Archiving and accessing PDFs

LIBRARIAN

This month you'll learn how to place articles in a private PDF archive and how to use a database to access those articles at a later time.

BY MICHAEL SCHILLI

No matter whether it is an entertaining article in your local newspaper or an excellent political piece in the New Yorker, both make a good read on a rainy day, and you might like to put them in an archive for easier retrieval.

Unfortunately, these gems of journalism are not available online, since their publishers haven't taken the plunge into the Web yet. Not a good thing, as printed media take up a lot of room. But do you really want to rent a shed or fill up your bookshelves with row after row of folders?

Scanning and Compressing

The alternative is to digitize the interesting bits using a scanner, and to store them on disk as PDF files. To allow the archivist to keep track of the continually growing collection, today's Perl script, *magsafe*, manages a database of scanned media.

Two or three pages are quickly scanned using the excellent *xsane* GUI tool by the *Sane* project [2], which supports almost any known scanner via the Sane back-end. Both the Epson photo scanner and the HP All-In-One Officejet in our Perlmeister lab worked just fine on Linux.

After scanning, save the individual pages as PNG files. Selecting 200 dpi as scanner resolution should be fine for this job to keep the text legible. The *convert* utility steals a trick from [3], merging multiple pages to give you a single PDF document:

```
convert -density 200 ➥
-quality 95 \
  -resize "1600x1600>" ➥
  *.png archive.pdf
```

This command collects the **.png* files from the current directory, restricting the height and width to 1600 pixels. Smaller images are left untouched due to the > character. *convert* bundles the individual pages to create a multiple page PDF file with a resolution of 200 dpi. The PNGs are compressed to JPGs in the PDF keeping 95% of the original quality.

Into the Archive

After scanning in all eleven pages of the excellent New Yorker article "Outsourcing Torture" by Jane Mayer, and converting it to *archive.pdf*, the following call to today's script *magsafe* bundles it off into the archive:

```
magsafe -m "New Yorker" -a ➥
"Jane Mayer"
    -t "Outsourcing Torture"
    -i 2005/02/14 -p 106 -d ➥
    archive.pdf
```

This creates a record with the title of the publication ("New Yorker"), a document title ("Outsourcing Torture"), the issue ("2005/02/14"), and the starting page (106).

The data are stored in a genuine database, which supports SQL queries. We have used the database engine, *SQLite*, in this Perl column before. If you check out the "Installation" section, you will see that you simply need a few CPAN modules and the script will handle everything else.

The script does not copy the PDF document, *archive.pdf*, to the database, but to a directory, where the documents

Figure 1: Digitizing an article with xsane.

are stored under numeric filenames (000001, 000002, …). Feeding the New Yorker article to a pristine database will create a new document called "000001", holding the PDF data.

Our second record will be a piece in a local newspaper, reporting on a brawl in a London pub:

```
magsafe -m ➥
"Southwest London Inquirer"
    -t "Pint Glasses Flying Low"
    -i 2005/02 -p 4 -d ➥
    archive.pdf
```

As you can see, there is no need to have the author's name; in fact, the authors of some articles in the daily papers are unknown.

If you call *magsafe* without specifying any parameters, the script interactively prompts you for the details:

```
$ magsafe
[1] New
[2] New Yorker
[3] Southwest London Inquirer
Magazine [1]>
```

As we have input two publications previously, *magsafe* has noted the newspaper titles and now displays an enumerated menu. To add a new magazine, simply press *1* and enter the name of the publication:

```
Magazine [1]> 1
Enter Magazine Name []> The Sun
Document []> ...
```

magsafe copies the PDF document specified by the *-d* option (or entered interactively as just seen) into a hard-coded directory, assigning a serial number to the individual documents (000001, 000002, and so on) in the process, and referencing the paths to these files from the database.

