FORUM NOKIA

Flash Lite 2.0: Screen Saver and Wallpaper

Version 1.0; June 27, 2008

Flash Lite 2.0

NOKIA

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this document at any time, without notice.

License

A license is hereby granted to download and print a copy of this document for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1 **Introduction** 5
2 **Flash Lite 2.0 standby screen animation support** 6
3 **Example 1: Classic bouncing animation screen saver** 7
3.1 Planning the project 7
3.2 Creating a new ActionScript class file 7
3.3 Creating the BouncingAnimation.as class 8
3.3.1 Specifying the canvas movieclip 8
3.3.2 Defining the boundaries of the screen 9
3.3.3 Creating the wall movieclips 10
3.3.4 Attaching the moving object 11
3.3.5 Initiating the animation 12
3.3.6 Moving the object 13
3.3.7 Detecting a collision 14
3.3.8 Setting the animation speed 15
3.4 Configuring an animation 16
4 **Example 2: Scaling the size of the moving object** 17
4.1 Setting the font size 18
4.1.1 Using the TextField autoSize property 18
4.1.2 Using the TextFormat size property 18
4.1.3 Repositioning the wall due to dynamic text margin 19
5 **Example 3: Adding visual effects** 21
5.1 Tiling a texture across the stage 21
5.2 Creating a tiled background layer 23
5.3 Creating a tiled masked layer 24
5.3.1 Understanding dynamic masks 25
5.4 Randomizing the text 26
6 **Summary** 27
7 **About the author** 28
8 **Evaluate this resource** 29

Change history

June 27, 2008	Version 1.0	Initial document release.

1 Introduction

In addition to being an effective platform for casual games and data driven applications, Adobe Flash Lite for Nokia devices supports compelling personalization content like animated dynamic screen savers and wallpapers. The Flash Lite platform brings to Nokia devices a sophisticated animation engine that supports a mix of vector and bitmap animation in screen saver/wallpaper content. Furthermore, Flash Lite screen saver/wallpaper applications can execute ActionScript to dynamically adjust the visual content of the animation depending on the date or the time of day, or completely randomly, enabling entertaining content that is dynamic and ever changing.

This document explores the different features of Flash Lite 2.0 for creating dynamic screen saver and wallpaper animation. Some of the concepts you will learn include using ActionScript to adjust the visual content to fit the stage, and creating visual interest with randomness and the application of a dynamic mask.

2 Flash Lite 2.0 standby screen animation support

Nokia Series 40 3rd Edition, Feature Pack 2 devices support Flash Lite 2.0 content for standby screen screen saver and wallpaper animation. Nokia S60 3rd Edition, Feature Pack 1 devices support Flash Lite 2.0 content for standby screen saver animation only.

Note: Nokia S60 and Series 40 phones require a SIM card with an active account to display Flash Lite screen savers and play timeline animation in wallpapers. Nokia devices will play ActionScript controlled animation for wallpapers without an active SIM card.

The Nokia 6290 S60 3rd Edition, FP1 phone includes a Flash Lite 2.0 standalone player but does not support SWF animation for screen savers.

3 Example 1: Classic bouncing animation screen saver

In the first example you will learn how to create the classic “bouncing” animation screen saver where an object moves across the screen and rebounds off of the edge of each side of the screen.

Since this is a common form of screen saver, you will develop a simple ActionScript 2 Class to streamline the process of creating a screen saver. The advantage of using a class is that it reduces the amount of time and code required to create a new screen saver. Once you have completed the initial work to develop the class, you simply import the class into the ActionScript code of your FLA, add a few lines of ActionScript to customize the screen saver, and your screen saver is ready for testing.

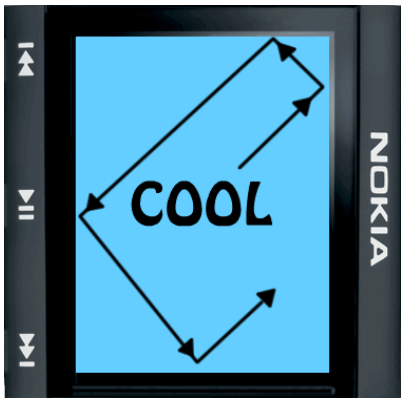

Figure 1: Example 1 bouncing animation

3.1 Planning the project

Before you begin working on the ActionScript for the class, it is a good idea to take an inventory of what your class needs to do. The most basic form of this animation consists of the following:

1. Specify a canvas movieclip on which to build the screen saver.
2. Define the boundaries of the screen saver.
3. Attach the moving object to the canvas movieclip.
4. Move the object on the screen.
5. Detect a collision between the object and the edge of the screen.
6. Determine the new direction for the object to travel based on the edge it collides with.
7. Set the speed of the moving object.

3.2 Creating a new ActionScript class file

An ActionScript Class file is a separate document from the FLA of your screen saver. The class file contains the ActionScript for the class. Creating an ActionScript Class file in Flash CS3 is a four-step process:

1. Open the **File** menu and select **New** to open the New Document dialog box.
2. Select **ActionScript File** from the menu in the New Document dialog box.

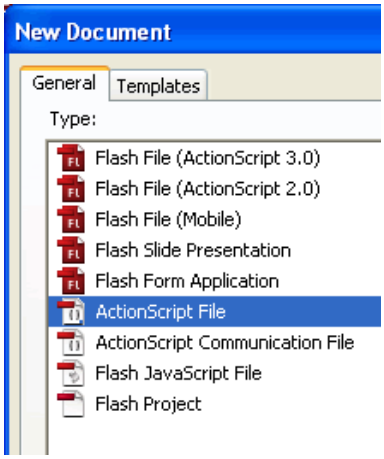

Figure 2: New Document dialog box

3. Click **OK**.
4. Save the ActionScript file as *BouncingAnimation.as* in the same folder as your screen saver FLA.

Tip: It is a common convention for the file name and the ActionScript name of the class to be the same.

