by Jeffreys Copeland and Haemer

ALEX GROSS

"A picture is worth a thousand words."
– Anonymous

"If a picture were really worth a thousand words, people would draw that saying instead."
– Alan Haemer

Pictures

Once upon a time, a family would go on vacation and return with a dozen or more little yellow and red (or green and white) boxes of film to be developed. Not anymore. Copeland recently returned from a two-week swing through California with 60 MB of images loaded from his digital camera onto his laptop.

While he claims the family vacation was fun, we were both puzzled by what to do with the photos. They were interesting enough shots–San Francisco's Chinatown, beaches with fog, Santa Monica Pier in the sunset–but the file names, such as `IMG00003/EX_15/P0000234.JPG`, were no good for organizing the pictures. Worse, the software that loads the pictures onto the computer leaves the file with a date stamp matching the transfer time, not the time the picture was taken.

What do we know that can help us? Well, we know that the pictures are more or less in JPEG format, because our old favorite image-display tool `xv` says that's what they are. (JPEG stands for Joint Photographic Expert Group; it's a format that compresses the image by, among other things, intentionally throwing away some of the detail your eyes can't resolve.) But there's more data in the file than just the image. Let's take a look at one of the photo files:

```
$ mv IMG00003/EX_15/
   P0000234.JPG p2.jpg
$ strings p2.jpg
Exif
Minolta Co., Ltd.
Dimage EX
0200
0100
1999:07:26 14:33:16
Minolta Co., Ltd.
Dimage EX
erence Platform
JPEG
```

So there is a date buried in the data somewhere. In fact, that's even the date and time the picture was taken. So far, so good.

A little investigation reveals that the picture formats for digital cameras are covered by ISO standard 12234, "Electronic Still Picture Imaging–Removable Memory." (Which was on the Web for a while at the Photographic and Imaging Manufacturers Association Web site, but is no longer available.) The particular format of these images is called *Exif*, which was developed by the Japan Electronic Industry Development Association, or JEIDA. The text of the standard is available by mail order from Japan. Unfortunately, we are not that patient.

On the other hand, we can conclude (from ISO 12234) that while the Exif files contain the JPEG image, it is preceded by a TIFF-format list of data about the image, as a JPEG comment.

If you have no experience with images, right about now you should be asking, "What's TIFF?" It's the Tagged Image File Format originally

developed by Aldus, and now owned by Adobe Systems Inc. It is a kind of Swiss Army Knife for images, which allows you to include fairly arbitrary data as tags in the image. Let's take a quick tour…

The data part of a TIFF file begins with either the characters `II` if the file is in little-endian byte order, or the characters `MM` if it's in big-endian byte order. This is followed by a `short` containing the "arbitrary but carefully chosen" value of `42`. Finally, we have a `long` specifying the offset of the first image file directory.

An image file directory contains a set of tags. It begins with a `short` containing the count of entries in the directory, followed by the entries themselves. Each entry contains a `short` with a tag identifier, a `short` with the tag type, a `long` count for the tag and a `long` containing the tag value or a pointer to the tag data. For example, our image file entry could consist of the following (in little-endian order):

```
0f 01 02 00 12 00 00 00 6e 00 00 00
```

In other words, tag identifier `0x10f`, of type `2` (that is, ASCII), with count `0x12` (that is, 18 bytes long), at offset `0x6e` into the data. Sure enough, in our example file, 110 bytes (that is, `0x6e`) from the beginning of the TIFF data, we have 18 bytes (including the trailing `NUL`) reading "Minolta Co., Ltd." The image file directory (IFD) ends with a `long` representing the offset to the next IFD.

> TIFF is a kind of Swiss Army Knife for images, which allows you to include fairly arbitrary data as tags in the image.

Organizing the Photos

We now know, in principle at least, how to dig out that date-and-time stamp inside the picture file. Given that date and time, we can write some software to rename those annoying file names, such as `P0000234.JPG`, to something more useful like `1999:07:26/14:33:16`, which will at least allow us to file the pictures by their date and time.

