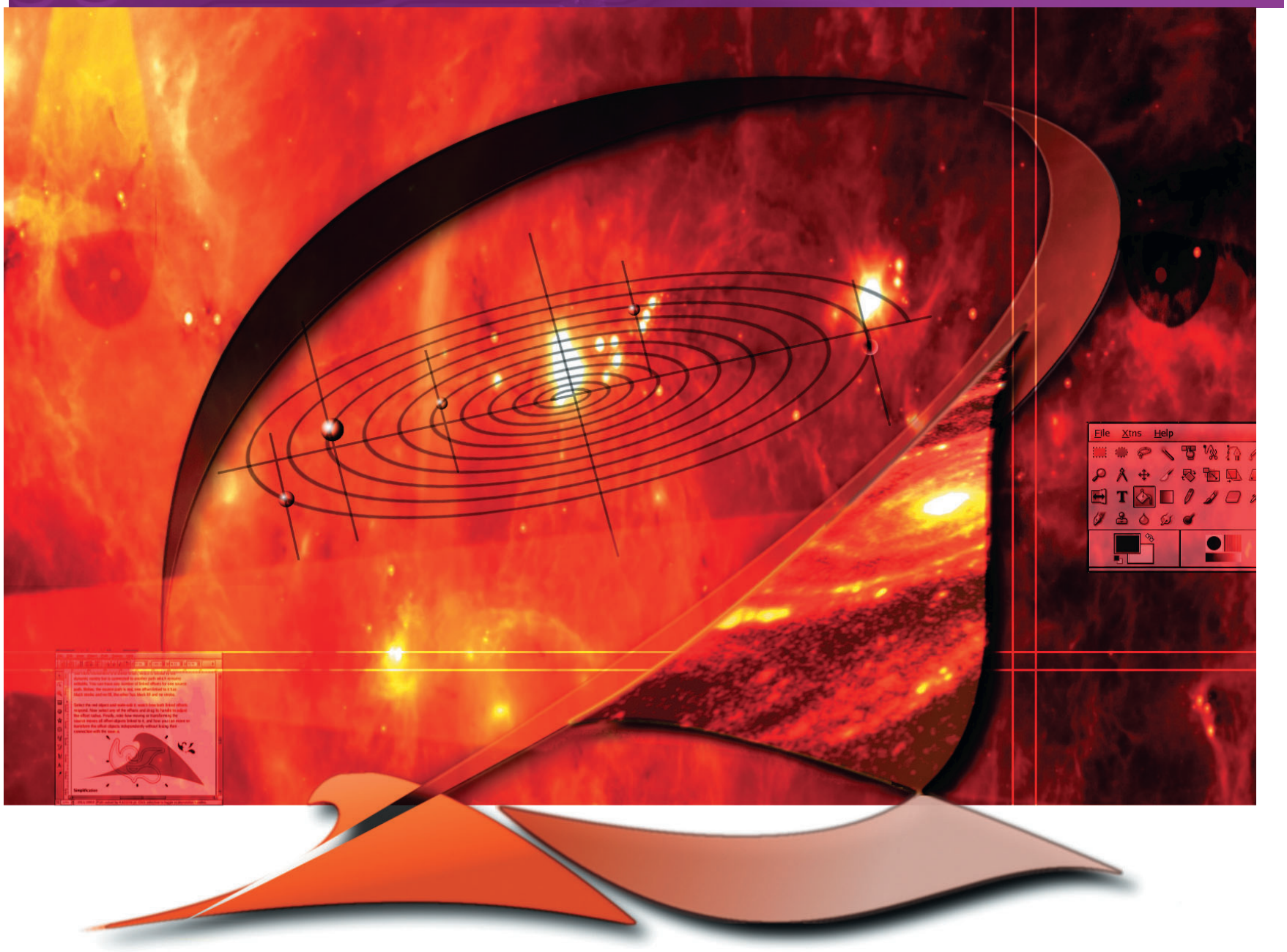


IMAGES FOR WEB

# Building Flash animations with PHP and Ming: text effects

**PART 2** **Programming Flash effects is a simple process that requires nothing more than modest looping, a drop of maths and your own creative ideas, says Michael J Hammel.**

ON the DISC

**ON THE DISC**

**All the code listings required for this tutorial can be found on your coverdisc.**

Last month we introduced *Ming*, the *Flash* library with multiple language bindings. In that article, we covered the concept of *Flash* movies, frames and shapes, and how to combine them inside loops to create simple animations. This month we'll dig into text animations. *Ming* handles text as objects, just as it does graphic images. A *Ming* program will establish one or more font objects and uses those objects in individual text objects. Once the text object characteristics – font, height, spacing and so on – are configured, the text object can be manipulated just like the graphic objects discussed last month. This means you can position them any way you want in successive frames, producing animated text.

