A Perl script catalogs books and CDs with the help of barcodes

CHECKOUT

Barcodes efficiently speed us through supermarket checkout lines, but the technology is also useful for totally different applications. An inexpensive barcode scanner can help you organize your private library, CD, or DVD collection. **BY MICHAEL SCHILLI**

Dealextreme.com, a company from Hong Kong, offers all kinds of inexpensive goodies. Customers can pay with PayPal, and shipping is free. Interested in a laser pointer for less than two dollars or a SATA/IDE adapter for just eight dollars? If you don't mind waiting up to two weeks for delivery, you're guaranteed to find a bargain with Dealextreme.

For quite a while I had my eye on the CCD-based barcode scanner for US$ 42 dollars (see Figure 1) (one of Dealextreme's most expensive products [2]) before I finally pressed the *Buy* button.

Mail from Hong Kong

When the mailman finally delivered the package, I could hardly wait to get started. The obvious choice was to write an application to scan the barcodes in my extensive collection of technical literature and store the results in a database. Depending on where the book comes from, the barcodes are either printed in UPC (Universal Product Number) or EAN (European Article Number) format, and Amazon.com offers a free web service that gives you detailed product information if you submit either barcode. This means that a Perl script can easily identify the author and title of a book or the artist for a CD that you scan. The data returned by the service includes CD and book cover images. Adding a graphical user interface to the application lets me display the book cover or CD case onscreen and in color after scanning.

The reader has a USB connector, and Linux immediately identifies it as a second keyboard. If you hold the sensor over the barcode of a book, CD, or DVD and press the button, the scanner switches the red light on (Figure 2) and enables the CCD sensor; then, the internal CPU starts to analyze the bars of different thicknesses to discover the encoded number.

The barcode scanner is very reliable; it beeps when it's done and sends the numbers to the computer's USB port, just as if the user had typed them at the keyboard and then pressed *Return*.

If the scanner fails to identify the barcode, which did not happen in my experiments, the user can still type the number in the input box of the application discussed in this article – the effect is the same.

Full-Color Image Included

The script, *upcscan* (Listing 1), uses a graphical interface based on the Tk toolkit. The GUI's text input box immediately grabs the keyboard focus when launched. Once the barcode scanner has identified a code, the numbers appear in the input box. The scanner sends a return key code when it is done, and the GUI responds with the callback function *scan_done()*. The function sends the barcode to the Amazon Web Services (AWS) and, after a delay of about a second, receives the title and author or artist of the book, CD, or DVD plus a URL

Figure 1: The barcode scanner can read the UPC or EAN code on a book.

Figure 2: The scanner has a CCD sensor and switches its red diodes on at the push of a button.

that points to a JPEG image of the book or CD cover.

Figure 3 shows the application shortly after scanning the barcode printed on the back of a JavaScript book. The data fields are filled in correctly, and the program has received the right book cover from Amazon in return. Figure 4 shows the results after scanning a CD by Beach Boys vocalist Brian Wilson. In both cases, the script drops the retrieved data into a local SQLite database, which I can then browse to my heart's desire and write applications around (Figure 5).

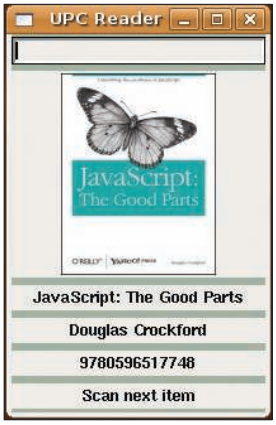

Figure 3: The script has sent the scanned code to Amazon.com and received the product data in return.

Keep on Ticking

Thanks to the Tk package from CPAN, slick GUIs are no problem for Perl scripts. Unfortunately, I had a problem with the application I had in mind: Longer operations, such as the web requests, caused the interface to freeze. Querying Amazon with a barcode can take a couple of seconds, and the interface would freeze in the meantime.

The POE module, also from CPAN, gave me a workaround – it lets the GUI run in an event-oriented userspace "kernel." However, don't confuse this with the Linux kernel; in POE, "kernel" is just a fancy name for an event loop. It provides mechanisms for cooperative multitasking.

The script does not handle web requests synchronously in this environment; instead, it sends a request to the web server and then immediately hands control back to the POE kernel and, therefore, the GUI event loop. When the response comes back from the Internet, the kernel wakes up the waiting task and passes in the data.

The *Net::Amazon* CPAN module handles communications with Amazon and supports a variety of requests to the giant retailer's web service. Internally, it does not use the asynchronous POE module to query the Amazon database. Instead, it uses the synchronous *LWP::UserAgent*. You can use the *ua* parameter to tell the module to work with a user agent that you pass in to it.