Flash CS3 opens the new ActionScript file in a separate window from your FLA. Create the ActionScript for the class in this window. When you work with class files in Flash CS3 you will likely be switching between the file containing the ActionScript for the class, and the ActionScript window of your FLA listing code specific to your screen saver. You may find it useful to use the Flash CS3 Window menu to switch between the two documents.

At this point, you should open the *BouncingAnimation.as* file from the files included in the article download to follow how the class file is structured. The *BouncingAnimation.as* file is located in the folder named *example 1*.

3.3 Creating the BouncingAnimation.as class

In the *BouncingAnimation.as* document create a class declaration, which is the name of the class. Add the properties and functions, also called methods, for your class inside the curly brackets.

```
class BouncingAnimation{
   // place members of the class within the curly braces
}
```

Note: Normally you would define properties at the beginning of the class and methods after the properties. To make the examples easier to follow, the code for the class will appear in the order as it is described in the article.

Note: An in depth discussion of ActionScript object oriented programming is beyond the scope of this article and not necessary for the simple examples included in the article. For more information, refer to ActionScript Classes in the Flash Help documentation.

3.3.1 Specifying the canvas movieclip

The first step in building the screen saver is defining a movieclip to act as a canvas that holds all of the visual assets for the animation. Consequently, your class needs a way to know what the canvas will be.

When you create a class you often define a constructor function. This is the function that creates an instance of your class. You must call this function from your FLA ActionScript before you can create or configure the screen saver. Since this is the first step in creating your BouncingAnimation instance, set up the class constructor function to expect a movieclip object and make this the canvas for building the screen saver.

Note: A constructor function's name must be exactly the same as the class declaration name.

```
/*
constructor function
*/
public function BouncingAnimation(mco:MovieClip){
    this.canvas_mc = mco;
    this.drawWalls();
}
```

The BouncingAnimation constructor function expects a movieclip object and assigns it to the `canvas_mc` property. This property represents the movieclip upon which Flash Lite builds all assets for your screen saver animation. For example, if you create your instance on the `_root` timeline and pass the keyword `this` as the movieclip object, then Flash Lite will build your screen saver on the `_root` timeline.

```
/*
ActionScript in the root timline of your screen saver FLA.
Create an instance of the BouncingAnimation class.
Pass the keyword "this" to the function to set the current timeline as
the canvas for the screen saver
*/
var cool:BouncingAnimation = new BouncingAnimation(this);
```

Because the constructor function refers to the `canvas_mc` property you also need to define this property in the class.

```
/*
canvas movieclip to draw and attach all animation assets
*/
private var canvas_mc:MovieClip;
```

The code above defines the `canvas_mc` property and sets its data type to a moveclip. This property will represent a container movieclip to hold all assets for the screen saver. Make this property `private` to protect it from being inadvertently altered by code running outside of the class.

Note: Public functions and properties can be called from ActionScript in your screen saver FLA. Private functions and properties can only be called by code within the class declaration braces. The `private` designation prevents outside ActionScript from inadvertently changing a value.

3.3.2 Defining the boundaries of the screen

The second step in building the screen saver is defining the boundaries of the screen. Add this capability also to the constructor function, so that when you call the constructor function to create the instance from your FLA Actionscript, it automatically defines both the canvas movieclip and the boundaries for the screen saver.

```
/*
constructor function
*/
public function BouncingAnimation(mco:MovieClip){
    this.canvas_mc = mco;
    this.drawWalls();
}
```

The `drawWalls()` function creates movieclips along each edge of the screen.

One approach for detecting when the moving object collides with an edge of the screen is to use the Flash Lite 2.0 `hitTest()` function. The `hitTest()` function determines whether two movieclips intersect. To make use of `hitTest()` collision detection you first need to create movieclips along each edge of the screen.

3.3.3 Creating the wall movieclips

The `drawWalls()` function begins the process of creating movieclips, that is, "walls", along each edge of the screen by establishing the `x` and `y` corners for each wall and passing these coordinates to the `drawWall()` function.

```
/*
draw four walls
*/
public function drawWalls(){
   // create objects representing corners of stage
   var topleftcorner:Object = {xpos:0,ypos:0};
   var toprightcorner:Object = {xpos:Stage.width,ypos:0};
   var bottomleftcorner:Object = {xpos:0,ypos:Stage.height};
   var bottomrightcorner:Object = {xpos:Stage.width,ypos:Stage.height};

   // draw 1 wall for each side of the stage
   this.topwall = this.drawWall("topwall",topleftcorner,toprightcorner);
this.bottomwall =
this.drawWall("bottomwall",bottomleftcorner,bottomrightcorner);
   this.leftwall =
this.drawWall("leftwall",topleftcorner,bottomleftcorner);
   this.rightwall =
this.drawWall("rightwall",toprightcorner,bottomrightcorner);
}
```

Determine the corners of the walls by using the `Stage` class `width` and `height` properties. Flash Lite 2.0 cannot detect the actual screen size of a device. However, it can read the size of the stage, using the `Stage.width` and `Stage.height` properties, as specified in the FLA before publishing to SWF.

The `drawWalls()` function sets out the points for each wall and calls the `drawWall()` function four times, once for each wall. Assign the movieclip returned from the drawWall() function to a property representing each wall of the screen saver. Since you have defined four new properties in this function, you must also declare these properties in your class.

```
/*
movieclips representing walls
*/
public var topwall:MovieClip;
public var bottomwall:MovieClip;
public var leftwall:MovieClip;
public var rightwall:MovieClip;
```

Make these properties public in case you need to reposition the walls from ActionScript in your FLA.