We should be stuck at this point. We don't know what the tag names are because we don't have a copy of the Exif specification. However, we do have a shortcut, because we can find software on the Web that reads TIFF and Exif files. Sometimes, an implementation is as good as a specification. (And we're lazy, so we'd just as soon find an implementation because it saves us rewriting code.)

We succeeded in our search when we discovered the gPhoto Web page, `http://gphoto.fix.no/`. The GNU digital camera application, gPhoto, comprises a suite of software for

everything from loading images off the camera–we hadn't found this package yet and so had to use Windows software for that–to plug-ins for GIMP, the GNU image manipulation program. But for our purposes, it has a frighteningly useful little Perl module, `exif.pm`, that handles the processing of tags from Exif files. From there, we can construct a Perl script to do the renaming that we described above.

We begin with our usual setup:

```
#!/usr/local/bin/perl -w
#  $Id: file-pix.pl,v ...

## rename Exif files from digital camera
## based on the date and time the picture
## was taken

use POSIX;
use exif;
use strict;

sub usage {
   die "Usage: $0 [files...]\n";
}
```

Obviously, we need to remember to include the `exif` module we described earlier and to provide for a usage message.

Most of the work will be done in the `rename_picture` subroutine, which we'll build next:

```
# Extract the photo time, and rename the file
#  based on that time.
sub rename_picture {
   my $name = $_;
   my %tags = exif::dump_jpeg($name);
   my $ptime = "";   # time inside the photo
   $ptime = $tags{$exif::datetimeoriginal}{value}
      if($tags{$exif::datetimeoriginal}{count});
```

We use the `dump_jpeg` interface in the `exif` module to populate the associative array `tags` of TIFF tags from the file.

Given that array, we can get the time and date of the original picture. The `exif` module lets us know what tags have been used with a `count` for each possible tag; we use that information to ensure we've got a real value for the time stamp, otherwise we default to the initialized null value:

```
# now that we've got the timestamp,
#  such as 1969:07:21 00:15:23",
# we need to extract the components
my $re = "(\\d\\d\\d\\d):(\\d\\d):(\\d\\d) " .
   "(\\d\\d):(\\d\\d):(\\d\\d)";
die "bad date format in $_"
   unless $ptime =~ /$re/o;
```

We've used a shortcut above and set up a regular expression for getting the date components. We've done this mostly for typesetting purposes.

Given the time breakdown–year, month, day and so on–we can construct a date string, a time string, the new file name and a UNIX `time_t` of the timestamp. We do that next:

```
my $fdate = $1 . $2 . $3;
my $ftime = $4 . $5 . $6;
my $newname = $fdate . "/" . $ftime;
my $ctime = mktime($6, $5, $4,
   $3, $2 - 1, $1 - 1900,
   0, 0, 1);
```

We want to file the picture in a directory named for the date, `$fdate`. We don't want to create the directory if it's already created, and we want to stop if we can't create it. We use a very idiomatic Perl statement to do this:

```
mkdir $fdate, 0777 ||
   die "can't create directory $fdate"
      unless -d $fdate;
```

Once we've created the directory, we want to assure ourselves that we aren't about to overwrite an existing file. We check if the target name exists and issue a warning if it does:

```
if( -r $newname ) {
   warn "file $newname already exists";
   return;
}
```

Then we rename the old file to one with the `$ftime` for its

name, and let the user know what we're doing:

```
rename $name, $newname   or
   die "can't rename $name to $newname";
print "rename $name, $newname\n";
```

Last, we change the times in the inode for the file so that they match the actual time the photo was taken:

```
   # now re-time the file, too
   utime $ctime, $ctime, $newname;
}
```

Because this is Perl, it is also very easy to change the naming scheme: for example, giving the pictures a name consisting of the date and time all in a single, large directory. We'll leave those modifications as exercises for the reader.