CPAN has *LWP::UserAgent::POE*, an agent with an LWP interface that respects the special asynchronous needs of the POE kernel. While the module issues web requests, and seemingly waits synchronously for the results, some black magic inside the module allows the POE kernel to keep on ticking, thus giving other tasks their turn.

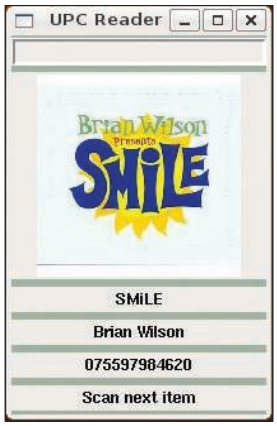

Figure 4: The scanner identifies CDs and DVDs by interpreting the UPC code and retrieves the product data and cover image from Amazon.com.

Off to the Base

The *upcscan* program uses the CPAN *Rose::DB* database wrapper to identify the database schema and insert new records into the *articles* table (Figure 6). Line 23 identifies the database file *articles.dat* in the current directory as an SQLite database, and the subsequent *AutoCommit* and *RaiseError* options ensure that new entries are written to the database without an explicit *commit* command and that any error that occurs immediately throws an exception.

The *make_classes()* method in line 27 imports the database objects into the script code, and because of this, I can simply say *Article->new()* later on to prepare a new entry in the *articles* database table.

Widgets in the POE Flow

The graphical interface is running in the main window, *$top*, which line 29 accepts from the *$poe_main_window* variable, which is exported from POE. The GUI's event loop is not left to its own devices in the script but needs to cooperates with POE's kernel.

If *use Tk* appears before *use POE* in the code, POE knows that it has to prepare an event loop for the Tk GUI; it initializes the main window and creates a pointer to it in *$poe_main_window*.

The *configure()* command in line 30 stores the *UPC Reader* title string in the window header and sets the background color for the GUI to *#a2b2a3* (light olive green).

At the top of the window, you can see the entry widget *$entry*, which accepts sequences of numbers from the scanner and stores them in the global variable *$UPC_VAR*. Farther down is a photo widget for the book and CD covers; for organizational reasons, it is embedded in a label widget. Four *Label*-type widgets follow that store the title (*$PRODUCT*), the author/artist (*$BYWHO*), the scanned UPC or EAN code (*$UPC*), and a status message (*$FOOTER*).

The *for* loop in line 50ff. drops the widgets, top down, into the main window and calls *-fill => "x"* and *-expand => 1* to ensure that the labels fill the horizontal space up to the border and automatically scale when the main window is expanded.

The *bind()* command in line 56 plays an important role. The entry widget ignores the Return character sent by the scanner because the input field is a single-line field. The *bind* method binds the Return key's keyboard code to the *scan_done()* function defined in line 68ff. to

Listing 1: upcscan

```
001 #!/usr/local/bin/perl -w
002 ########################################
003 # upcscan - Scan/store UPC coded articles
004 # Mike Schilli, 2008 (m@perlmeister.com)
005 ########################################
006 use strict;
007 use Tk;
008 use Tk::JPEG;
009 use POE;
010 use LWP::UserAgent::POE;
011 use Net::Amazon;
012 use Net::Amazon::Request::UPC;
013 use MIME::Base64;
014 use Rose::DB::Object::Loader;
015 use Log::Log4perl qw(:easy);
016
017 my @MODES = qw(books music dvd);
018
019 my $UA = LWP::UserAgent::POE->new();
020
021 my $loader = Rose::DB::Object::Loader->new(
022   db_dsn =>
023     "dbi:SQLite:dbname=articles.dat",
024   db_options   => {
025     AutoCommit => 1, RaiseError => 1 },
026 );
027 $loader->make_classes();
028
029 my $top = $poe_main_window;
030 $top->configure(-title     => "UPC Reader",
031                 -background=> "#a2b2a3");
032 $top->geometry("200x300");
033
034 my $FOOTER = $top->Label();
035 $FOOTER->configure(-text =>
036                    "Scan next item");
037
038 my $BYWHO = $top->Label();
039 my $UPC   = $top->Label();
040 my $PHOTO = $top->Photo(-format => 'jpeg');
041 my $photolabel =
042             $top->Label(-image => $PHOTO);
043 my $entry = $top->Entry(
044             -textvariable => \my $UPC_VAR);
045
046 my $PRODUCT = $top->Label();
047
048 $entry->focus();
049
050 for my $w ($entry, $photolabel, $PRODUCT,
051            $BYWHO, $UPC, $FOOTER) {
052   $w->pack(-side => 'top', -expand => 1,
053            -fill => "x" );
054 }
055
056 $entry->bind("<Return>", \&scan_done);
057
058 my $session = POE::Session->create(
059   inline_states => {
060     _start => sub{
061       $poe_kernel->delay("_start", 60);
062   }
063 });
064
065 POE::Kernel->run();
066
067 ########################################
068 sub scan_done {
069 ########################################
070   $PHOTO->blank();
071   $PRODUCT->configure(-text => "");
072   $FOOTER->configure(-text =>
073                      "Processing ...");
074   $BYWHO->configure(-text => "");
075   $UPC->configure(-text => $UPC_VAR);
076   resp_process(
077           amzn_fetch( $UPC_VAR ) );
078   $UPC_VAR = "";
079 }
080
081 ########################################
082 sub amzn_fetch {
083 ########################################
084   my($upc) = @_;
```