## Ming fonts

Text in *Ming* is a little more difficult than other tools under Linux because you have to carry around your own set of font files – you can't simply use the fonts that you already have on your system. Instead, you have to convert them to a special format and keep them close at hand to your *Ming* program, either by compiling them into your .swf file or by referencing them in a

path accessible to your PHP script and your web server. *Ming* fonts have their own special format called fdb. To get this format, you first convert a TrueType font to an intermediate format called fft. The tool to do this is *ttf2fft* and source for it can be found on the *Ming SourceForge* download page.

After the file is in fft format, you can pass it to *Ming*'s *makefdb* program to convert it to fdb format. **Listing 1** shows the output from running this command to convert an existing TrueType font file to fft, and then to fdb formats.

```
<Listing 1 – Convert TrueType fonts to FDB format>
% ./ttf2fft /usr/local/fonts/TrueType/xfiles-3
Converting font /usr/local/fonts/TrueType/xfiles-3.ttf
    family: X-Files
    style:  Regular
using charmap:   0: platform: Macintosh, encoding: Roman
retrieving chars from current charmap ... found 100 chars
using 1-byte char codes
retrieving glyphs outlines ... done.
generating glyph outlines ... done.
using short glyph offsets
generating layout
   writing advance table
   writing bbox table
   no kerning table found!

% ./makefdb X-Files.fft
Found DefineFont2 block at offset 23
Block length: 11810
Writing X-Files.fdb, 11812 bytes
```

The resulting file should be placed in a directory called fdb, in the same directory as the source for these examples. This means all fonts can be referenced relative to the local directory.

*Ming* includes the makefdb tool in its source package in the util directory. Although this isn't built by default, it's easily built. All you have to do is run

```
  make makefdb
```

in that directory.

## A single-frame text movie

The first example this month consists of a single-frame animation. This is just like last month's, but this time it has a text message. The *Ming* environment is set up first. A movie object is created and its frame rate and dimensions set. The background is then initialised to grey. **Listing 2** shows this configuration.

```
<Listing 2 - Example 2 Source>
<?php

  // Ming requirements
  Ming_setScale(20.0);
  ming_useswfversion(6);

  // Initialise our environment.
  $movie = new SWFMovie();
  $movie->setRate(20.000000);
  $movie->setDimension(550, 400);
  $movie->setBackground(0xcc,0xcc,0xcc);

  // Setup the string and font to use.
  $str="Linux Format and PHP/Ming";
  $font="fdb/Bitstream Vera Sans-B-I.fdb";

  // Create font and text objects
  $f = new SWFFont($font);
  $t = new SWFText();
  $t->setFont($f);
  $t->setColor(0xff, 0x33, 0x33);
```

### WEB RESOURCES

Minging URLs

- **Ming:** http://ming.sourceforge.net/
- **Ming Source Downloads:** http://sourceforge.net/project/showfiles.php?group_id=18365
- **Ming Font package, containing the Bitstream Vera font family in FDB file format:** http://prdownloads.sourceforge.net/ming/ming-fonts-1.00.tar.bz2?download
- **Examples from this article:** www.ximba.org/articles/phpming2/article.html
- **Ming Function Reference:** http://ming.sourceforge.net/docs/

**Note that all of the methods (also known as functions) that are used in the examples in this article are detailed in the Ming Function Reference online documentation.**

```
  // Set some text characteristics
  $t->setHeight(40);
  $t->setSpacing(1.0);

  // Add the string to the movie.
  $t->addString($str);
  $i = $movie->add($t);
  $i->moveTo(275-$t->getWidth($str)/2, 200+$t-
>getAscent()/2);

  // Output directly to server
  header('Content-type: application/x-shockwave-flash');
  $movie->output();

?>
```

After the environment is established, a font file is loaded into a Font object. A text object is created and the font object is set as the font to use for this text. The colour of the text is set using the **setColor()** method. Remember, the background of the entire movie has been set to grey.

