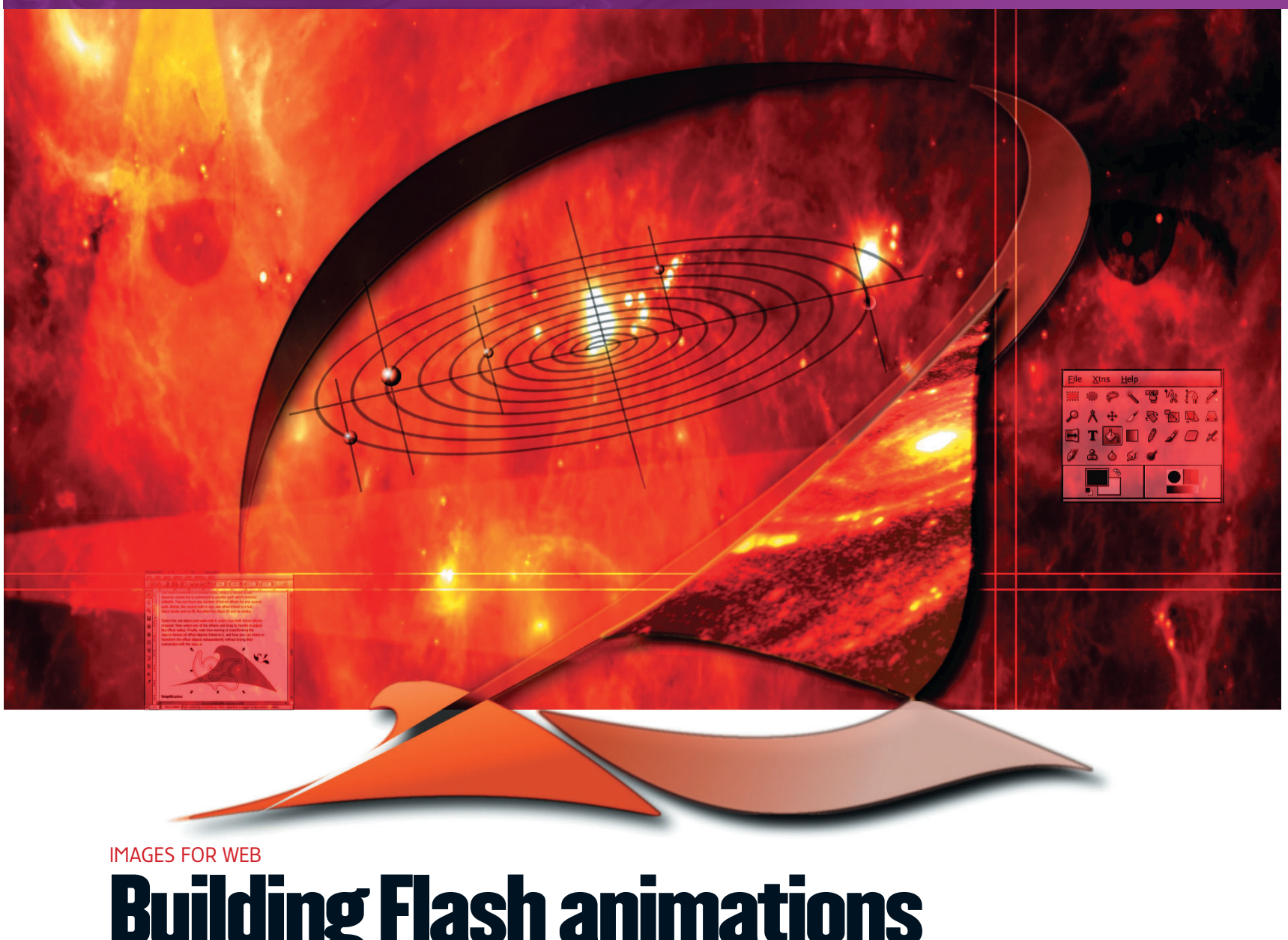
LXF59.tut_gimp 14/9/2004 10:25 AM Page 80



IMAGES FOR WEB

Building Flash animations with PHP and Ming

ON the DISC

Linux is on par with other platforms for displaying Flash animations; all that is missing is an easy way to create them. Michael J Hammel shows us how the Ming library fills that gap for developers.

