Vasiliy Yakobchu, Fotolia

Perl scripts watch Amazon prices

BARGAIN HUNTER

If you are a bargain hunter, you might enjoy this Perl script that monitors price developments at Amazon and alerts you if Amazon suddenly drops the prices on the products you have been watching.

BY MICHAEL SCHILLI

Should I buy the digital camera that caught my eye recently? Should I wait until the price drops? These questions are difficult to answer, but a glance at the price developments over the past few months could tell you where the prices are heading.

Price History

If Amazon offered a price history for its products – something similar to the share prices on major financial pages – customers might be annoyed about missing bargains. Or they might speculate on prices continuing to drop and wait for a more favorable opportunity. The online shopping site does not offer this service yet, so you'll have to set it up yourself.

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.

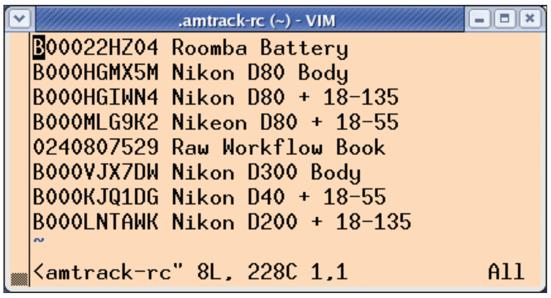
Prices in the Box

Today's script, *amtrack* [1], parses an *~/.amtrack-rc* configuration file, much like that in Figure 1, to discover the products the user wants. A cronjob calls the script at regular intervals. Each time the script connects to the Amazon web service, it queries the prices of the configured articles and stores them in a local SQLite database.

If the price of a product drops, the script sends an email with the product's URL and the current price to the address configured in line 97. All that's left for the bargain hunter to do is click the URL in the email client, take another look at the product in the browser, and maybe snap it up on the spot.

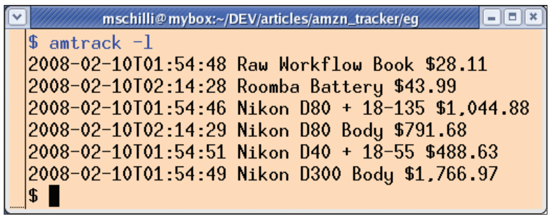
Because the prices are stored locally in a database, the script can query and display historical data at the drop of a hat. A call to *amtrack -l* returns the latest prices for all monitored products (Figure 2). If you are interested in the complete content of the database, you can set the *-a* flag, but be aware you will probably see a lot of output if you have been running the price monitor for a longer period of time and have been watching several products.

When launched without any command-line options, *amtrack* works its way through the product shortcuts de-



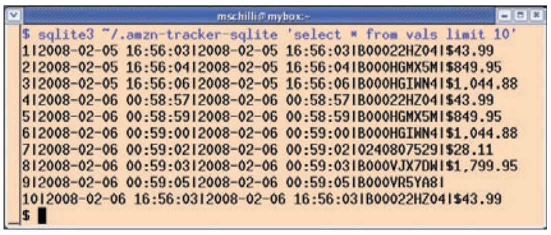
```
B00022HZ04 Roomba Battery
B000HGMX5M Nikon D80 Body
B000HGIWN4 Nikon D80 + 18-135
B000MLG9K2 Nikeon D80 + 18-55
0240807529 Raw Workflow Book
B000VJX7DW Nikon D300 Body
B000KJQ1DG Nikon D40 + 18-55
B000LNTAWK Nikon D200 + 18-135
~
<amtrack-rc" 8L, 228C 1,1            All
```

Figure 1: The ~/.amtrack-rc configuration file lists the specific products and their ASIN numbers.



```
$ amtrack -l
2008-02-10T01:54:48 Raw Workflow Book $28.11
2008-02-10T02:14:28 Roomba Battery $43.99
2008-02-10T01:54:46 Nikon D80 + 18-135 $1,044.88
2008-02-10T02:14:29 Nikon D80 Body $791.68
2008-02-10T01:54:51 Nikon D40 + 18-55 $488.63
2008-02-10T01:54:49 Nikon D300 Body $1,766.97
$
```