Needle in a Haystack

A database gives you the advantage of being able to search the data as you see fit. For our newspaper database, which we have stored in a file called *scanned_docs.dat*, this is quite easy to do with the *sqlite3* command line tool and a smattering of good old SQL:

```
$ sqlite3 scanned_docs.dat
sqlite> SELECT * from doc where
        title like '%Pint%'
2|Pint Glasses Flying Low➥
||2|4|2005/02
CTRL-D
$
```

Now you might rightly object that the process is slightly inconvenient, so *magsafe* gives you a simplified query language.

If you use the *-s* parameter, you need to specify a search string in the following format:

```
"field:pattern field:➥
pattern ..."
```

The following line searches for articles with the word *Pint* in their titles:

```
$ magsafe -s title:Pint
"Pint Glasses Flying Low", ➥
Unknown, Southwest London ➥
Inquirer, 2005/02,
4, /DATA/DOCS/000002
```

Under the hood, *magsafe* generates an SQL query from your input, after wrapping the search string in percent characters. Let's assume you are looking for articles published in 2005, that is, with a value of "2005" in the *issue* field. In this case, you would enter:

```
magsafe -s issue:2005
```

You could additionally restrict the query to articles from the "Southwest London Inquirer":

```
magsafe -s "issue:2005 ➥
mag:Southwest"
```

Abstract Database

We have seen the *Class::DBI* database abstraction used by *magsafe* a few times in this Perl column. Today, we are going to make things even simpler. The *Class::DBI::Loader* module farther simplifies the class definitions for *Class::DBI*, by analyzing the database table layout and generating abstraction classes and their relations by reference.

The *magsafe* listing shows how this is implemented. The *Getopt::Std* module parses the command line options that *magsafe* understands. Line 29 looks for *-a*, *-m*, *-t*, and so on. The colons in the format string stipulate that a parameter must follow each option marked in this way. *getopts* stores the values in a hash, *%o*, which is processed later.

Line 33 checks if the document directory exists (it can be empty), and is writable. If not, the directory needs to be created before the program is launched.

The *db_init()* function called in line 36 makes sure that users will not need to concern themselves with the database details. If the database does not exist, *db_init()* uses a few lines of SQL, starting in line 147, to create a new database with two tables. Figure 2 shows you the schema.

The *doc* table contains a line for each document you save. The line includes the title and author of the article, the issue and page, and the name of the publication from which the article was taken. As most people do not read too many different publications, but do so regularly every month, it would be poor database design to store the full title of the publication in each line. Instead the *doc* table has a *mag* field with a numerical ID that points to a line in the *mag* table, which has the ID and the full name of the publication.

The table definitions that follow as of line 147 in *magsafe*, honor the SQL standard, including the

```
mag INT REFERENCES mag,
```

line in the *doc* table, which looks slightly unusual. This is a way of saying that the *mag* column points to the *mag* table, and of establishing the relationship between the publication ID and title.

The first two columns in both tables are numerical Ids, which are tagged as *primary keys*. The *Class::DBI::Loader* class constructor called in line 40 needs them to assign unique IDs to the objects that make up the lines in the table. Newly created rows are automatically assigned new IDs by incrementing the last used ID by one.

The following line

```
namespace => "Scanned::DB"
```

in the database loader constructor call ensures that the table abstraction classes are visible in Perl in the *Scanned::DB* namespace.

After calling *Class::DBI::Loader->new()*, the *find_class()* method fetches the objects that represent the tables, as seen in lines 50 and 52, providing the table names as arguments. *$docdb*, an object of the *Scanned::DB::Doc* type, points to the *doc* document table, and *$magdb* (*Scanned::DB::Mag*) points to the publication table, *mag*.

To allow the objects to perform both *Class::DBI* standard queries, and handle more complicated WHERE conditions, the *additional_classes* parameter in line 43 requests the *Class::DBI::AbstractSearch* package.

The *relationships* flag in line 46 tells *Class::DBI::Loader* to analyze the relationships between the *doc* and *mag* tables and to link both of them. Thanks to the *REFERENCES* condition in SQL, it understands that the *mag* column in the *doc* table is simply a *foreign key*, used to perform a *JOIN* with the *mag* table.