The `drawWall()` function creates a wall as a transparent 1-pixel line movieclip along the specified edge of the stage. This function expects a name for the wall, and a start and end point. The start and end point variables are objects with two properties; one for the point's `x` coordinate and one for the `y` coordinate.

```
/*
draw a movieclip containing a single line
```

```
position movieclip along specified side of the stage
*/
public function
drawWall(wallname:String,startpoint:Object,endpoint:Object):MovieClip{
   // create a movieclip for the wall
   var wall:MovieClip =
this.canvas_mc.createEmptyMovieClip(wallname,this.canvas_mc.getNextHighe
stDepth());

   // draw a single line from start point to end point
   wall.beginFill(0x000000,0); // 0 _alpha creates transparent fill
   wall.lineStyle(0,0xFF0000,0); // hairline thickness, 0 _alpha creates
transparent line
   wall.moveTo(startpoint.xpos,startpoint.ypos);
   wall.lineTo(endpoint.xpos,endpoint.ypos);
   wall.endFill();

   return wall;
}
```

This function calls the MovieClip object `createEmptyMovie()` method to create a movieclip for a wall. It places the new movieclip on the `canvas_mc` and uses the Flash Lite 2.0 Drawing API to draw a 1 pixel sized transparent line into the movieclip. Flash Lite automatically positions the movieclip on the canvas movieclip depending on the `x` and `y` position of the `moveTo()` Drawing API method.

To use the Flash Lite 2.0 Drawing API first call the `beginFill()` function to establish the fill color and opacity of the movieclip. In this case you create a transparent fill by passing 0 as the `_alpha` channel value for the fill. Next, define a 1-pixel transparent line style from the `lineStyle()` method. Use the `moveTo()` method to define the starting point for drawing the movieclip and the `lineTo()` method to actually draw a line from the starting point to the ending point. Finally, call the `endFill()` method to complete the drawing. These five lines of code create the movieclip along the specified edge of the screen.

An advantage of generating the wall movieclips with ActionScript is that the boundaries of your screen saver adapt to the size of the SWF's stage. When you publish your screen saver SWF to a different phone screen size, you do not need to manually adjust the size of the wall movieclips. Flash Lite automatically builds the screen saver wall movieclips to the size of the stage you specify in the FLA. This speeds up production time because all you need to do is publish your screen saver SWF to the various screen sizes of your target phones. When the SWF runs as a screen saver or wallpaper on the phone, Flash Lite automatically draws the wall movieclips to the screen size of the phone.

3.3.4 Attaching the moving object

Now that you have defined the canvas movieclip and drawn the four movieclip walls to make the boundaries of the screen saver, you can set up the process for attaching the moving object to the screen saver canvas.

First you need to create a movieclip to represent the moving object. For Example 1 there is a movieclip asset named "textmc" in the library of *bouncinganimation.fla*. The textmc movieclip consists of a static text box with the word "COOL" set to bold style at size 50. The registration point in the center of the text box and positioned at 0,0.

Create a function in the class file called `setMovingOjbect()` that attaches the textmc movieclip asset to the screen saver canvas movieclip. To dynamically attach a movieclip asset from the library to the stage using ActionScript, you must assign an ActionScript identifier to the asset.

Open the linkage properties window for the asset by selecting the asset and choosing linkage from the library menu. Enter a linkage identifier, which must be a string with no spaces, and check "Export for ActionScript".

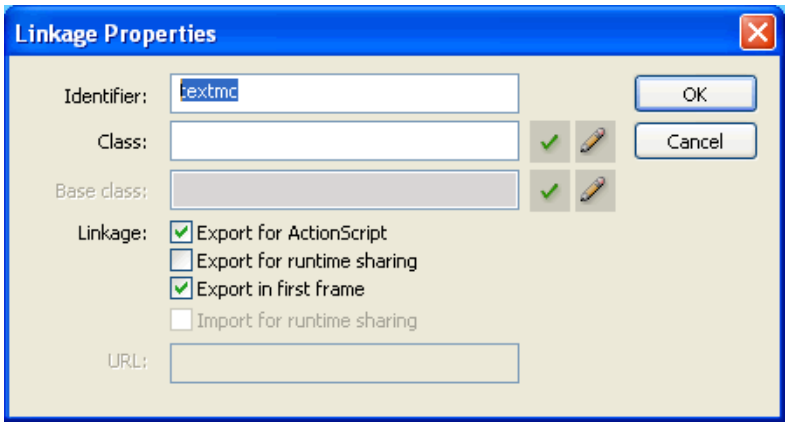

Figure 3: Linkage Properties dialog box

In your class file create the `setMovingOjbect()` function. This function expects the linkage identifier for the asset, and a starting x and y coordinate.

```
/*
assign movieclip of the moving object to the instance
startx and starty are starting corordinates of for the moving object
*/
public function setMovingObject(asset,startx:Number,starty:Number){
   // attach movieclip asset from library onto canvas_mc
   this.movingobject =
this.canvas_mc.attachMovie(asset,"movingobjectmc",this.canvas_mc.getNext
HighestDepth());

   // position moving object
   this.movingobject._x = startx;
   this.movingobject._y = starty;
}
```

The `setMovingOjbect()` function uses the `MovieClip` object `attachMovie()` method to attach the movieclip asset to the canvas movieclip at runtime and positions it in the specified starting point by assigning the movieclip `_x` and `_y` properties to the `startx` and `starty` variables. Since you created a new property in this function to represent the moving object, you must declare this property in the class.

```
/*
movieclip representing the moving object that bounces off of walls
*/
public var movingobject:MovieClip;
```

The `movingobject` property represents the movieclip for the moving object of the screen saver. Make this property public so you can access it from code in your screen saver FLA.

