

ResizeToolkit Quick Start Guide

Jeff dePascale
www.jeffdePascale.com

ResizeToolkit is a collection of interrelated tools that aim to assist in and simplify full browser window and full screen flash implementations. Two primary classes, ResizeController and FullScreenController, serve as starting points for 'taking charge' of their respective jobs; namely laying out objects on the stage when the stage is resized and handling fullscreen enabling and disabling, respectively.

Let's start with a basic implementation of ResizeController and we'll add on from there:

```
package {  
  
    import flash.display.*;  
    import flash.events.*;  
    import com.jeffdePascale.resizeToolkit.*;  
  
    public class Main extends Sprite {  
  
        private var resizeController:ResizeController;  
  
        public function Main():void {  
            resizeController = new ResizeController();  
            resizeController.addEventListener(Event.RESIZE, eResizeHandler);  
            resizeController.attach(this);  
        }  
  
        private function eResizeHandler(e:Event):void {  
            //layout items here  
        }  
    }  
}
```

The tangible benefits to using the ResizeController as opposed to just listening to the stage resize event are:

- Dispatches an initial call on instantiation to force layout the objects before the stage is ever resized by the user.
- Ability to disable the event.RESIZE dispatch by toggling the 'enabled' property.
- Event.RESIZE can be force dispatched at any time by calling the 'resize()' method.

Here, we simply attach an instance with a reference to the stage. In this case we are in the document class so we have used 'this'. This is the recommended implementation for ResizeController. All of the layout adjustment code goes into the eResizeHandler method.

Now, let's add in some full screen functionality. Note that, as with any flash fullscreen implementation, the embed for the SWF file must have the allowFullScreen parameter set to true. We'll assume we have a movieclip button, called 'toggle_mc', and that movieClip has two frames – the first says 'go to fullscreen' and the second frame says 'back to window'. We'll make that a functional button and change the button frame like this:

```

package {

import flash.display.*;
import flash.events.*;
import com.jeffdePascale.resizeToolkit.*;

public class Main extends Sprite {

    private var resizeController:ResizeController;
    private var fullScreenController:FullScreenController;

    public function Main():void {
        resizeController = new ResizeController();
        resizeController.addEventListener(Event.RESIZE, eResizeHandler);
        resizeController.attach(this);
        fullScreenController = new FullScreenController(this);
        fullScreenController.addEventListener(Event.CHANGE, eFSChangeHandler);
        toggle_mc.addEventListener(MouseEvent.MOUSE_UP, eToggleHandler);
    }

    private function eToggleHandler(e:MouseEvent):void {
        fullScreenController.toggle();
    }

    private function eFSChangeHandler(e:Event):void {
        toggle_mc.gotoAndStop((fullScreenController.displayState == StageDisplayState.NORMAL) ?
1:2);
    }

    private function eResizeHandler(e:Event):void {
        //layout items here
    }
}
}

```

So here we can see we added another reference to 'this', and again, it is highly recommended that this class be used in the document class for the project. We added code to the toggle_mc button to trigger the toggle() method on click, and from there the frame is selected for the movieClip based on the current status of the displayState property when the CHANGE event is dispatched from the FullScreenController instance. If at any point, full screen should be disabled (say, for example, you have an on screen form with text fields, which fail to function by design in Flash 9), the enabled property can be changed. If enabled is set to false and the SWF is in full screen mode, it will be forced back to normal. The setState() method can be used to force a window mode, by passing a string value from the stageDisplayState class.

So there's the basics, now lets add some graphical layout control to this. The trick to an effective full window flash implementation is making assets adjust to the stage at any size. This is where the ResizeShortcut class comes in. ResizeShortcut is a collection of static methods that control positions of displayObject's based on stage dimensions. In the previous example, all of the ResizeShortcut calls would be placed inside the eResizeHandler method. Lets say you have a background that you would like to fill the screen, called bg_mc. Maybe it's a photo, something that should stay proportional. The code would look like this:

```
private function eResizeHandler(e:Event):void {
    ResizeShortcut.fill(bg_mc);
}
```

This will center the clip and stretch it proportionally to fill the stage, clipping the edges if necessary to fill the space while maintaining the proper aspect ratio. Optionally, you can specify to *skew* the proportions so it evenly fits in the space provided, pixel to pixel, so nothing gets clipped. Also, these methods assume that your displayObject is laid out with a registration point at 0,0. If your displayObject has a central registration point, that can be specified in many methods as well to account for this. The ResizeStretchType and RegistrationPoint classes define the options for these parameters. To implement both of these change to the code above, it would look like this:

```
private function eResizeHandler(e:Event):void {
    ResizeShortcut.fill(bg_mc, ResizeStretchType.SKEW, RegistrationPoint.CENTER);
}
```

Remember that toggle_mc clip? Lets lay that out in the upper right corner of the screen at all times. To do that, we use the float method of ResizeShortcut. Float locks a displayObject to either the right edge of the stage, bottom edge of the stage, or both, so when scaling the stage, displayObjects aligned by this method 'float' along the sizing edges as you adjust them. You can optionally specify an offset value as well, to add a margin gap from the edge for the object. Lets say you want a 10 pixel buffer from the right edge, and you want it placed 10 pixels from the top. The code would now be:

```
private function eResizeHandler(e:Event):void {
    ResizeShortcut.fill(bg_mc, ResizeStretchType.SKEW, RegistrationPoint.CENTER);
    ResizeShortcut.float(toggle_mc, ResizeAxis.SET_X, 10);
    toggle_mc.y = 10;
}
```

One more option for this method – a problem with scaling the stage in this manner is that a small enough size, your objects will begin to overlap each other. To counter this, you can specify in the last optional parameter a minimum value to move the item to. If the placement of the object on the stage drops below the minimum value specified in the parameter, it will lock to that parameter and slide off the edge if the stage is reduced further in size.