trigger processing of the code read in by the scanner.

First, the photo object's *blank()* method is called to remove the previous cover image. Then the title and author/artist names are replaced with blank strings. A "*Processing ...*" message appears in *$FOOTER*, and the request is sent to Amazon by calling *amzn_fetch()*. To get ready for the next scan, line 78 immediately deletes the code read by the scanner from the entry widget. The barcode for the current product is stored in the *$UPC* widget.

After completing these preparations, line 58 defines a POE session, and line 65 launches the POE kernel, which controls the program from this point on until it is terminated. The POE kernel accepts user input, mouse clicks, or keyboard/scanner input and makes sure that a time slot is assigned to each handler triggered by an event. POE is very strict with the sessions it controls. If they do not have a task to complete, they are eliminated. POE does not understand that a Tk application might just be waiting for user input. To prevent it from killing the GUI, lines 58 through 63 define a session that jumps to the *_start* event every 60 seconds.

Ask Amazon

Once the scanner has returned a UPC or EAN code, the *amzn_fetch()* function in line 82ff. creates an instance of a

Listing 1: upcscan

```
085
086   my $resp;
087
088   my $amzn = Net::Amazon->new(
089       token => 'XXXXXXXXXXXXXXXXXXXX',
090       ua    => $UA,
091   );
092
093   for my $mode (@MODES) {
094
095     my $req =
096       Net::Amazon::Request::UPC->new(
097           upc  => $upc,
098           mode => $mode,
099       );
100
101     $resp = $amzn->request($req);
102
103     if($resp->is_success()) {
104         return($resp, $mode, $upc);
105         last;
106     }
107
108     WARN "Nothing found in mode '$mode'";
109   }
110   return $resp;
111 }
112
113 ########################################
114 sub resp_process {
115 ########################################
116   my($resp, $mode, $upc) = @_;
117
118   if($resp->is_error()) {
119     $PRODUCT->configure(
120                 -text => "NOT FOUND");
121     return 0;
122   }
123
124   my ($property) = $resp->properties();
125   my $imgurl = $property->ImageUrlMedium();
126   img_display( $imgurl );
127
128   my $a = Article->new();
129   $a->upc($upc);
130   $a->type($mode);
131   $a->title( $property->Title() );
132
133   if($mode eq "books") {
134     $a->bywho( $property->author() );
135   } elsif( $mode eq "music") {
136     $a->bywho( $property->artist() );
137   } else {
138     $a->bywho( "" );
139   }
140
141   $BYWHO->configure(-text => $a->bywho() );
142   $PRODUCT->configure(
143                   -text => $a->title() );
144
145   if($a->load( speculative => 1 )) {
146       $PRODUCT->configure(
147                 -text => "ALREADY EXISTS");
148   } else {
149     $a->save();
150   }
151
152   $FOOTER->configure(
153                 -text => "Scan next item");
154   return 1;
155 }
156
157 ########################################
158 sub img_display {
159 ########################################
160   my($imgurl) = @_;
161
162   my $imgresp = $UA->get( $imgurl );
163
164   if($imgresp->is_success()) {
165     $PHOTO->configure( -data =>
166       encode_base64( $imgresp->content() ));
167   }
168 }
```

```
$ sqlite3 articles.dat
SQLite version 3.4.2
Enter ".help" for instructions
sqlite> .width 14 40 20
sqlite> .mode col
sqlite> .headers on
sqlite> select upc, title, bywho from articles;
upc            title                                    bywho
-------------  ---------------------------------------  --------------------
724356061125   Minimum-Maximum                          Kraftwerk
9780596009762  SQL Cookbook (Cookbooks (O'Reilly))       Anthony Molinaro
9780596517748  JavaScript: The Good Parts               Douglas Crockford
9781416553649  Born Standing Up: A Comic's Life         Steve Martin
5107597984620  SMiLE                                    Brian Wilson
sqlite>
$
```

Figure 5: The scanned article data is sent to the SQLite database.