Feeling motivated and organised, I recently completed a ground-up redesign of one of my websites. The inspiration for the new layout came from the wondrous designs I found at the CSS Zen Garden **www.csszengarden.com/**. CSS stands for Cascading Style Sheets, a web standard for page layout that is finally – after a far too long waiting period – well supported by most popular web browsers.

In the process of learning CSS, I discovered a few important things about browser support of Web standards. Most importantly, *Internet Explorer* doesn't understand standards; it consistently gets the box model wrong. The box model defines where and how sections of the document will be presented. *Internet Explorer* also doesn't understand transparency in PNG images. Because of that, layout design for *IE* is fairly limited. Your design options are far greater with *Mozilla*-based browsers, such as *Firefox*. Unfortunately, until the new revolution arrives, we're all stuck with dealing with *Internet Explorer* for many visitors to our sites.

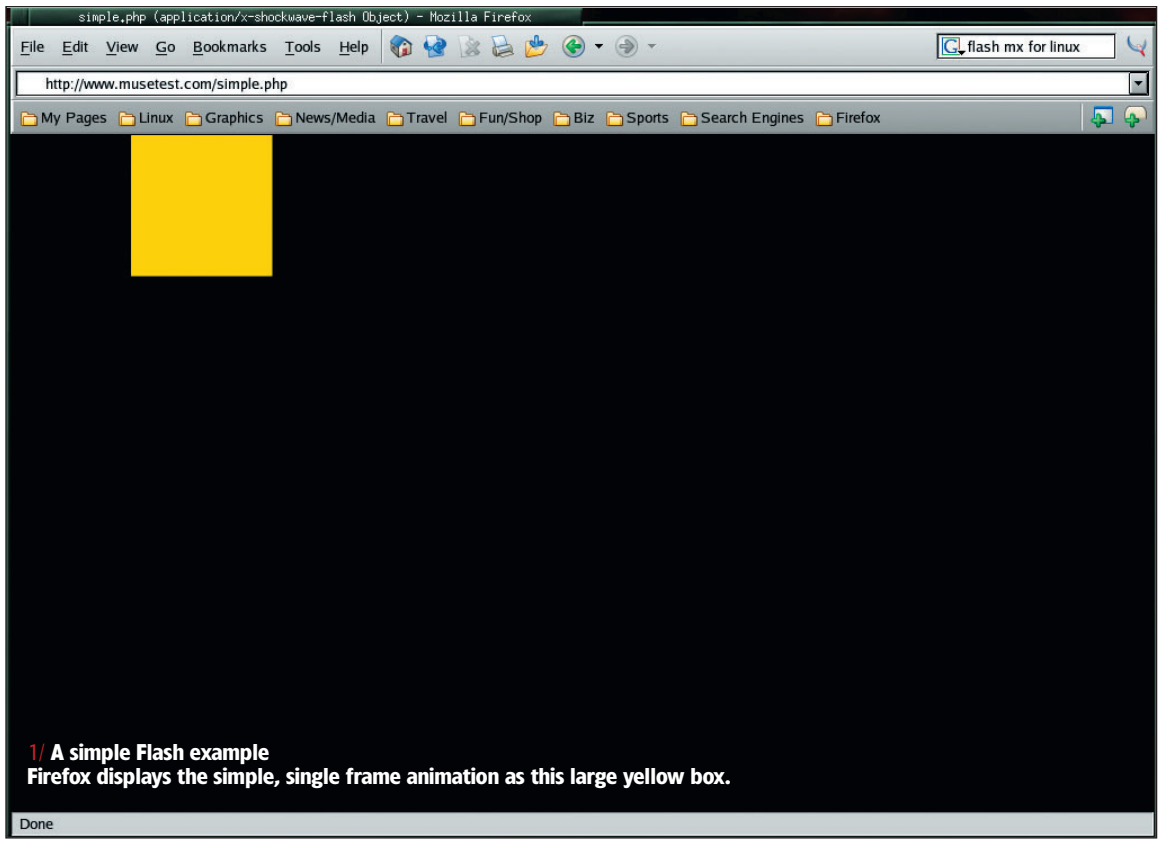
Despite this, there is one area where browser support has recently reached an equal footing – Flash. Macromedia's Flash is a highly popular animation format; which, while not strictly an official standard, is very popular indeed. Macromedia has provided browser plugins and players for many platforms: in the past year, Macromedia has not only released a supported Flash player for Linux, it hired someone to maintain it! So, while not an official standard or Open Source by nature, Linux users can at least live on par with their Mac and Windows brethren.