Command-Line Controls

Starting in line 56, the script processes command line parameters. If *-s* is not set, the user is not searching for an exist-

Listing 1: magsafe

```
001 #!/usr/bin/perl -w
002 ##############################
003 # magsafe - Archive
004 #           magazine articles
005 # Mike Schilli, 2005
006 # (m@perlmeister.com)
007 ##############################
008 use strict;
009
010 use DBI;
011 use Class::DBI::Loader;
012 use Sysadm::Install qw(:all);
013 use Getopt::Std;
014 use Text::Iconv;
015
016 my $DB_NAME =
017    "/DATA/scanned_docs.dat";
018 my $DSN =
019   "dbi:SQLite:$DB_NAME";
020 my $UTF8_TERM = 1;
021
022 my $cv =
023   Text::Iconv->new("Latin1",
024                    "utf8");
025 $cv->raise_error(1);
026
027 my $DOC_DIR = "/DATA/DOCS";
028
029 getopts("a:m:t:i:p:d:s:",
030   \my %o);
031
032 die "$DOC_DIR not ready"
033   if !-d $DOC_DIR
034   or !-w $DOC_DIR;
035
036 db_init($DSN)
037   unless -e $DB_NAME;
038
039 my $loader =
040   Class::DBI::Loader->new(
041   dsn       => $DSN,
042   namespace => "Scanned::DB",
043   additional_classes => qw(
044    Class::DBI::AbstractSearch
045   ),
046   relationships => 1,
047   );
048
049 my $docdb =
050   $loader->find_class("doc");
051 my $magdb =
052   $loader->find_class("mag");
053
054 my @objs = ();
055
056 if (!exists $o{s}) {
057   my $mag =
058     mag_pick($magdb, $o{m});
059   my $doc = $o{d}
060     || ask "Document", "";
061   my $author = $o{a} || "";
062   my $title = $o{t}
063     || ask "Title", "";
064   my $page = $o{p}
065     || ask "Page", "";
066   my $issue = $o{i}
067     || ask "Issue", "";
068
069   my $id = $mag->add_to_docs(
070     {
071       map {
072         $UTF8_TERM
073           ? $_
074           : $cv->convert($_)
075       }
076       title => $title,
077       page   => $page,
078       issue  => $issue,
079       author => $author
080     }
081   );
082
083   cp $doc, docpath($id);
084   exit 0;
085 }
086
087 my %search = ();
088
089 for (split ' ', $o{s}) {
090
091   my ($field, $expr) =
092     split /:/, $_;
093
094   if ($field eq "mag") {
095     my @mags =
096       $magdb->search_like(
097           name => "%$expr%");
098
099     $search{$field} =
```

ing database entry, but would like to enter a new row. Lines 59 through 67 accept values from various command line options. If one or more options are not set, the *ask()* function (injected by the *Sysadm::Install* module) interactively prompts the user for the missing values. The author field is optional and is not prompted for.

Selecting a publication is slightly more complex and uses the *mag_pick* function defined in line 169 ff. The *retrieve_all()* method of the *$magsdb* database table object reads all existing rows in the *mag* table. The returned data objects provide methods for accessing the individual fields in the records: for example, *$obj->name()* returns the name of the publication, which is stored in the *name* column of the *mag* table.

If *$picked* is not set, that is, the command line option for the publication name is empty, line 179 uses the *pick()* function (also from *Sysadm::Install*) to give the user a menu based choice of magazine names. If the user selects the first entry, *New*, line 181 invalidates the selection, and line 186 allows the user to enter the name of a new publication, which will be displayed in the selection list next time. The *find_or_create()* method then creates a new entry in the *mag* table, or finds an existing magazine entry.