Figure 2: When called with the -l option, the amtrack script lists the current prices of all the products it is watching.



```
$ sqlite3 ~/.amzn-tracker-sqlite 'select * from vals limit 10'
1|2008-02-05 16:56:03|2008-02-05 16:56:03|B00022HZ04|$43.99
2|2008-02-05 16:56:04|2008-02-05 16:56:04|B000HGMX5M|$849.95
3|2008-02-05 16:56:06|2008-02-05 16:56:06|B000HGIWN4|$1,044.88
4|2008-02-06 00:58:57|2008-02-06 00:58:57|B00022HZ04|$43.99
5|2008-02-06 00:58:59|2008-02-06 00:58:59|B000HGMX5M|$849.95
6|2008-02-06 00:59:00|2008-02-06 00:59:00|B000HGIWN4|$1,044.88
7|2008-02-06 00:59:02|2008-02-06 00:59:02|0240807529|$28.11
8|2008-02-06 00:59:03|2008-02-06 00:59:03|B000VJX7DW|$1,799.95
9|2008-02-06 00:59:05|2008-02-06 00:59:05|B000VR5YA8|
10|2008-02-06 16:56:03|2008-02-06 16:56:03|B00022HZ04|$43.99
$
```

Figure 3: The SQLite database can also be queried with the sqlite3 command-line client.

fined in the ~/*.amtrack-rc* configuration file and updates the database with the latest prices.

Wish List

The configuration file has two columns. Column one contains the ASIN number for the product in question, with a short product description to its right, separated by one or multiple blanks. This doesn't make any difference to the database, but it makes the alert email easier to read.

Comment lines start with a pound sign (#), and the script ignores them, just as it ignores blank lines. The *config_read()* function (Listing 1, lines 113-136) loads the configuration and returns two references: one to the ordered Array *@config*, and one to the *%config* hash. The array contains pairs of ASIN and text values, whereas the hash directly maps ASINs to texts for quick look-ups.

Bargain Basement

The CPAN *Net::Amazon* module provides an object-oriented interface to Amazon's REST-based web service. If you enter the ASIN number for a product, the module contacts Amazon and retrieves the price.

When it started, Amazon only sold books, which can be uniquely identified by their ISBN numbers. As the product portfolio grew, Amazon added the ASIN number, which has a similar structure but also includes letters and is thus capable of addressing far more products.

The *request()* method of the *Net::Amazon* class accepts a *Net::Amazon::Re-quest::ASIN* object with parameters that include the ASIN number for a product. After doing so, it handles communications with the Amazon web server and returns a *Net::Amazon::Response::ASIN* class object. The object's *is_success()* property tells you whether the request has worked. If so, the *properties()* method returns a single *Net::Amazon::Property* class object that contains the matching product, including a product description, customer ratings, URLs to images, and much more, including the price. Amazon offers different search options – by author, for example – so *properties()* can also return multiple entries. The *OurPrice()* method of a property returns the current price of the product formatted $X.XX, £X.XX (for the UK), or EUR X,XX (for other European locations; see below).

Historical Cache

The script also relies on the CPAN *Cache::Historical* module, which not only stores data under a primary index, like any normal cache, but also inserts a date that it then uses as a secondary index. The script stores the product prices, with the ASIN as the primary index, and saves the retrieval date to the cache. Under the hood, *Cache::Historical* relies on the file-based SQLite database, which the module lists as a requirement and which it also installs thanks to the CPAN shell. The *new()* constructor's *sqlite_file* parameter sets the name for the ~/*.amzn-tracker-sqlite* file into which the database is dropped. If you like, you can query the SQLite database with the *sqlite3* client program to view the data, as shown in Figure 3.