Note the ResizeAxis class referenced above. ResizeAxis is used in many of the ResizeShortcut methods to specify which axis the method should effect a change to on the displayObject, either X, Y, or both. Lets add in two more – we'll center a clip on the stage called main_mc (the center method also allows axis selection and choosing an upper left or central registration point), and we'll specify that a object called footer_mc should scale to fit the bottom of the screen and float along that edge, and that it should skew itself to maintain the same height but force stretch the width. Now we have:

```
private function eResizeHandler(e:Event):void {
    ResizeShortcut.fill(bg_mc, ResizeStretchType.SKEW, RegistrationPoint.CENTER);
    ResizeShortcut.float(toggle_mc, ResizeAxis.SET_X, 10);
    toggle_mc.y = 10;
    ResizeShortcut.center(main_mc);
    ResizeShortcut.scaleAndCenter(footer_mc, ResizeAxis.SET_X);
    ResizeShortcut.float(footer_mc, ResizeAxis.SET_Y);
}
```

Lastly, lets add in a shadow gradient on the background – lets say you have a dark gray overlay called gray_mc that you want to start at 50 pixels on the y axis, and stretch the remainder of the way to the bottom of the screen, no matter how tall the stage area is, while always filling the width of the stage. For this we will use the scale method for the Y axis, along with another scaleAndCenter reference for the X axis:

```

private function eResizeHandler(e:Event):void {
    ResizeShortcut.fill(bg_mc, ResizeStretchType.SKEW, RegistrationPoint.CENTER);
    ResizeShortcut.float(toggle_mc, ResizeAxis.SET_X, 10);
    toggle_mc.y = 10;
    ResizeShortcut.center(main_mc);
    ResizeShortcut.scaleAndCenter(footer_mc, ResizeAxis.SET_X);
    ResizeShortcut.float(footer_mc, ResizeAxis.SET_Y);
    ResizeShortcut.scaleAndCenter(grey_mc, ResizeAxis.SET_X);
    grey_mc.y = 50;
    ResizeShortcut.float(gray_mc, ResizeAxis.SET_Y);
}

```

Take a look at the documentation for these static methods, many of them have the optional parameters mentioned for axis selection, proportional or skewed stretching, registration point specification, value offsets, and minimum values.

One last thing to look at. Lets say you have a layout that has many items located in a grid like pattern on the stage. Say you have a logo in the upper left (logo_mc), a nav in the upper middle (nav_mc), the full screen button in the upper right (toggle_mc), a content area in the middle of the screen (main_mc), and a footer in the bottom middle (footer_mc). Lets also say we want all of the outer edge objects to be 10 pixels away from their respective edges. There we have portions of a 9 point grid – the four corners, center, and middle edges would make up that grid. The ResizeGrid class is a tool for implementing the ResizeShortcut methods simply when your layout follows this grid pattern. You don't need a full grid of objects to use it, as is evidenced in this situation. The class expects an array of three arrays, each containing three objects. Each sub array represents a row in the grid, and each object represents the position in the grid row. Within that object, you can specify 'target' for the object to position (required), and optionally, 'offsetX' and 'offsetY' values for moving the objects final relative position. The class itself has a parameter called 'margin' that specifies a buffer pixel area around the entire stage. When set to a value, the offset values of an object if specified are relative to the margin, not the edge. So lets go ahead and lay that out in our same eResizeHandler method from before. This time I will include the full code so you can see the imports and declarations:

```

package {

    import flash.display.*;
    import flash.events.*;
    import com.jeffdePascale.resizeToolkit.*;

    public class Main extends Sprite {

        private var resizeController:ResizeController;
        private var grid:ResizeGrid;

        public function Main():void {
            resizeController = new ResizeController();
            resizeController.addEventListener(Event.RESIZE, eResizeHandler);
            resizeController.attach(this);
            grid.margin = 10;
            var gridArray:Array = new Array([
                {target:logo_mc}, {target:nav_mc}, {target:toggle_mc}],
                [{}, {target:main_mc}, {}],
                [{}, {target:footer_mc}, {}]);
            grid = new ResizeGrid(gridArray);
        }

        private function eToggleHandler(e:MouseEvent):void {
            fullScreenController.toggle();
        }

        private function eResizeHandler(e:Event):void {
            grid.resize();
        }
    }
}

```

Granted, this method has its limitations. You don't have access to some of the extra parameters of the ResizeShortcut methods, which ResizeGrid does use to layout the objects. You can't specify when to scale to the stage either. But if multiple items require just simple placement on the grid, this is a fast, one shot method to do it. All that is needed after specifying the array and passing it into the constructor is to add a resize() call for the instance of the class inside your resize handler, the ResizeGrid class does the rest of the work. Note the visual layout of the array of objects – I find this to be very helpful when laying out objects using this class, especially if someone else is looking at your code. You wind up with a one to one relationship between what the screen displays and what your code looks like for the grid array.

Be sure to check the documentation for this entire package, and using FDT, FlashDevelop, or FlexBuilder with this package is definitely recommended for the code hinting, it saves tons of time and makes these classes both simple to implement and useful. Hopefully you find them useful as well! visit www.jeffdepascale.com for the latest updates on this open source package, and please feel free to contact me with any suggestions, comments or bugs. Happy coding!

-Jeff dePascale
www.jeffdePascale.com