3.3.5 Initiating the animation

Now you are ready to create the code to move the object on the screen. In the class add a new function named `startAnimation()` that initiates the animation.

```
/*
starts the animation by repeatedly calling moveObject from canvas_mc
onEnterFrame loop
*/
public function startAnimation(){
```

```
    /*
    randomly determine the starting vertical and horizontal direction
    */
    this.xdir = (Math.random() >= 0.5) ? 1: -1; // random x direction
    this.ydir = (Math.random() >= 0.5) ? 1: -1; // random y direction

    /*
    start the animation by repeatedly calling moveObject from
onEnterFrame loop
    scope moveObject function to the BouncingAnimation instance instead
of canvas_mc
    */
    this.canvas_mc.onEnterFrame = Delegate.create(this, moveObject);
}
```

First, the function randomly determines the direction for the moving object to travel. It uses the `Math.random()` method to generate a random value between 0 and 0.9. If the value is greater than 0.5, assign the `xdir` and `ydir` properties to + 1, otherwise assign them to –1.

Next, the function assigns the `moveObject()` function to the canvas movieclip `onEnterFrame` event. The `onEnterFrame` event executes the `moveObject()` function at the frame rate specified in your screen saver FLA.

Normally you assign the `moveObject()` function directly to the `onEnterFrame` event as an object literal. However, in this case the keyword `this` changes its scope from referring to your instance of the `BouncingAnimation`, to referring to the `canvas_mc` movieclip, causing unexpected behavior in your code.

Instead, use the `Delegate.create` method of the `Delegate` class to rescope the keyword `this` to your instance of the `BouncingAnimation` class. In the code at the top of your class file before the `BouncingAnimation` class declaration, add the following line of code to import the `Delegate` class, so your code can use the `Delegate.create` method:

```
import mx.utils.Delegate; // required to use Delegate
```

Flash CS3 automatically includes the appropriate ActionScript for the `Delegate` class in your FLA when you publish the SWF.

Since you created the `xdir` and `ydir` properties within the `startAnimation()` function you must also define these in the class. These properties are private and should only be accessible to functions within the class.

```
/*
control animation direction
xdir = 1, right, xdir = -1, left
ydir = 1,down, ydir = -1, up
*/
private var xdir:Number;
private var ydir:Number;
```

3.3.6 Moving the object

The `moveObject()` function changes the position of the moving object each time it is called. Flash Lite repeatedly calls the `moveObject()` function from the `onEnterFrame` loop at the speed of the frame rate. This causes the moving object to move across the screen.

```
/*
change position of the movingobject
```

```
repeatedly called from canvas_mc onEnterFrame loop
*/
private function moveObject(){
   // change movingobject position
   this.movingobject._x += this.speed * this.xdir;
   this.movingobject._y += this.speed * this.ydir;

   // check for collision with left and right walls
   if(this.movingobject.hitTest(this.rightwall) ||
this.movingobject.hitTest(this.leftwall)){
      this.xdir = -this.xdir; // change x direction
   }

// check for collision with top and bottom walls
   if(this.movingobject.hitTest(this.topwall) ||
this.movingobject.hitTest(this.bottomwall)){
      this.ydir = -this.ydir; // change y direction
   }

}
```

The `moveObject()` function determines the new position for the moving object and assigns the new values to the `_x` and `_y` movieclip properties on each frame of the `onEnterFrameLoop` causing the moving object to move on a diagonal line. The movieclip will continue to move along the same diagonal line until it reaches an edge of the screen.

The `xdir` and `ydir` values determine the direction of the moving object. A +1 value for `xdir` moves the moving object to the right, and a –1 value moves the object to the left. A + 1 value for `ydir` property moves the moving object down, and a –1 value moves the object up.

The `speed` property determines distance in pixels to move the object. A higher speed causes the moving object to move a larger distance in pixels on each frame of the `onEnterFrame` loop. Increasing the FLA frame rate also creates a sense of a faster moving animation because Flash Lite is calling the `moveObject()` function more frequently which in return moves the object across the screen more quickly.

Tip: For the smoothest quality of animation, choose the fastest frame rate and a smaller speed setting. You may need to experiment to find the most CPU friendly combination.

3.3.7 Detecting a collision

The `moveObject()` function also checks for a collision using the `hitTest()` function. To check for a `hitTest()` you pass the movieclip object representing a given wall to the `hitTest()` method of the moving object movieclip. If `hitTest()` returns true then the moving object has collided with the specified wall.

```
private function moveObject(){
   // change movingobject position
   this.movingobject._x += this.speed * this.xdir;
      this.movingobject._y += this.speed * this.ydir;

   // check for collision with left and right walls
   if(this.movingobject.hitTest(this.rightwall) ||
      this.movingobject.hitTest(this.leftwall)){
      this.xdir = -this.xdir; // change x direction
   }

   // check for collision with top and bottom walls
   if(this.movingobject.hitTest(this.topwall) ||
```

```
            this.movingobject.hitTest(this.bottomwall)){
            this.ydir = -this.ydir; // change y direction
        }
    }
```

If one of the `hitTest()` checks returns `true`, that is, when the moving object reaches the edge of the screen, the function changes the value of either the `xdir` or `ydir` properties to +1 or – 1 depending on the wall the moving object collides with.

For example, if the moving object is traveling along a diagonal line to the right and down and it collides with the right wall then the `moveObject()` function changes the `xdir` to –1 so that the moving object changes direction to the left, that is, bounces to the left, but still continues in a downward direction. Setting `xdir` to –1 reduces the value of `_x` causing the moving object to move towards the left. The value of +1 for the ydir increases the `_y` value causing the object to continue to move downward.

In some cases the moving object will bounce in the corner colliding with two walls simultaneously, so it is important to check for a collision with all walls on each frame to prevent the moving object from moving in an unintended manner.