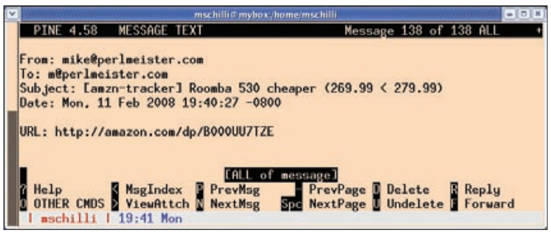
The *get_interpolated()* call for the cache retrieves the database value for a specified date (the current date in the script) and a specific key (the ASIN for a product). The script stores this in the *$last_price* variable and then updates the database with the latest value from the Amazon website. After doing so, it retrieves the current price from the database again and compares it with the *$last_price*.

In contrast, the *values()* method returns a list of pairs of values that match the specified key. Each pair is a reference to an array that contains the date as a *DateTime* object and the price.

Do What I Mean

If the current price of a monitored product is lower than the last price stored in the database, the script dispatches an email in line 96 (Figure 4). Although many CPAN modules can send email, *Mail::DWIM* (Do What I Mean) is one of the simplest: It exports the *mail()* function, which accepts a recipient, a subject line, and the body text of the mail as parameters. It sets meaningful defaults for the remaining parameters, such as the sender or the mail transport (in this case, the active user plus the configured domain and the active Sendmail daemon). For other mail transport mechanisms, it also supports SMTP with a mail host specification. These defaults are set as parameters in a local *.maildwim* file. For more details about this, just read the *Mail::DWIM* man page.

The email also contains the URL for the product, which is achieved by add-



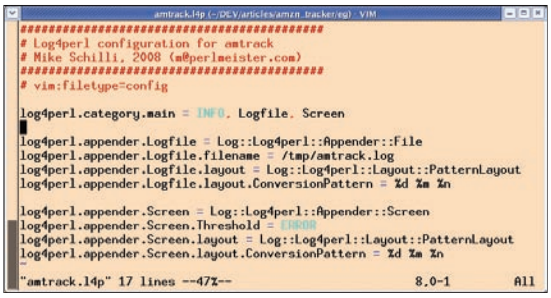
```
  PINE 4.58   MESSAGE TEXT                 Message 138 of 138 ALL

From: mike@perlmeister.com
To: m@perlmeister.com
Subject: [amzn-tracker] Roomba 530 cheaper (269.99 < 279.99)
Date: Mon, 11 Feb 2008 19:40:27 -0800

URL: http://amazon.com/dp/B000UU7TZE


                              [ALL of message]
? Help        < MsgIndex  P PrevMsg     - PrevPage D Delete    R Reply
O OTHER CMDS  > ViewAttch N NextMsg   Spc NextPage U Undelete  F Forward
 | mschilli | 19:41 Mon
```

Figure 4: An email arrives announcing that the price of the Roomba [4] vacuum cleaner robot has dropped by US$ 10.



```
###########################################
# Log4perl configuration for amtrack
# Mike Schilli, 2008 (m@perlmeister.com)
###########################################
# vim:filetype=config

log4perl.category.main = INFO, Logfile, Screen

log4perl.appender.Logfile = Log::Log4perl::Appender::File
log4perl.appender.Logfile.filename = /tmp/amtrack.log
log4perl.appender.Logfile.layout = Log::Log4perl::Layout::PatternLayout
log4perl.appender.Logfile.layout.ConversionPattern = %d %m %n

log4perl.appender.Screen = Log::Log4perl::Appender::Screen
log4perl.appender.Screen.Threshold = ERROR
log4perl.appender.Screen.layout = Log::Log4perl::Layout::PatternLayout
log4perl.appender.Screen.layout.ConversionPattern = %d %m %n
~
"amtrack.l4p" 17 lines --47%--                       8,0-1         All
```

Figure 5: The Log4perl configuration for the script.

ing */dp/$asin* to the base URL for the Amazon website.