The *$mag* variable in line 57 thus represents a newly created or existing magazine. As *Class::DBI::Loader* has done a good job previously, and analyzed the relationships between the *doc* and *mag* tables, thanks to the *relationships* flags, line 69 can now simply call *$mag->add_to_docs()* to add an article to the *doc* table and allow its *mag* column to point to a publication in the *mag* table. The *add_to_docs()* method of

Listing 1: magsafe

```
100       [ map { $_->id() }
101         @mags ];
102   } else {
103     $search{$field} =
104                     "%$expr%";
105   }
106 }
107
108 @objs = $docdb->search_where(
109   \%search, { cmp => "like" }
110 );
111
112 if (@objs) {
113   print join(", ",
114     '"' . $_->title() . '"',
115     $_->author()
116       || "Unknown",
117     $_->mag()->name(),
118     $_->issue(),
119     $_->page(),
120     docpath($_->docid())),
121     "\n"
122     for @objs;
123 } else {
124   print STDERR
125     "No entries\n";
126 }
127
128 ##################################
129 sub docpath {
130 ##################################
131   my ($id) = @_;
132
133   return sprintf "%s/%06d",
134     $DOC_DIR, $id;
135 }
136
137 ##################################
138 sub db_init {
139 ##################################
140   my ($dsn) = @_;
141
142   my $dbh =
143     DBI->connect($dsn, "",
144     "");
145
146   $dbh->do( q{
147   CREATE TABLE doc (
148       docid  INTEGER
149              PRIMARY KEY,
150       title  VARCHAR(255),
151       author VARCHAR(255),
152       mag    INT REFERENCES
153              mag,
154       page   INT,
155       issue  VARCHAR(32)
156   );
157   } );
158
159   $dbh->do( q{
160   CREATE TABLE mag (
161       magid  INTEGER
162              PRIMARY KEY,
163       name   VARCHAR(255)
164   );
165   } );
166 }
167
168 ##################################
169 sub mag_pick {
170 ##################################
171   my ($magsdb, $picked) = @_;
172
173   my @mags =
174     map { $_->name() }
175      $magsdb->retrieve_all();
176
177   if (@mags and !$picked) {
178     $picked =
179       pick "Magazine",
180       [ "New", @mags ], 1;
181     undef $picked
182       if $picked eq "New";
183   }
184
185   if (!$picked) {
186     $picked = ask
187       "Enter Magazine Name",
188       "";
189   }
190
191   $picked = $UTF8_TERM
192     ? $picked
193     : $cv->convert($picked);
194
195   my $mag =
196     $magsdb->find_or_create(
197     { name => $picked });
198
199   return $mag;
200 }
```

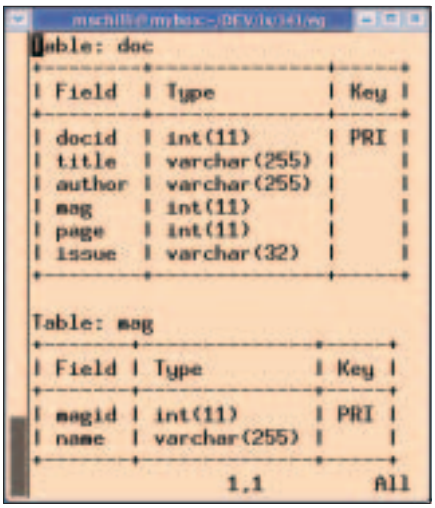

Figure 2: Layout of the SQLite database.

the magazine object is not explicitly defined in *magsafe*. It is automatically generated by the DBI::Class database abstraction layer.

To copy the current PDF document to the document directory, *magsafe* calls the *cp* function from Sysadm::Install in line 83. The *docpath()* function defined in line 129 returns the complete future path of the file. To do so, it simply converts the ID passed to it into a six-digit integer padded with zeros and prepends the document directory path.

Simpler than SQL

In case of a search request, line 89 iterates over the *column:pattern* pairs, which are provided by the *-s* Option and separated by blanks. Line 92 then separates the column name from the search key. If the column name is *mag*, line 96 first searches for a matching magazine by wrapping the search key in percentage characters and calls *search_like()* on the *mag* table. This can return no magazine objects, one, or multiple objects in *@mags*. The *id()* method finds matching magazine IDs, allowing line 99 to store a tuple of

```
"mag" => [$id1, $id2, ...]
```

in the *%search* hash. *$id1*, *$id2*, and so on are the numerical IDs for the magazines that match the search request, and the hash entry assigned to the key "*mag*" points to an array which contains these IDs as elements.