Unfortunately, there is still one part of this picture that is missing: *Flash MX for Linux*: the tool for creating Flash animations. Late in 2003 a version of *Flash MX* was certified to run with CrossOver Office, the *Wine*-based package that lets you run Windows applications on Linux. Macromedia has said that, if there was enough developer interest, a native port of *Flash MX* for Linux would be the next step. So far, news of such a port has not been made public: but as always in Linux, there is an alternative.

TIP

If you try to load your PHP script directly into your browser through your website, you'll need to make sure the the PHP module for *Apache* has been loaded. Check the webserver documentation for some more specific information on configuring PHP with *Apache*.

LXF59.tut_gimp 14/9/2004 10:25 AM Page 81



1/ A simple Flash example
Firefox displays the simple, single frame animation as this large yellow box.

RESOURCES

The official site for Ming is http://ming.sourceforge.net/. Documentation is thorough, if not always easy to follow. Sections include language-specific API references and examples. There's additional documentation for using *Ming* with PHP at http://us3.php.net/ming. The official Macromedia *Flash Player for Linux* can be downloaded from www.macromedia.com/shockwave/download/alternates/. A wealth of CSS (Cascading Style Sheets) information can be had from Zen Garden www.csszengarden.com/ – click through some of the 'Select a Design' options for a great demo of the power and flexibility of CSS.

Ming is a library of functions, complete with multiple language APIs, for generating Flash files. A Flash file is similar to a graphics image – it lives outside of your HTML and is referenced using a specific HTML tag. *Ming* supports much of the Flash file format specification, up to and including support for *Flash Player 7*, and can be used with programs written in C/C++, PHP, Perl, Python and Ruby. You can use *Ming* and *ffmpeg* together to convert an AVI video into a Flash FLV format and include it in your Flash animation. Flash ActionScript data can be compiled on-the-fly as well, allowing import of existing ActionScript programs. The potential here is limitless.

This article will look at using the PHP API with *Ming*, from installation and configuration to your first animation. I'll also cover basic requirements for referencing your animation from HTML. Specific language constructs will be covered briefly, but for a more detailed introduction to Flash you should consult the *Ming* websites or one of the many printed texts on Flash animation. There are also numerous tutorial sites for learning Flash ActionScript programming.

Grab, configure and go!

PHP is a scripting language developed primarily for the web, although in this project we'll use it on the command-line. The PHP interpreter uses a configuration file to load extensions to the base language. *Ming* is one such extension. To add the extension you need to compile the source code, drop the .so file into the correct directory and edit the PHP configuration file.

Ming requires PHP 4.0.2 or later (we used 4.2.2). Note that the newly released PHP5 will probably not work – yet, any way. You'll also need the development package for PHP. For users of *apt* (which I highly recommend); you can install these from just about any *apt* repository:

```
apt-get install php php-devel
```

apt is not just an application for Debian users, it's also available for Fedora, Red Hat and Yellow Dog distributions from **http://apt.freshrpms.net/**). You'll also need to get the *bison*, *flex* and *libungif* development packages:

```
apt-get install bison flex libungif
```

You're now ready to download the source from the *Ming* website. The release used for this project is 0.3Beta1. Unpacking the source will create a package directory. From within the package directory, build the *Ming* libraries:

```
tar xvzf ming-0.3beta1.tar.gz
cd ming-0.3beta1
make && make static
```

This will produce both shared (*libming.so*) and static (*libming.a*) libraries in the current directory. Next task to undertake is to change directories to the PHP extension directory and build the *Ming* PHP extension.

```
cd php_ext
make
```

From the same directory, and as the root user (to get access to system directories), install the PHP extension.

```
make install
```

The default location for PHP 4 extensions is /usr/lib/php4. You can change this in the PHP configuration file, /etc/php.ini. You also need to edit this file to add a reference to the newly installed extension. Add the following line to the section for extensions.

```
extension=php_ming.so
```

Red Hat users may want to copy one of the extension configurations that distribution uses that are in the directory /etc/php.d, but that particular configuration is beyond the scope of this article.