Next, specify what string to use in this text object. A text object's string, font, height, spacing and other characteristics are used to render the text into the movie. The height and spacing are explicitly set in this example, along with the dimensions of the movie. Some experimentation may be necessary to find a fit between the dimensions of the movie and the size of the text, as will be shown in our last example.

The string is added using the text object's **addString()** method and the text object is inserted into the movie's current (and only) frame. We then move the text object within that frame to a location that's centred on the dimensions of the movie. Finally, the movie is displayed.

## Text effects

As you can see, the only real difference between handling an image and handling text with *Ming* is that text requires two objects: an SWFFont object and an SWFText object. Once the text object is configured with its font properties, the text can be distorted using any of *Ming*'s numerous built-in text functions.

In the next example, shown in **Listing 3** on the coverdisc, a library of distortion functions – **infuncs.php**, which is shown in **Listing 4** – is included in the main script. This library is a modified version of the same functions in the Flashdot example from the *Ming* website. The modifications are primarily to allow for more flexible centring of the text.

The text in this example remains constant, but a series of transformations are applied to it, such as skewing, fading in or out, sliding in from the left or right, and so on. Each distortion function produces a series of 35 frames, followed by a series of »

20 unchanged frames via the **pause()** function, and finally a series of 20 blank frames (no text) via the **blankSpace()** function. Both **pause()** and **blankSpace()** are functions in the included distort function library and are not part of *Ming*.

The calls to **add()** are *Ming* functions that return a **SWFDisplayItem** object, and this is saved in the **$i** variable in each distort function. In each function, you can find the built-in Ming functions by finding a function referenced from the **$i** variable, such as **$i->scaleTo()** in the **zoomin()** distort function.

Most of these functions take either X,Y co-ordinate pairs or Red/Green/Blue/Alpha colour values. The **multColor()** function is especially interesting. It's used to multiply the current colour by various amounts specified as arguments to that function. In the distort functions, the Red, Green and Blue values to **multColor()** are set to 1, meaning the colour doesn't change. However, the Alpha value is varied from 0 (transparent) to 1 (opaque), which is how fade in and fade out effects are made.

One important thing to note for the next example is that all of the function names are placed in an array at the end of the distortion functions library. This enables us to pick one of the functions in a random order.

The examples on your disc call the distort functions in a defined order and don't loop or change the string used in the text object. The script only changes the shape of that text when it's displayed over a series of frames.

## Using random effects on multiple text strings

While those effects are interesting, you might wonder what they could be used for. The most obvious answer is as a scrolling headline. Place an animated text movie inside a DIV block in your HTML and position it at the top or bottom of your page. Now modify **Example 2** to read in a text file of headlines, such as an RSS feed. As it turns out, the Flashdot example does exactly this. However, the code there is much more complex than I need to get in this tutorial, so I'll settle for a simple text file that lives right in our local directory for **Example 3**. Wrap this code inside a simple HTML file that references a simple CSS file, both of which are shown in **Listing 7**.

The CSS is pretty blunt, telling the browser that any embedded or object tag will be 16 pixels tall and will cover the width of the display. That's fine for this example, but you'll probably want something that's not quite so specific.

The first big change from **Example 2** is that I create a series of SWFText objects from the headlines. These are saved in an array for later reference. Each line from the Headlines file is then processed with a random distort function using the array of function names mentioned earlier, followed by the **blankSpace()** frames.

## See your scripts in action

If you look at the source files on your coverdisc together, you can see how each example builds upon the last. **Example 3** adds more strings and picks a random function, but other than that it's pretty much the same as **Example 2**. PHP and *Ming* are like that – it's easy to build one piece on top of another.

Screenshots are nice for applications but they don't really tell the whole story behind a movie, so you should really take a look at these scripts in action. You'll find them at the Ximba site (**www.ximba.org/articles/phpming2/article.html**). All the code from the examples featured in this month's tutorial can be downloaded in a single compressed tar file from there too. LXF