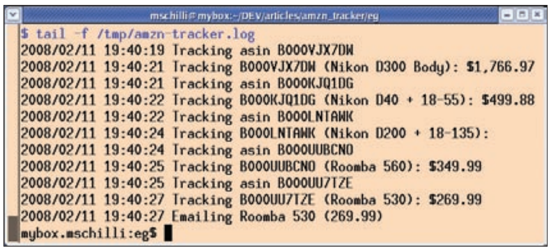
Professional Logging

To keep the user current with what the script is doing, it uses Log4perl to log its activities. The *amtrack.l4p* file, which initializes Log4perl, is stored in the same directory as the script (Figure 5). To allow the script to find the configuration file, even if called from a different directory (e.g., as *bin/amtrack* or as *amtrack* from the home directory), the *FindBin* module helps by exporting the *$Bin* variable as the directory the script was found in, thus making sure that *$Bin/amtrack.l4p* will represent the absolute path to the Log4perl configuration.

The Log4perl configuration is not exactly easy; after all, you want the script to write regular activities to the logfile (Figure 6) but display errors at the console.

A cronjob called at regular intervals will just keep appending to the logfile (by default), but it will send errors (such as a failed network connection) to *STDERR* and thus cause cron, which launched the script, to mail the admin.

A single logger is defined for the *main* category (i.e., for the main program). *Net::Amazon* is also Log4perl-enabled, and another entry in the configuration file would quickly send details of communications with the Amazon web server to the screen. The *main* logger controls two appenders: *Logfile* and *Screen*. To make sure that *Screen* only receives messages with *ERROR* priority or higher, the line



```
$ tail -f /tmp/amzn-tracker.log
2008/02/11 19:40:19 Tracking asin B000VJX7DW
2008/02/11 19:40:21 Tracking B000VJX7DW (Nikon D300 Body): $1,766.97
2008/02/11 19:40:21 Tracking asin B000KJQ1DG
2008/02/11 19:40:22 Tracking B000KJQ1DG (Nikon D40 + 18-55): $499.88
2008/02/11 19:40:22 Tracking asin B000LNTAWK
2008/02/11 19:40:24 Tracking B000LNTAWK (Nikon D200 + 18-135):
2008/02/11 19:40:24 Tracking asin B000UUBCNO
2008/02/11 19:40:25 Tracking B000UUBCNO (Roomba 560): $349.99
2008/02/11 19:40:25 Tracking asin B000UU7TZE
2008/02/11 19:40:27 Tracking B000UU7TZE (Roomba 530): $269.99
2008/02/11 19:40:27 Emailing Roomba 530 (269.99)
mybox.mschilli:eg$
```

Figure 6: Excerpt from the logfile after a successful script run.

Listing 1: amtrack

```
#!/usr/bin/perl -w
use strict;
use Getopt::Std;
use Net::Amazon;
use
  Net::Amazon::Request::ASIN;
use Log::Log4perl qw(:easy);
use Cache::Historical 0.02;
use DateTime;
use Mail::DWIM qw(mail);
use FindBin qw($Bin);

my ($home) = glob "~";
my $amzn_rc =
  "$home/.amtrack-rc";

Log::Log4perl->init(
 "$Bin/amtrack.l4p");

my $cache =
  Cache::Historical->new(
 sqlite_file =>
   "$home/.amtrack-sqlite");

my $UA = Net::Amazon->new(
 token => 'YOUR_AMZN_TOKEN',
   # locale => 'uk',
);

my ($config, $txt_by_asin) =
  config_read();

getopts("al", \my %opts);

if ($opts{l} or $opts{a}) {
 for my $key (
  sort keys %$txt_by_asin)
 {
  my $txt =
    $txt_by_asin->{$key};
  for my $val (
   $cache->values($key))
  {
   my ($dt, $price) = @$val;
   print "$dt $txt $price\n";
   last if $opts{l};
  }
 }
} else {
 update($config);
}

###############################
sub fix_price {
###############################
 my ($price) = @_;

 if (defined $price) {
  $price =~ s/[^\d]//g;
  $price =~ s/..$/.$&/g;
 }
 return $price;
}

###############################
sub update {
###############################
 my ($config) = @_;

 for my $line (@$config) {

  my ($asin, $txt) = @$line;
  my $now = DateTime->now();

  my $last_price = fix_price(
   $cache->get_interpolated(
    $now, $asin
   )
  );

  track($asin, $txt, $cache);

  my $price_now = fix_price(
   $cache->get_interpolated(
    $now, $asin
   )
  );

  if ( defined $last_price
   and defined $price_now)
  {

   if (
log4perl.appender.Screen.↩
Threshold = ERROR
```

sets this threshold in the appender definition.