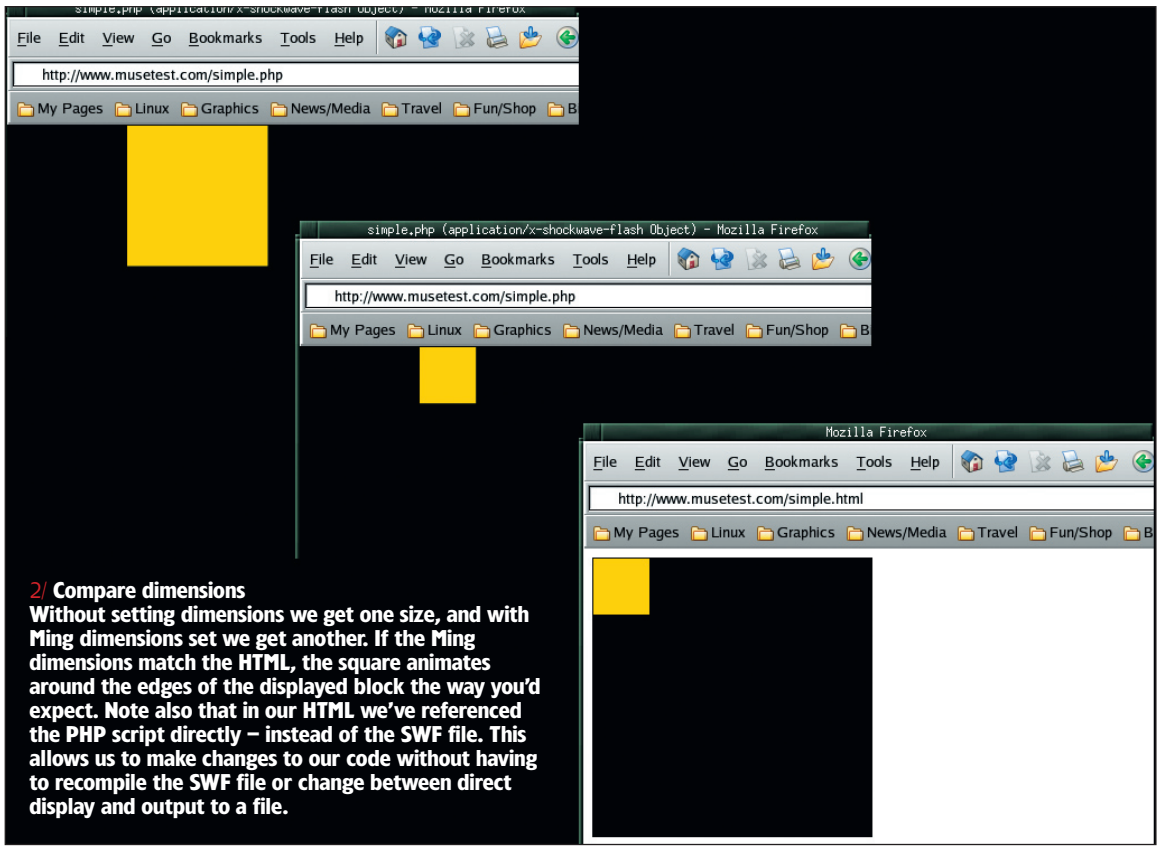
To verify the configuration works, try this command:

```
php -i | grep module_ming
```

This should return some text wrapped in HTML. If it returns nothing, verify you have installed the **php_ming.so** file in the PHP extensions directory specified in the /etc/php.ini file.