3.3.8 Setting the animation speed

You need to configure the speed property from ActionScript in your FLA, which requires that the value of the `speed` property be both readable and writeable from ActionScript. Use special `getter` and `setter` functions to read and write values to the `speed` property.

In your class create a private property to store the `speed` value.

```
/*
read/write speed property, use get/set
rate of movement per frame in pixels
*/
private var __speed:Number;
```

Use the double underscore naming convention for this property and make it private as a precaution to avoid ActionScript variable naming confusion.

To make a property readable create a `getter` function.

```
/*
get value of speed property
*/
public function get speed(){
    return this.__speed;
}
```

To make a property writable create a `setter` function.

```
/*
set value of speed property
*/
public function set speed(value:Number){
    this.__speed = value;
}
```

The name of these functions actually represents the `speed` property. When you get or set the value of `speed` in your FLA ActionScript you access `speed` as a property of your instance, not as a function. Flash Lite will know to execute the `get` or `set speed()` functions depending on the context of your code. You do not need to include the `()`.

```
// FLA ActionScript
// set the speed property
// calls set speed() function in the class
cool.speed = 6;

// trace the value of the speed property
// calls get speed() function in the class
trace(cool.speed) // prints "6" in the output window
```

3.4 Configuring an animation

Now you have finished creating the BouncingAnimation class. Next, set up the FLA ActionScript to configure your screen saver. Open the *bouncinganimation.fla* from the *example 1* folder included in the article download. The ActionScript from frame 1 related to configuring the screen saver is reproduced below. As you can see, once you have created the class there is not much code required to create a new screen saver animation.

```
import BouncingAnimation.as;

// create a new instance of BouncingAnimation, pass a movieclip
reference to be the canvas
var cool:BouncingAnimation = new BouncingAnimation(this); // _root is
the canvas for this instance

// set speed, the number of pixels to change per frame
cool.speed = 4;
// attach a movieclip asset and position it in the center of the stage
cool.setMovingObject("textmc",Stage.width/2,Stage.height/2);

// finished building screen saver, now play the animation
cool.startAnimation();
```

The first step is to use the `import` statement to import the class. Make sure the class file is in the same folder as your screen saver FLA. Then create an instance of the class and pass the canvas movieclip to the constructor function. In this case you will use the `_root` timeline as the canvas. Next, set the `speed` and attach the moving object at the center of the canvas. Finally, start the animation. You can now test your animation in the Adobe Device Central emulator.

4 Example 2: Scaling the size of the moving object

In the first example you used ActionScript to draw wall movieclips according to the size of the stage so that the content adapts to the size of the screen. In example 2 you will extend this concept to adapt the size of the moving object according to the size of the stage. Nokia Flash Lite 2.0 devices support at least two screen sizes; 128 x 160 and 240 x 320. The word "COOL" set at 50 point font size in example 1 works well for 240 x 320 devices but seems too large in devices with a 128 x 160 screen.

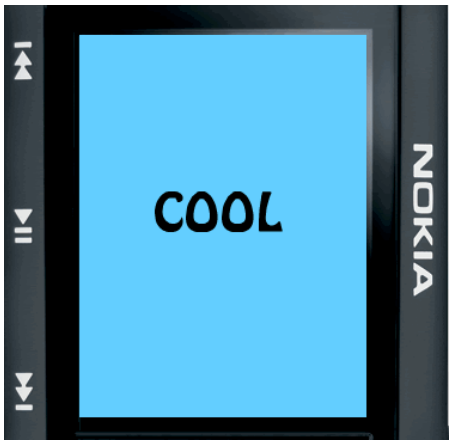

Figure 4: 240 x 320 screen COOL set to 40 pt

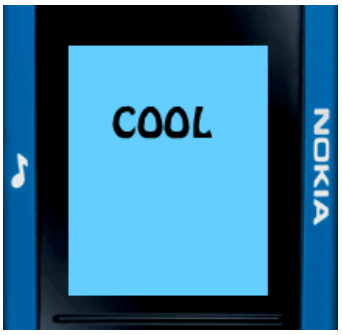

Figure 5: 128 x 160 screen COOL set to 24 pt

In example 2 you will modify your FLA ActionScript to change the font size depending upon the size of the stage, so the word "COOL" occupies approximately the same percentage of the screen regardless of screen size. Using ActionScript to dynamically adjust the size of the moving object offers a more consistent experience across device screen sizes and also streamlines the screen saver production process.

Open the *bouncinganimation.fla* from the example 2 folder included in the article download. To dynamically change the font size of the moving object, you need to change the textfield in the textmc movieclip to a dynamic text box. Open the textmc movieclip, change its textfield type to dynamic, and give it the instance name `msg` in the property inspector.

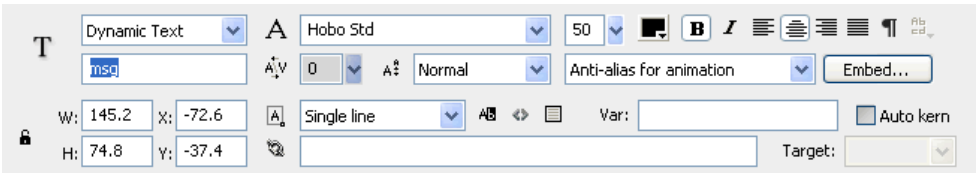

Figure 6: Property Inspector settings for msg dynamic textfield

4.1 Setting the font size

Flash Lite defaults to setting the size of the `msg` TextField based on the width and height values in the property inspector, which are approximately 145 x 74 pixels, and scales the dimensions of the movingobject movieclip according to the TextField size.

Changing the font size of the `msg` TextField does not change its physical dimensions. While the text appears smaller, the box containing the text and the movingobject movieclip are sized according to the width and height of the TextField, not the text. If you scale the text to a smaller font size, there will be extra space around the text causing the appearance that the moving object bounces before the text reaches the edge of the screen.