If you would like to learn more about the Log4perl framework, check out the Log4perl homepage [2], which has exhaustive documentation and a FAQ with frequently used sample configurations.

Not Without My Token

Amazon requires a token from scripts that mess around with its web service; the token is free to anybody who registers and accepts the conditions [3]. After receiving the token, just replace *YOUR_AMZN_TOKEN* in line 26 with the correct token.

The script will work with the US website or with that of any subsidiary, such as the UK site. For the latter, you just need to uncomment *locale* = > '*uk*' in line 27.

For other European locales, prices might be displayed in the EUR X,XX format, but the *fix_price* function converts them into proper floating point format, which you can compare with the use of numeric operations.

Because US figures use both a dot as the floating point, and commas to separate thousands, *fix_price()* simply ditches everything that is not a digit and inserts a decimal point in front of the last two digits.

Installation

A CPAN shell installs the CPAN modules specified at the start of the script, immediately resolving all the dependencies at the same time.

A crontab entry of the format

```
23 0 * * * /path/to/amtrack
```

calls the script once a day at 23 minutes after midnight. This should be more than enough snapshots to keep you up to date. The *Net::Amazon* module makes sure that the script keeps to Amazon's conditions of use, and rate-limits itself if the user retrieves prices at too short an interval.

Extensions

To extend the script, you could assign a limit for each price in the configuration file and tell the script not to notify you unless the price drops below this value. Another application would be to draw a graph of price changes over an period of time; the CPAN *RRDTool::OO* or *Imager::Plot* modules are perfect for this. ■

INFO

[1] Listings for this article: *ftp://www.linux-magazin.com/pub/listings/magazine/92/Perl*

[2] Log4perl homepage: *http://log4perl.com*

[3] Amazon Web Services tokens are available from: *http://www.amazon.com/soap*

[4] The Roomba vacuum robot: *http://www.amazon.com/iRobot-Roomba-Intelligent-Floorvac-Robotic/dp/B00008439Y*

Listing 1: amtrack

```
094     $price_now < $last_price)
095    {
096     mail(
097      to => 'foo@bar.com',
098      subject => "[amtrack] "
099 ################################
100        . "$txt cheaper ("
101        . "$price_now < "
102        . "$last_price)",
103      text => "URL: "
104        . "http://amazon.com"
105        . "/dp/$asin",
106     );
107    }
108   }
109  }
110 }
111
112 ################################
113 sub config_read {
114 ################################
115
116  my @config = ();
117  my %config = ();
118
119 open AMZNRC, "$amzn_rc"
120   or die
121   "Cannot open $amzn_rc";
122 while (<AMZNRC>) {
123  s/#.*//;
124  next if /^\s*$/;
125  chomp;
126  my ($asin, $txt) =
127    split ' ', $_, 2;
128  push @config,
129    [ $asin, $txt ];
130  $config{$asin} = $txt;
131  }
132 close AMZNRC;
133
134 return \@config, \%config;
135 }
136
137 ################################
138 sub track {
139 ################################
140  my ($asin, $txt, $cache) =
141    @_;
142
143  INFO "Tracking asin $asin";
144
145  my $req =
146    Net::Amazon::Request::ASIN
147    ->new(asin => $asin);
148
149  my $resp =
150    $UA->request($req);
151
152  if ($resp->is_success()) {
153   my ($prop) =
154     $resp->properties();
155   my $price =
156     $prop->OurPrice();
157   INFO "Tracking $asin ",
158     "($txt): $price";
159   $cache->set(
160    DateTime->now(), $asin,
161    $price)
162     if $price;
163  } else {
164   ERROR
165 "Can't fetch asin $asin: ",
166     $resp->message();
167  }
168 }
```