»

LXF59.tut_gimp 14/9/2004 10:25 AM Page 82



2/ Compare dimensions
Without setting dimensions we get one size, and with Ming dimensions set we get another. If the Ming dimensions match the HTML, the square animates around the edges of the displayed block the way you'd expect. Note also that in our HTML we've referenced the PHP script directly – instead of the SWF file. This allows us to make changes to our code without having to recompile the SWF file or change between direct display and output to a file.

A simple example

The process for creating Flash animations is simple: write some code and run the code to produce SWF content. This content can be output directly to the web or to a file for later embedding in HTML. To illustrate this process, we'll start with an example that draws a plain yellow box. We'd begin with something akin to the traditional "Hello World" programming example, but as handling fonts in *Ming* makes working with text a bit more complex than drawing simple shapes, we'll display a yellow box instead.

```
<?

  $movie=new SWFMovie();
  $movie->setBackground(0,0,0);
  $square = new SWFShape();
  $square->setRightFill(0xff,0xff,0);
  $square->drawLine(50,0);
  $square->drawLine(0,50);
  $square->drawLine(-50,0);
  $square->drawLine(0,-50);
  $squarepos = $movie->add($square);
  header('Content-type: application/x-shockwave-flash');
  $movie->output();

?>
```

The example starts as all Flash animations do, by defining a **movie** object. If you don't know object programming, just think of a movie as a series of frames to which we'll add other objects, such as shapes or imported animations. For example, the line

```
  $squarepos = $movie->add($square);
```

adds a object called **square** to the current frame of the movie object. We create a **SWFShape** object and tell *Ming* that this shape will be filled with yellow.

```
  $square = new SWFShape();
  $square->setRightFill(0xff,0xff,0);
```

The method **setRightFill()** tells *Ming* that when it draws pixels, from left to right, that the color specified will be used when ever it has crossed an odd number of defined line. We draw a square in this object like so:

```
  $square->drawLine(50,0);
  $square->drawLine(0,50);
  $square->drawLine(-50,0);
  $square->drawLine(0,-50);
```

The lines defined here determine when the yellow colour will be applied. Note that drawing with *Ming* is pen-based, which means the offsets specified in the **drawLine()** method are from the current pen location, with the origin (X,Y = 0,0) being the upper left corner. This example draws a line from X=0 to X=50 with Y=0, then draws a line from that point to y=50, then from that point back to X=0, then back to 0,0. That outline defines a shape inside which the yellow color is displayed.

After the **SWFShape** object called **square** is added to the **movie** object, we display the results directly to standard output.

```
  header('Content-type: application/x-shockwave-flash');
  $movie->output();
```

This method would be used if you placed this PHP file in your webserver to be loaded directly by visitors to your site. Another method would be to generate an .swf file that can then be referenced by HTML. In this latter case you'd replace these last two lines with:

```
$movie->save("simple.swf");
```

where **"simple.swf"** is the name of the output file. This file is then referenced with HTML code that would look similar to this:

```
<object>
<param name="movie" value="simple.swf">
<embed src="simple.swf"></embed>
</object>
```

This particular code should work with just about any browser that has a Flash plugin installed – including those that seem a bit vague in their observance of standards, like *Internet Explorer*!

TIP

The *png2dbl* program may not compile correctly on Linux systems due to a known minor bug in *png2dbl.c*. To fix this you need to add a missing line. First, find the following line:

```
#include <zlib.h>
```

and then add this line:

```
#define byte int
```

The program should then compile without error.

LXF59.tut_gimp 14/9/2004 10:25 AM Page 83

A simple animation

Animation is a series of individual frames played at a certain speed. First, we need to tell Flash how fast to play frames. We use a standard 60 frames per second rate:

```
$movie->setRate(60);
```

To animate the square we drew earlier, we will need only add a set of four loops – one each to move the square right, down, left, and up. Each loop adds a frame to the movie with the square moved to a new location. This code is added to our original code just after we call **movie->add()**.

```
for($i=0; $i<25; ++$i)
{
  $movie->nextFrame();
  $squarepos->move(8,0);
}
for($i=0; $i<25; ++$i)
{
  $movie->nextFrame();
  $squarepos->move(0,8);
}
for($i=0; $i<25; ++$i)
{
  $movie->nextFrame();
  $squarepos->move(-8,0);
}
for($i=0; $i<25; ++$i)
{
  $movie->nextFrame();
  $squarepos->move(0,-8);
}
```