Net::Amazon object and passes in both the developer token (you will need to apply to Amazon if you do not have one yet – see the "Installation" section below), and the global special agent *LWP::UserAgent::POE*, which not only retrieves web requests, like its base class *LWP::UserAgent*, but also collaborates with POE.

A *Net::Amazon::Request::UPC*-type request object talks to the Amazon web service that provides the UPC/EAN lookup facility. The request must state up front whether it wants to search for the UPC/EAN code in the Books, Music, or DVD sections.

Because the scanner does not know whether it is scanning a book or a CD, the *for* loop that begins in line 93 simply tries all three supported sections and terminates as soon as Amazon reports a successful search. The response returned here offers the *is_success()* method, which tells you whether a matching code has been found.

The *amzn_fetch()* function returns three parameters: the response object *$resp*, the section in which the match was found (Books, Music, or DVD), and the scanned UPC/EAN code.

The *resp_process()* function defined in line 114ff. grabs the results and uses it to update the GUI fields. The *ImageUrlMedium()* method for the retrieved article in *$property* includes a URL for a medium-sized JPEG image on Amazon's server depicting the product cover.

Reading JPEGs

To allow the Tk toolkit's photo widget to read and display JPEG images, line 8 loads the *Tk::JPEG* module, which is also part of the Tk distribution. The *img_display()* function defined in line 158ff. expects the URL for an image hosted by Amazon.com and then asks the POE-friendly user agent to retrieve it.

Because the Tk photo widget insists on Base64-encoded data (at least for JPEGs), the *encode_base64()* function from the *MIME::Base64* module converts the JPEG data extracted via *content()* into a format that the widget will handle. After this, the photo widget's *configure()* method sets the *-data* option, which in turn tells the widget to parse the data, convert it to the internal Tk format, and display the decoded image in the GUI.

Database with Wrappers

Now back to the *resp_process()* function: It not only displays the product data, but also stores it in the database. To allow this to happen, in line 128 the *Rose::DB* wrapper creates a new *Article* type object and sets the object's *upc*, *type*, *title*, and *bywho* fields, which all correspond to the database table columns with the same names.

The *load()* method with the *speculative* attribute then attempts to find a matching entry in the database. If the search succeeds, the script sends an "*ALREADY EXISTS*" message to the GUI display to tell the operator that this product has already been scanned. If *load()* fails, *save()* in line 149 saves the newly entered article in the database.

Installation

Before the script can use AWS, you must apply for an Amazon developer token and enter it in line 89. This process is fast, and you can trigger it [3] by entering a valid email address to which the token will be sent. Without a valid token, the script will say *NOT FOUND* every time. The database is created by the SQLite client *sqlite3* when the schema file (Figure 6) is passed in to it:

```
CREATE TABLE articles (
  id  INTEGER PRIMARY KEY,
  upc   TEXT,
  type  TEXT,
  title TEXT,
  bywho TEXT,
  UNIQUE(upc, type)
);
~
"schema.sql" 8L, 132C 1,1          All
```

Figure 6: The SQLite database schema.

```
sqlite3 articles.dat <schema.sql
```

The client then creates the *articles.dat* database and an empty database table, *articles*, containing the columns *id*, *upc* (the UPC/EAN code), *type* (Books, Music, or DVD), *title* (the title of the book or CD/DVD), and *bywho* (the author or artist).

The *UNIQUE* command in the database table's SQL uses the article type and UPC number as unique keys to allow the Rose database wrapper to run *load()* quickly and check to see whether the product with a given UPC and type has already been scanned.

Conclusion

All of the modules used here are available from CPAN and can be installed with a CPAN shell. The article database created by the script can be used in various ways: as a buying aid ("Do I own this book already?"), as a CD archive, or – if you include location details (as in, "Room 1, Shelf 4, Compartment 3") – as an online catalog for absent-minded librarians. ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/resources/article_code*

[2] Dealextreme: *http://www.dealextreme.com/details.dx/sku.12559*

[3] Amazon Web Services: *http://amazon.com/soap*

THE AUTHOR

Michael Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He is the author of *Goto Perl 5* (German) and *Perl Power* (English), both published by Addison-Wesley, and he can be contacted at *mschilli@perlmeister.com*. Michael's homepage is at *http://perlmeister.com*.