

# Ticker & Scroller

## Playing with text in Flash7

*by Armand Niculescu  
Media Division*

*It's been 3 years since my last tutorial on Flash programming. So much has changed, it's impossible to pick up where I left. Actionscript has gone through many changes, becoming more mature. The 'flashers' have changed too. Because of its complexity, Flash is no longer for everyone - we no longer have 'flashers' – we have specialized groups, Designers and Developers.*

*Some time ago I wanted to do an introduction in Object Oriented Programming, but my work has kept me away from dedicating time and energy for this, and others (like Robin Debreuil) have done it better than I could.*

*Still, I realize there are many things one can learn, and today I want to share with you a very simple application: the text ticker. Most of these applications are built with javascript, which is OK, but the scrolling is not smooth unless you do a lot of tricks. So, we'll do it in Flash and we'll make sure it's easily configurable, so that there's no need to recompile every time the text or layout changes.*

*In this tutorial you'll learn how to use `setInterval`, how to read text from a file and how to create and style a `textField`.*

*Have fun,*

*Armand Niculescu*

[armand@media-division.com](mailto:armand@media-division.com)

[www.MediaDivision.com](http://www.MediaDivision.com)

[www.RichNetApps.com](http://www.RichNetApps.com)

## The Ticker

The text ticker is just a line of text that scrolls right to left, over and over. It's one of the simplest things you can do with Flash.

Start with an empty Stage of 200x20 pixels.

Add this code to the first frame:

```
1. Stage.scaleMode = "noscale";
2. var myStyle:TextField.StyleSheet = new TextField.StyleSheet();
3. var myData:XML = new XML();
4. var scrollSpeed:Number = 2;
```

The ticker will be embedded in a html page, so we want to stage to resize, not rescale, that is we want the Stage to have more/less pixels available for drawing, not to have the elements on the Stage larger or smaller, that's why we use `Stage.scaleMode = "noscale"`

Then, we'll read data from two files, the text and the style. The `scrollSpeed` variable is just the number of pixels scrolled at at time.

Lets build the files first:

`Tickerstyle.css` will contain the basic definitions for paragraph text and links. Of course you can add other definitions.

```
1. /* tickerstyle.css */
2. p {font-family: Arial, Helvetica, sans-serif; font-size: 12px;}
3. a:link {color: #CC0000; font-weight: bold}
4. a:hover{color: #FF6600; text-decoration: underline; font-weight: bold}
```

`Tickertext.htm` will contain the text and its formatting. Usually it's just text and links:

```
1. <!-- tickertext.htm -->
2. <p>Some information with a <a href="#">link</a>... And also with
   <b>bold</b> and <i>italic</i> and even more text if needed... end of
   text...</p>
```

Save the files and get back to Flash.

Now that we have the text files, we need to load them. For the purpose of this tutorial, we won't handle errors – if Flash can't load a file, it won't do anything. You can improve by adding error messages or other more advanced handling.

With the `myStyle` variable defined, we must load a stylesheet in it and upon loading, to trigger another event, so we should add this:

```
5. myStyle.onLoad = function(success:Boolean)
6. {
7.   if (success)
8.     myData.load("tickertext.htm");
9. };
10.myStyle.load("tickerstyle.css");
```

The `onLoad` handler, if the load was successful, will cause the loading of the text file. As you can see, the chain of events is initiated with `myStyle.load("tickerstyle.css")`

Now, we must add a handler for successful loading of the text file, and here is the first trick. We have defined **myData** as XML, but actually all we want is to load a string of text. Flash doesn't have a specialized way of loading simple text files, but it has an **onData** event for XML files, that returns the raw file, before it's parsed as XML. If you set up an **onData** handler, **onLoad** won't execute. This way, in theory you could build your own XML parser.