At this point we need to talk about scaling. The PHP code introduced here does not specify the size of the animation. When we used the code that embedded the animation in HTML, we could have specified the size of the area to hold the animation, as in this example:

```
<object width="250" height="250">
<param name="movie" value="simple.php">
<embed src="simple.php" width="250"
height="250"></embed>
</object>
```

But the size of the animation itself is only bounded by this. It is scaled to fit inside this region. Note the difference between our code and what happens when you add the line

```
$movie->setDimension(800,600);
```

immediately after the line that calls **setRate()**.

Scaling issues will likely be one of the harder issues you try to resolve when learning *Ming*. For the time being, we will use the default (*ie* unspecified) scaling to prevent clouding the issue.

A Slide Show

We can use extend this simple animation to create a slightly more interesting one: a set of images shown and hidden by a sliding window. The first thing to do is add two images to our code. An image object in *Ming* is called an **SWFBitmap**. The image file must be in a special format called 'DBL' There is a utility program in the *Ming* source called *png2dbl* that will convert PNG images to DBL format. You need to build this tool manually. From the main *Ming* source directory, change into the util directory.

```
cd util
  make png2dbl
```

Once built, you simply run the program against any PNG image to generate its DBL equivalent.

```
png2dbl misc0018.png
```

The **.png** suffix is replaced with **.dbl** for the output image file.

FEEDBACK

Tell us what you want!

Is there any aspect of Linux-based art that you'd like to see covered in a future issue of *Linux Format*? Whether you'd like a more advanced recap of the subjects that we've already covered, or there's something that you'd like clarified, or think that there's perhaps something we've missed, then please email us at linuxformat@futurenet.co.uk with 'Art tutorial suggestion' as the subject-line. The whole *LXF GIMP* tutorial series will soon be available on the coverdiscs and online, so watch this space!

Having created your DBL image files, you can load them into your **SWFBitmap** object with a single line of code:

```
$image1 = new SWFBitmap(fopen("misc0018.dbl", "rb"));
```

Now add the image to the movie and position it at the display origin.

```
$image1pos = $movie->add($image1);
$image1pos->moveTo(0,0);
```

Repeat this process for a second image. Note that adding the second image after the first places it on top of the first. The yellow box is added after this. That means when we move the box up and out of the way the second image is exposed. To simulate the slide show, we simply move the second image out of the way while the yellow box is down and, when we raise the box again, the first image is exposed!

```
for($loop=0; $loop<2; ++$loop)
{
  $movie->nextFrame();
  if ( $loop == 1 ) $image2pos->moveTo(0,-250);
  if ( $loop == 2 ) $image2pos->moveTo(0,0);

  for($i=0; $i<25; ++$i)
  {
    $movie->nextFrame();
    $squarepos->move(0,-10);
  }
  sleep(5);
  for($i=0; $i<25; ++$i)
  {
    $movie->nextFrame();
    $squarepos->move(0,10);
  }
}
```

We only add enough frames to the movie to run through this process once. We'll let the Flash player/plugin handle looping the animation for us. That helps reduce the code size and lets the user decide how long the animation should run.

Our new PHP script can now generate our animation on the fly. But there is a catch: the image files have to be in the correct directory if we run the PHP directly from our webserver. If we compiled the program (using the output() method shown earlier) to generate an .swf file we would not need to include the image files and the code – we would just have the single .swf file.

Which method you use depends on what you want to accomplish. Having the image files on the webserver allows you to change them on the fly so visitors can see different images over time. This could be used to present different ads on a website, for example. But if managing multiple images and their associated PHP scripts is potentially difficult, then a single .swf file might be in order.

These simple examples are only meant to introduce you to programming with *Ming*. To see the real power of *Ming* and Flash you should check out the online examples. Text input, interactive drawing, and streaming audio are all possible with these tools. LXF

NEXT MONTH

Next issue, we'll be even more Merciless. We'll dive a little deeper into *Ming* by looking at integrating fonts and text into our animations.