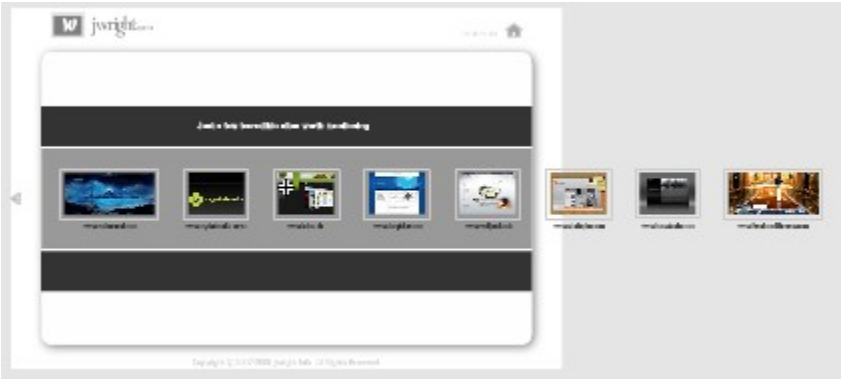
Next, add a heading.



On to the thumbnails! Again, feel free to use the thumbnail buttons I've already made, or you can make your own. The concept
is pretty much like what you did on the *Portfolio* page–just a little smaller. Here's what one of mine looks like.

After you get all of your thumbnails made, place them all centered, and spaced evenly, on a single layer called
Thumbs in your MC.

Next, you'll need to go back into that MC and paste this code. No real need to explain here. We're just popping up a new
browser window and loading the site that was clicked.



```
//////////////////////////////////////////////////////////////////////////
// Includes and Imports.
//////////////////////////////////////////////////////////////////////////
import flash.net.navigateToURL;
import flash.net.URLRequest;

//////////////////////////////////////////////////////////////////////////
// Event Setup.
//////////////////////////////////////////////////////////////////////////
advanced_btn.addEventListener(MouseEvent.CLICK, gotoSite);
henz_btn.addEventListener(MouseEvent.CLICK, gotoSite);
kalou_btn.addEventListener(MouseEvent.CLICK, gotoSite);
studio_btn.addEventListener(MouseEvent.CLICK, gotoSite);
crystal_btn.addEventListener(MouseEvent.CLICK, gotoSite);
deepBlue_btn.addEventListener(MouseEvent.CLICK, gotoSite);
wolf_btn.addEventListener(MouseEvent.CLICK, gotoSite);
freedom_btn.addEventListener(MouseEvent.CLICK, gotoSite);

//////////////////////////////////////////////////////////////////////////
// Event Handlers.
//////////////////////////////////////////////////////////////////////////
function gotoSite(Event:MouseEvent):void
{
    var url:URLRequest = null;
    switch (Event.target)
    {
        case advanced_btn :
            url = new URLRequest("http://www.2advanced.com/");
            break;
        case henz_btn :
            url = new URLRequest("http://www.henzstudio.com/");
            break;
        case kalou_btn :
            url = new URLRequest("http://www.kalou.ch/fr/");
            break;
        case studio_btn :
            url = new URLRequest("http://www.studiopiro.com/index2.htm");
            break;
        case crystal_btn :
            url = new URLRequest("http://www.crystalmedia.co.yu/");
            break;
        case deepBlue_btn :
```

```
            url = new URLRequest("http://www.deepblue.com/");
            break;
        case wolf_btn :
            url = new URLRequest("http://www.wolfpeak.net/");
            break;
        case freedom_btn :
            url = new URLRequest("http://www.freedomoftheseas.com/");
            break;
    }

    if (url != null)
    {
        navigateToURL(url, "_blank");
    }
}
```

After you've done that, select all of your thumbs and convert them into one single Movie Clip called *ThumbsScrollerMc*.
You'll see why in just a minute.

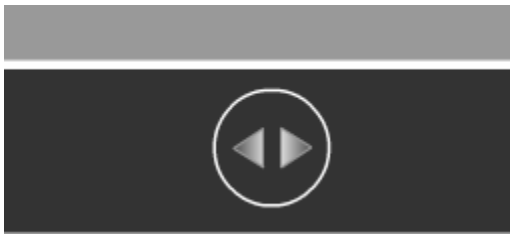
As you can see right now, the thumbnails are running off of the stage to the left. If you test your movie you'll
see them there as well. This is not what we want. We want the thumbnails to disappear as they move off of splashy.

To achieve this effect we need to use a technique called *Masking*. To mask the thumbnails, insert a new layer
above your *Thumbs* layer called *Scroll Mask*. Right click on the new layer and select *Mask*.

On this layer you'll need to create a rectangle the same exact size as the one on your *Scroll Area* layer. May I suggest that
you just copy and paste-in-place that rectangle from the *Scroll Area* layer. You will need to change its fill color alpha value to 10%. This is just so you can see what's behind the mask.

If you test your movie now, you'll notice how the only thumbnails that display are the ones that are directly behind the mask.

Now, the cool part. To be able to scroll left and right, you'll need a left and right arrow. I just used the arrow buttons
we used on our main timeline and placed them on a new layer. I also added a little white border around the arrows.



Here's the code you'll need to scroll your *ThumbScrollerMc* named *thumbs_mc*.

```
//////////////////////////////////////////////////////////////////
// Includes and Imports.
//////////////////////////////////////////////////////////////////
```

```
import fl.transitions.Tween;
import fl.transitions.easing.Regular;

////////////////////////////////////////////////////////////////////
// Event Setup.
////////////////////////////////////////////////////////////////////
left_btn.addEventListener(MouseEvent.CLICK, scrollLeft)
right_btn.addEventListener(MouseEvent.CLICK, scrollRight)

////////////////////////////////////////////////////////////////////
// Event Handlers.
////////////////////////////////////////////////////////////////////
function scrollLeft(Event:MouseEvent):void
{
    //Shift the MC left 150 units.
    var myTween:Tween = new Tween(thumbs_mc, "x", Regular.easeOut, thumbs_mc.x, (th
}

function scrollRight(Event:MouseEvent):void
{
    //Shift the MC right 150 units.
    var myTween:Tween = new Tween(thumbs_mc, "x", Regular.easeOut, thumbs_mc.x, (th
}
```

Essentiall, we are just shifting the *ThumbsScrollerMc* left and right when the user clicks an arrow button. Test it out!

The only thing left to do is to add the fade in effect on the main timeline for the *RespectStageMc* symbol. Great work!

Conclusion

Well congrats fellow Flashians! You have successfully completed the entire jwright.info website!

Feel free to also look at the HTML file included in the package to see how the .swf file is displayed in the
browser.

I know this tutorial was a little long–maybe the longest you've ever done. But the content covered in this
walk through should give you a **great** foundation for developing your own Flash site. Keep up the good work! By the way, if you feel like this is too much work, you can create a free professional flash website with Wix.

Tags: Creating Websites, flash portfolio, flash tutorial, flash web design, flash website

Flash Website Tutorial