Add this:

```
11.myData.onData = function(src:String)
12.{
13.  if (src != null)
14.  {
15.    createTextField("tickText", this.getNextHighestDepth(), Stage.width,
16.    0, 10, Stage.height);
17.    tickText.html = true;
18.    tickText.selectable = false;
19.    tickText.styleSheet = myStyle;
20.    tickText.htmlText = src;
```

So, if the loading has succeeded, we'll have the text file given to us as a string. We can now create a textfield (with **createTextField**) and set its properties. We set its **html** property to **true**, so that the links work, and **selectable** to **false** (you could leave it **true**). Assigning the previously loaded stylesheet is easy with **tickText.styleSheet=myStyle**. Finally the loaded text is assigned to the textfield.

The textfield height is equal to the Stage height, you should make sure there's enough space for the text to be fully visible. We need to calculate the textfield necessary width (**Stage.width** is temporary). As homework, try to optimize the code below by using the **autosize** property instead.

```
20.  var format:TextFormat = tickText.getTextFormat();
21.  var sizes:Object = format.getTextExtent(tickText.text);
22.  tickText._width = sizes.textFieldWidth;
```

To get the needed width, we use the **getTextExtent** method, which is part of the **TextFormat** class.

Finally, we start the animation.

To animate the text, we can't use the "old-fashioned" way of using the timeline, because the textfield is created dynamically and it's width is also variable. To animate, we use **setInterval**. If you don't know what it is, well, it's a simple way of calling a method repeatedly at predetermined time intervals. The great thing about it is that it independent of the timeline and frame rate.

```
23.  var scrollInterval:Number = setInterval(scroll, 20);
24.  }
25.};
```

Now the method **scroll** will be called every 20 milliseconds. The job is handled by Flash and we

have control over it via the returned `scrollInterval`. If we need to stop the method from being called, we can just use `clearInterval(scrollInterval)` – although we won't need it here.

And now we've arrived to the heart of our little program – the scroll method:

```
26.function scroll()
27.{
28. tickText._x -= scrollSpeed;
29. if (tickText._x < -tickText._width) tickText._x = Stage.width;
30.}
```

“*Is that all?*” I can hear you saying, Yes, it is. Every time the `scroll` method is called, it will decrease the `x` position of the textfield, until it completely runs off the screen, at which point it will be moved to the right side of the screen.

So, in 30 lines of code, we made it. There's just one thing left to do, embedding it in a html page. Make a new html page and insert this code in it:

```
1. <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
   codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.
   cab#version=7,0,19,0" width="200" height="20">
2.   <param name="movie" value="ticker.swf">
3.   <param name="quality" value="high">
4.   <param name="wmode" value="transparent">
5.   <embed src="ticker.swf" width="200" height="20" quality="high"
   pluginspage="http://www.macromedia.com/go/getflashplayer"
   type="application/x-shockwave-flash" wmode="transparent"></embed>
6. </object>
```

Note that you can make the Stage bigger or smaller. The `wmode=transparent` parameter allows you to put the ticker just about anywhere, over any background.

## The Scroller

The Scroller inherits from Ticker, but the text scrolls up. It's more complex in that there are several lines of text that must be scrolled one a time. As an added twist, the text must pause in the middle of the screen for a while before moving again.

Make a bigger Stage, say 200x40 pixels.

The first lines of code are:

```
1. Stage.scaleMode = "noscale";
2. Stage.align= "TL";
3. var myStyle:TextField.StyleSheet = new TextField.StyleSheet();
4. var myData:XML = new XML();
5. var scrollSpeed:Number = 2;
6. var textArray:Array;
7. var scrollInterval:Number;
8. var textItem:Number;
9. var startTimer:Number;
```

```
10.var maxDelay:Number = 2 * 1000;
```

`textArray` will hold our lines of text. `textItem` will tell the current index in array. `startTimer` will be used to measure the time spent by the line of text on the screen, while `maxDelay` is the time the text will stay on the screen.