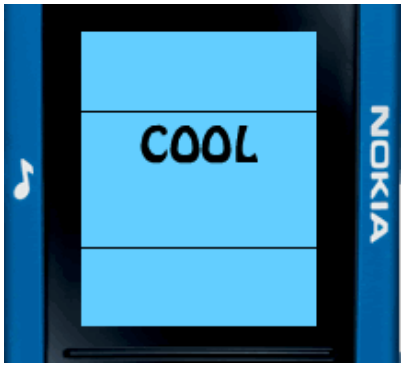

Figure 7: Border shows margin around text.

To address this problem use the `autoSize` property of the TextField class to adjust the width and height of the `msg` TextField to fit the text.

4.1.1 Using the TextField autoSize property

The following code configures the `msg` TextField so that its width expands or contracts to the size of the text. The value of `autoSize` is "center" to coincide with the original alignment setting from the properties inspector of the `msg` TextField.

```
/*
apply formatting to the msg textfield object in the movingobject
movieclip
*/
// fit different text length and font size, align center
cool.movingobject.msg.autoSize = "center";
```

With this configuration, Flash Lite 2.0 changes the `msg` TextField width and height according to the font size of the text and also to fit the text length of different words. Flash Lite 2.0 also adjusts the size of the movingobject movieclip so there is no extra space to the right or left of the text. Consequently, the moving object will appear to bounce when the text reaches the edge of the screen, which is the desired visual effect.

4.1.2 Using the TextFormat size property

To change the font size, use the `TextFormat` class. First, create an instance of the `TextFormat` class called `basestyle`:

```
// create a style for the text field
var basestyle:TextFormat = new TextFormat();
```

Then set the new font size by assigning a value to the `size` property of `basestyle`. Use a `switch` statement to assign a font size according to the `Stage.height` property. This adjusts the font size according to the phone screen size.

```
// set font size based upon Stage height
switch(Stage.height){
   case 320: basestyle.size = 40; break;
   case 160: basestyle.size = 24; break;
   default: basestyle.size = 34; // default font size for other stage
sizes
}
```

Finally, apply the format using the `TextField setTextFormat()` method by passing the `basestyle` instance to the method.

```
// apply style to the text field
cool.movingobject.msg.setTextFormat(basestyle);
```

When you publish your SWF to different stage sizes to match different phone screen sizes and test the SWF in Device Central or on a phone, Flash Lite 2.0 will adjust the font size of the moving object based on the stage size of the SWF.

4.1.3 Repositioning the wall due to dynamic text margin

At this point if you test the SWF in Device Central you will see that the font size correctly adjusts according to the stage size. However, during the animation, the moving object appears to bounce before the text reaches the top or bottom walls.

This is due to a margin that Flash Lite adds above and below text in a dynamic textbox. This margin makes the physical height of the moving object movieclip taller than the size of the text. There is no formatting option to remove the top and bottom margin that Flash Lite adds for dynamic text boxes.

To work around this problem reposition the top and bottom walls so that the moving object movieclip moves beyond the top and bottom of the screen enough so that the text reaches the edge of the screen before bouncing.

The following lines of code calculate the size of the margin and then shift the `_y` position of the top wall upward and the bottom wall downward so that both walls are now off of the visible portion of the stage.

```
// adjust y position of top and bottom walls so text reaches edge of
screen before bouncing.
// text top and bottom margin increases movieclip actual height to be
taller than text height.
var margin:Number = (cool.movingobject._height - basestyle.size)/2; //
calculate margin
cool.topwall._y -= margin; // shift topwall upward by size of the margin
cool.bottomwall._y += margin; // shift bottomwall downward by size of
the margin
```

During the animation the moving object will travel off of the stage by the amount of the margin so that the text reaches the edge of the screen before bouncing.

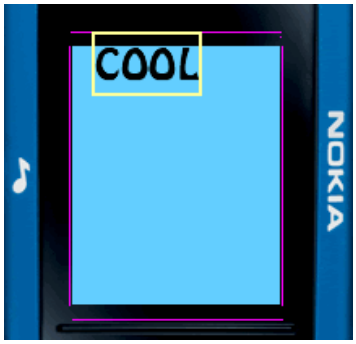

Figure 8: Pink lines indicate the position of walls, yellow box indicates true size of the moving object movieclip.

5 Example 3: Adding visual effects

Building on the previous examples, in example 3 you will use ActionScript techniques to add more variety to the screen saver. In this example, you will learn techniques to tile an image across the stage. You will create tiled movieclips that display in a background layer and a foreground layer and use the setMask() function to enable the shape of the text in the moving object to mask the foreground layer. You will also randomly select a new word for the moving object so the shape of the mask changes each time the screen saver plays.

Figure 9: Example 3 with tiled background and dynamic mask

5.1 Tiling a texture across the stage

Tiling a seamless texture across the stage is a useful technique because it enables you to create an interesting visual effect that also dynamically scales to any phone screen size.

First you prepare some textures and add them to your FLA library. The *bouncinganimation_withmask.fla* included in the example 3 folder of the article download contains a number of textures in the library that you can choose from. You can also add your own texture through the following steps:

1. Using a graphic creation software, create a texture in JPEG or PNG format.
2. Import your desired texture into the Flash CS3 library.
3. Create movieclip assets in the library for each texture.
4. Give each movieclip an ActionScript identifier so you can dynamically attach the movieclip to the stage at run time using ActionScript.

In your class file, create a new function named `createTiledMC()` to tile a specified texture to cover the entire stage. The function will place all of the tiles into a container movieclip so you can stack layers of tiled patterns on the canvas movieclip to create masking effects.