To finish up, we need a "main" program, which is pretty simple:

```
### main program
if( @ARGV ) {
   for( @ARGV ) {
      rename_picture($_);
   }
} else {
   usage();
}
```

Once we had a flock of pictures on our screen, we came up against another problem.

Color Blindness

As we've mentioned before, we often get some amusement out of Copeland's color blindness. Actually, Copeland's children get more amusement:

"Hey, Daddy! What color is this?"

"Um, orange?"

"Nope! Red! Ha ha!!"

Since the last time we mentioned color blindness, we've had some correspondence with Dr. Neil Cuadra of Cuadra Associates Inc., comparing Web sites and anecdotes about missing colors.

For example, Dr. Cuadra found a Web page at `http://www.geocities.com/Heartland/8833/coloreye.html` containing a selection of the Ishihara color perception test circles. Those are the arrays of colored dots, usually very carefully printed, your ophthamologist uses to check your color vision. You look at a circle of dots, and if you have normal color vision, you see a particular number. If your color vision is lacking in the range for that chart, you see a different number. (Both Dr. Cuadra and Copeland fail to see any numbers in most of the charts.)

He also pointed out to us that the nice folks at Design Matrix, Topanga, CA, have a Web page devoted to designing for people with color-vision problems (see `http://www.designmatrix.com/pl/cyberpl/cftcb.html`).

As an aside, we'll tell you that Design Matrix principal Gary Swift is an old friend from our Interactive Systems Corp. days and originally turned us on to architect Chris Alexander's idea of design patterns. The idea is that in designing things, some patterns of design recur, so it's helpful to understand what they are and to know how to use them. Alexander described his notions in an excellent book, *A Pattern Language* (published by Oxford University Press, 1977, ISBN 0-19-501919-9). Its companion volume, *Timeless Way of Building* (Oxford University Press, 1979, ISBN 0-19-502402-8), contains several hundred patterns for residential building. It was Alexander who inspired Erich Gamma and his merry band to adopt the idea for their book, *Design Patterns: Elements of Reusable Object-Oriented Software* (published by Addison-Wesley Publishing Co., 1995, ISBN 0-201-63361-2).

In any event, on the Design Matrix site, Swift provides a set of design patterns for visual design with color blindness in mind. We found his exposition particularly useful, and wish all magazine editors and Web designers were conversant with it. (Hey, you guys at *Wired*, take note!)

After that, in our search for design notes on color blindness, we tripped over `http://www.iarchitect.com/color.htm` at the Interface Hall of Shame, an online design critique. Also, we found the Web site for Lighthouse International, which has a number of additional clues for helping your color-blind readers, `http://www.lighthouse.org/color_contrast.htm`. (Paradoxically, we were led to these pages from a reference at *Feed Magazine*, `http://www.feedmag.com`. *Feed*'s navigation bar down the left side of the page is simply a series of colored squares, with no clue as to what any of them link to.)

That's a rather reference-dense recounting of our exploration on the subject. With luck, it will give you some things to think about the next time you design a Web page. Among Western Europeans, about 8% of males are color-blind (for more details see `http://www3.ncbi.nlm.nih.gov/htbin-post/Omim/dispmim?303800`). Remember that statistic when you start to build something that depends on colors.

That's all for this time. Next month, we'll think about slides. No, not photographic slides, but an alternative to using PowerPoint from Microsoft Corp. for your next presentation.

Until then, happy trails.

Jeffrey Copeland *(*`copeland@alumni.caltech.edu`*) lives in Boulder, CO, and works at Softway Systems Inc. on UNIX internationalization. He spends his spare time rearing children, raising cats and being a thorn in the side of his local school board.*

Jeffrey S. Haemer *(*`jsh@usenix.org`*) works at QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things, like everyone else in the software industry.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work` *or alternately at* `ftp://ftp.expert.com/pub/Work`.