The text file is now split on lines:

```
1.<p align="center">Some information with a <a href="#">link</a></p>
2.<p align="center">And also with <b>bold</b> and <i>italic</i></p>
3.<p align="center">and even more text if needed</p>
```

Style loading is the same:

```
1.myStyle.onLoad = function(success:Boolean)
2. {
3.   if (success)
4.     myData.load("scrolltext.htm");
5. };
6.myStyle.load("scrollstyle.css");
```

The textfile loading is similar, but we must handle the different lines:

```
7.myData.onData = function(src:String)
8. {
9.   if (src != null)
10.  {
11.    createTextField("tickText", this.getNextHighestDepth(), 0,
    Stage.height, Stage.width, 20);
12.    tickText.html = true;
13.    tickText.selectable = false;
14.    tickText.multiline = true;
15.    tickText.wordWrap = true;
16.    tickText.styleSheet = myStyle;
17.
18.    textArray = src.split("\n");
19.    textItem = 1;
20.    tickText.htmlText = textArray[textItem];
21.
22.    scrollInterval = setInterval(scroll, 20);
23.  }
24.};
```

So, we split the text in lines, each one being an element in the array, before starting the animation.

The scroll method however, is a little more complex:

```
25.function scroll()
26. {
27.   if (Math.abs((Stage.height/2)-(tickText._y+tickText._height/2)) <=
    scrollSpeed)
28.  {
29.    if (startTimer == null)
30.      startTimer = getTimer();
31.    else
32.      {
33.        if ((getTimer() - startTimer) >= maxDelay)
```

```
34.     {
35.         tickText._y -= scrollSpeed*3;
36.         startTimer = null;
37.     }
38. }
39. }
40. else
41. {
42.     tickText._y -= scrollSpeed;
43.     if (tickText._y + tickText._height < 0)
44.     {
45.         textItem = textItem < (textArray.length-1) ? textItem+1 : 1;
46.         tickText.htmlText = textArray[textItem];
47.         tickText._y = Stage.height;
48.     }
49. }
50. }
```

The scroll method is a little more complex, so lets dissect it. First, the simple case of scrolling up is done at the end, here:

```
42.     tickText._y -= scrollSpeed;
43.     if (tickText._y + tickText._height < 0)
44.     {
45.         textItem = textItem < (textArray.length-1) ? textItem+1 : 1;
46.         tickText.htmlText = textArray[textItem];
47.         tickText._y = Stage.height;
48.     }
```

The code is commented, so you shouldn't have any problem understanding it. The textfield scrolls up, when it's out of the Stage, the text inside the textfield is replace with the next item in the array that holds the lines of text and the textfield is moved at the bottom of the Stage.

The other part of the code deals with pausing the text in the middle of the screen for a while (defined in **maxDelay**).

```
27. if (Math.abs((Stage.height/2)-(tickText._y+tickText._height/2)) <=
    scrollSpeed)
28. {
29.     if (startTimer == null)
30.         startTimer = getTimer();
31.     else
32.     {
33.         if ((getTimer() - startTimer) >= maxDelay)
34.         {
35.             tickText._y -= scrollSpeed*3;
36.             startTimer = null;
37.         }
38.     }
39. }
```

When the text is approximately in the middle of the screen (given that the textfield height and the scrolling speed may be variable, we don't know if the text can arrive exactly in the middle) we record the relative time (and don't do anything else). Then, when the method is called again, the

text is in the middle and the time recording exists, so we measure the time passed since the recording. If enough time has passed, we move the text out of the middle and delete the time recording.

I must confess I am not very satisfied with this solution. When one makes a tutorial, things like clarity, complexity and efficiency must be carefully balanced. I chose this solution because it seems simple (to me) and because it may show you how to use `getTimer()`. I'd be very happy if you'd try to come up with a more elegant solution – for example you could `clearInterval(scrollInterval)` when the text is in the middle and create another interval, `delayInterval` to execute a method `resumeScroll()` after `maxDelay` milliseconds. `resumeScroll()` would `clearInterval(delayInterval)` and restart `the scroll()`.

You may try something different entirely, so go ahead, be creative!