The `createTiledMC()` function expects a `containerName` variable representing the movieclip to hold the tiled textures, and a `textureName` representing the ActionScript identifier of the movieclip containing the desired texture bitmap image from the library.

```
/*
create a new movieclip named by containerName variable
tile the specified texture movieclip across the containerName movieclip
return the newly created movieclip
*/
```

```
private function createTiledMC(containerName:String,
textureName:String):MovieClip{
```

For flexibility, enable this function to dynamically change the texture during run time. The following `if/else` code block determines if there is a pre-existing movieclip of the same name as the containerName variable. If this movieclip already exists then ActionScript saves its depth in the `depth` variable and removes the pre-existing movieclip, clearing its tiled textures from the stage. If there is no pre-existing movieclip then ActionScript will store the next available depth in the `depth` variable.

```
// remove pre-existing tiled movieclip
if(this.canvas_mc[containerName] != null){
   var depth:Number = this.canvas_mc[containerName].getDepth(); // store
the depth
   this.canvas_mc[containerName].removeMovieClip() // remove the pre-
existing tiled movieclip
} else {
   var depth:Number = this.canvas_mc.getNextHighestDepth(); // new depth
}
```

Next, ActionScript will create a new `tilecontainer` movieclip at the previously determined depth. The `depth` sets a movieclip object's vertical stacking order in relation to other movieclips in the same parent movieclip.

```
// create a movieclip to hold the tiles
var tilecontainer:MovieClip =
this.canvas_mc.createEmptyMovieClip(containerName,depth);
```

By keeping track of the `depth`, you can use ActionScript to replace a pre-existing movieclip with a new one at the same depth, preserving the stacking order of all dynamically generated movieclips on the canvas movieclip. This prevents ActionScript from inadvertently stacking a tiled movieclip intended to appear as a background layer, in the foreground on top of the moving object obscuring it from view.

Now that you have created the `tilecontainer` moveclip, you can attach the first tile to the screen saver.

```
// attach a movieclip containing the selected texture
var texture:MovieClip =
tilecontainer.attachMovie(textureName,"texture",tilecontainer.getNextHig
hestDepth(),{_x:0,_y:0});
```

This code attaches the moveclip asset from the library with the ActionScript identifier stored in the `textureName` variable, and places it on the stage in the upper left hand corner. Next, duplicate this texture across the stage to create a tiled effect.

The first step to tile the background is to determine how many copies of the texture you need to create. You can think of your tiled movieclip as consisting of rows and columns of duplicated images.

```
// determine the number of rows and columns required to tile this
texture over the entire stage
var rows:Number =
(Math.ceil(Stage.height/tilecontainer.texture._height));
var cols:Number = (Math.ceil(Stage.width/tilecontainer.texture._width));
```

These two statements calculate the number of rows and columns required to cover the stage with the selected texture based upon the dimensions of the texture compared to the dimensions of the stage. The `Math.ceil()` function converts any decimals to the next highest whole number.

The final step is to set up loops to tile the texture across the stage. Use a `for loop` to count the number of rows and a `while loop` to add tiles based upon the number of columns.

```
// loops to tile the texture
for(var r:Number = 0; r<rows; r++){ // loop through rows
   // texture already attached in row 0 column 0
   // for first row start in second column (c1)
   var c:Number = (r==0) ? 1 : 0;
      while(c<cols){ // loop through columns
      var mcname:String = "r" + r + "c" + c; // generate a movieclip
name based upon row and column
      var xpos:Number = tilecontainer.texture._width * c;
      var ypos:Number = tilecontainer.texture._height * r;

   tilecontainer.texture.duplicateMovieClip(mcname,tilecontainer.getNext
HighestDepth(),{_x:xpos,_y:ypos});
      c++;
   }
}
```

You can consider the starting texture as being the first column of the first row. Before starting the while loop ActionSript checks if `r == 0`, meaning the first row. If the loop is in the first row, set the column count variable named `c` to 1 to start placing textures in the second column of the first row, because you have already attached a tile to the first column of the first row.

The `while` loop contains the code to attach the tiles. First ActionScript will build a unique name for the tile movieclip, based upon its row and column position, that is, "r0r1". Then it determines the `x` and `y` position of the next tile to be attached and finally calls the `duplicateMovieClip()` method to make a copy of the texture at the new position. You can define the `_x` and `_y` properties for the new movieclip by passing these properties in object shorthand, that is, `{_x:xpos,_y:ypos}`, as the last argument of the `duplicateMovieClip()` method.

Finally, after creating the `tilecontainer` movieclip and duplicating the tile across the stage, the function returns the `tilecontainer` as a movieclip to the calling function. In the next section you will define the functions for creating a background layer and a masked layer. These functions will call the `createTiledMC()` function to receive the tiled movieclip.

```
   // return the tiled movieclip to the calling function
   return tilecontainer;
}
```

5.2 Creating a tiled background layer

Now that you have created the core function for creating a tiled movieclip, you can add functions and properties to the class to define a background layer with a tiled texture. First create a `private` read-only property to store the texture used for the background. Add code to define the property and its corresponding getter function.

```
/*
read only property, texture for background
*/
private var __backgroundtexture:String;

public function get backgroundtexture():String{
   return this.__backgroundtexture;
}
```

Next, create the `backgroundlayer` property which is a movieclip representing the tiled texture to be used in the background layer.

```
/*
movieclip containing background image
*/
public var backgroundlayer:MovieClip;
```

Finally, define a function, `createBackgroundLayer()` that expects a texture name. This function assigns the texture name to the `_backgroundtexture` property and then calls the `createTileMC()` function to build the tiled movieclip, and assign this movieclip to the `backgroundlayer` property.

```
/*
creates the tiled backgroundlayer movieclip
*/
public function createBackgroundLayer(asset:String){
   this.__backgroundtexture = asset;
   this.backgroundlayer = this.createTiledMC("backgroundmc",asset);
}
```