On the other hand, if a user searches for any other column than *mag*, the *else* branch in line 103 is enabled and the search key is wrapped in percent signs and assigned to the column name key in the *%search* hash.

The content of the *%search* hash is exactly what the *search_where()* method expects; it is called in line 108 and returns lines that match *all* the conditions specified by the hash. As the compare parameter, *cmp*, is set to "*like*" in an additional options hash, *search_where()* does not search for literal matches, but for patterns with wildcards given as *%*, compliant with the SQL standard.

The *print* command in line 113 is called for each object found, triggered by the subsequent *for @objs* clause.

Non-Standard Characters

SQLite3 expects all strings in UTF-8. In case you are storing foreign titles with accented characters, providing them in ISO 8859-1 won't work. Many newer Linux distributions have UTF-8 terminals, whereas older distributions typically use ISO 8859-1. Check your *LANG* environmental variable to find out: if this is set to *en_US.UTF-8*, for example, your terminal uses UTF-8.

If the *$UTF8_TERM* variable in line 20 is set to a valid value, the script will interpret all user input as UTF-8 and not attempt to convert. If *$UTF8_TERM* is 0, *magsafe* will assume user input is ISO 8859-1 and convert everything to UTF-8 before placing input in the database.

If it needs to convert, *magsafe* will resort to the CPAN *Iconv* module. Line 23 creates a *Text::Iconv* object to convert from ISO-8859-1 to UTF-8. It additionally calls *raise_error()* with a value of 1, to force an exception in case an error occurs. The *convert()* method converts any strings passed to it from one encoding type to the other. The *Encode* module for those of you who have Perl 5.8.x or later is a possible alternative.

Installation

The script needs the *DBI*, *Class::DBI*, *Class::DBI::SQLite*, *Class::DBI::AbstractSearch*, and *DBD::SQLite* modules for the abstraction, all of which are available from CPAN. Additionally, *Text::Iconv* and *Sysadm::Install* are required.

If you prefer to use the *sqlite3* command line client to mess around with the SQL database manually, download the source tarball from [4], build the source code, and install from there.

Sqlite versions 1 or 2 will not work, as *DBD::SQLite* is currently based on sqlite 3.x, which is not compatible with databases built using other SQlite versions.

Caution: If you are using SQLite databases for other applications with earlier versions of the *DBD::SQLite* module and want to go on using them, you will need to convert to the new SQLite format before upgrading:

```
sqlite OLD.DB .dump | ➥
sqlite3 NEW.DB
```

Once you install the latest version of the *DBD::SQLite* module, you will not be able to read databases created with earlier versions.

The *$DOC_DIR* variable defines the document directory in line 27 of the *magsafe* script. Additionally, the variable *$DB_NAME* in line 16 sets the name of the SQLite database file. Its directory should exist before you launch the program and it must be writable.

Now, printed media aficionados can scan interesting articles from publications they have read, and then put the printed mags in the recycling bin. Just think of what you can do with all the space that saves you. ■

Copyright Matters

This article is intended to demonstrate techniques related to storing and retrieving PDFs. The assumption is that you own a copy of the magazine you are scanning. The article also assumes that the PDF is made for personal use and will not be distributed. Laws related to scanning copyrighted material are complex and vary from country to country. For instance, some legal systems place a limitation on the number or articles you can copy from a single issue. Before you employ these techniques with copyrighted material, you should investigate these copyright issues as they apply to your situation.

INFO

[1] Listing for this article: *http://www.linux-magazine.com/Magazine/Downloads/55/Perl*

[2] *http://www.sane-project.org*

[3] *PDF Hacks*, by Sid Steward, O'Reilly 2004

[4] SQLite Home Page, *http://www.sqlite.org*