In your FLA ActionScript add a line of code to create the tiled background movieclip layer. The value `drops1` is the ActionScript identifier of a movieclip in the library containing the texture you want to tile onto the `backgroundlayer` movieclip.

```
// FLA ActionScript to create tiled layer for the backgroundlayer
property.
cool.createBackgroundLayer("drops1");
```

An important issue to be aware of is that the `createTileMC()` function will build the backgroundlayer movieclip at the next highest depth on the canvas movieclip. It does not automatically place the `backgroundlayer` movieclip in the background behind other assets on the canvas. This line of code must occur before you call `setMovingObject()`, so that the movingobject movieclip is built in a depth higher than the `backgroundlayer` movieclip. Otherwise the `backgroundlayer` movieclip may appear in front of the `movingobject`,obscuring it from view.

5.3 Creating a tiled masked layer

Follow the same steps for the `backgroundlayer` movieclip to create the class functions and properties to build the `maskedlayer` movieclip.

```
/*
read only property, texture for mask
*/
private var __maskedtexture:String;

public function get maskedtexture():String{
   return this.__maskedtexture;
}

/*
movieclip containing tiled image to be masked by moving object
*/
public var maskedlayer:MovieClip;

/*
creates the tiled maskedlayer movieclip
*/
public function createMaskedLayer(asset:String){
   this.__maskedtexture = asset;
   this.maskedlayer = this.createTiledMC("maskedmc",asset);
}
```

To properly create the mask effect you must build the `backgroundlayer` movieclip first, followed by the `maskedlayer` movieclip, and lastly the `movingobject` movieclip. The following code demonstrates the order in which you must build your screen saver assets to enable the mask effect work as you intend.

```
// set these assets in proper screen stacking order
cool.createBackgroundLayer("drops1"); // build first, in background
cool.createMaskedLayer("water2"); // build second, in foreground
cool.setMovingObject("textmc",Stage.width/2,Stage.height/2); // build
last, on top
```

Once you have created all of these assets and stacked them in the proper order you can apply the `movingobject` movieclip as a mask using the `MovieClip.setMask()` method.

5.3.1 Understanding dynamic masks

A mask displays a portion of an image from a layer beneath the mask, in the shape of the mask. It is literally like placing a mask on a person's face, you only see the portions of the face where there are holes in the mask. In example 3 you are using the shape of the text to mask the maskedlayer tiled texture.

Figure 10: Text on top of tiled pattern

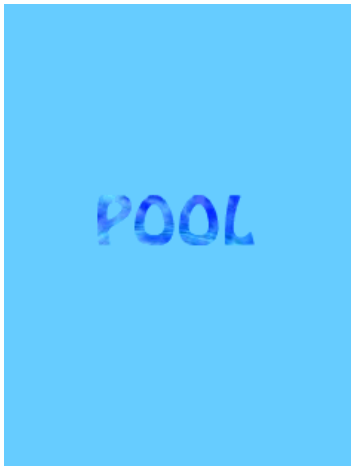

Figure 11: Text applied as a mask on top of tiled pattern

Flash Lite 2.0 supports dynamic masks which are movieclips that act as a mask and can move on screen and be manipulated by ActionScript. To create a dynamic mask use the `MovieClip.setMask()` method. To apply the `movingobject` movieclip as a mask for the

maskedlayer movieclip, pass the movingobject movieclip as an argument to the setMask() method of the maskedlayer movieclip.

```
// FLA ActionScript to mask the maskedlayer movieclip with the
movingobject movieclip
cool.maskedlayer.setMask(cool.movingobject); // set the mask
```

As the mask moves around on the screen it shows the varied qualities of the "watery" texture in the maskedlayer, creating a rippling water effect.

5.4 Randomizing the text

As a final touch for your screen saver, add some code to randomly select a word from a list and make this the text for the msg textfield in the movingobject. Each time Flash Lite plays the screen saver animation it will display with a different word adding more variety through randomness.

First create an array called messages to hold the different words for the msg textfield.

```
// define a list of cool messages
// change text before applying text formatting
var messages:Array =
["COOL","POOL","SWIM","dive!","splash!","sploosh!"];
cool.movingobject.msg.text = messages[random(messages.length)]; // get a
random message
```

You can use the random() function to randomly select an index from the array and assign the corresponding word to the text property of the msg textfield. The random() function generates a number between 0 and the number you pass – 1. In this case, you pass the length of the messages array, which is 6, to the random() function and it will return a value between 0 and 5, which corresponds to the index range of the array.

Note: The random function is deprecated Flash Lite 2.0; however, it is a simple and well suited option for selecting a random index for an array.

In your FLA ActionScript code, make sure you assign the new word to the text property of the msg textfield before you change the font size. Otherwise, Flash Lite will reset the text formatting of the msg textfield to the font size of 50 point that you originally assigned in the property inspector.

6 Summary

You have now completed a screen saver that can dynamically resize itself to whatever stage size you publish it to. It adapts the size of all assets of the screen saver including the walls, the moving object, the tiled background, and the tile for the masked layer. In addition you have added visual interest with a dynamic mask and randomized text. By building the code in an object oriented manner you can quickly develop a new screen saver or wallpaper animation and easily customize its behavior by extending the class or through ActionScript in your FLA.

7 About the author

Hayden Porter, the author of this document, is a Flash Lite developer with a special interest in developing multimedia content for mobile devices. He has written extensively on the subject of developing mobile content, including white papers for leading mobile device manufacturers and articles for publications such as Electronic Musician Magazine, Music Education Technology Magazine, and DevX.com.

For more information, see http://www.aviarts.com.

8 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by rating this resource.