A Perl script for Automatically Collecting News Headlines

Spot Reporter

Instead of visiting news sites periodically to pick up the latest reports, most people prefer to let a news aggregator do the job. The aggregator automatically draws your attention to incoming news. If a website does not have an RSS feed, a new Perl module simplifies the task of programming an RSS feed for private use. BY MICHAEL SCHILLI

www.sxc.hu

The sheer bulk of information on the Internet means that nobody can read it all. Visiting a couple of dozen websites a day to pick up the latest news is so time consuming that you would need to quit your job to get finished.

This prompted many news sites to introduce RSS feeds with headlines and links to articles in machine-readable format. RSS is short for *RDF Site Summary*, where *RDF* means *Resource Description Framework*. RSS files use XML – a format that so-called news aggregators can easily parse. Articles that users have not yet read are served up as clickable headlines.

This process of *Syndication*, that is, compiling and serving up messages that are available from another location, helps to manage the flood of information and saves a lot of time.

Well-known sites such as Slashdot now offer RSS feeds, which aggregators such as *Amphetadesk* ([2] and Figure 1) fetch at regular intervals if a user has subscribed to the service, that is, clicked the *Subscribe* button.

Building Your Own

Unfortunately, not all news pages have RSS feeds. Do they really think that users will stop by every day to rummage through the information they provide? The *RssMaker* module that we will be looking at in this article gives you a function that can help you generate an RSS file from a title page with headlines and URLs with about 10 lines of Perl code. If you then set up a cronjob to generate the RSS file once a day, you can hand the file to your news aggregator, which will give you the kind of extensive news coverage we have come to expect in the 21st century.

All it takes is a call to the *make* function inside the *RssMaker* Perl module, shown in Listing 1, *RssMaker.pm*. It expects a URL that points to the news site. It picks up the site off the Web, parses its HTML, and then extracts embedded links and their display text. For every instance it finds, it calls a user-definable filter function, passes it the link and its textual description, and lets it decide. If the filter function returns a *true* value, the link is a headline and gets added to the RSS overview.

Finally, *make()* sends the XML output to a file specified by the *output* parameter.

Two Date Formats

To convert the HTTP time stamp in the Web document into the ISO-8601 format

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.

Figure 1: The books.perl.org news feed in Amphetadesk.

Figure 2: the RSS validator at feeds.archive.org.

that RSS needs, the *str2time* function of the *HTTP::Date* module first scans the date string (for example, "Tue, 26 Oct 2004 05:10:08 GMT") and returns the Unix time in seconds. The *from_epoch()* function of the *DateTime* module grabs the value and generates a new *DateTime* object, which gets converted to ISO-8601 format inside double quotes ("2004-10-26T05:10:08").

Encoding

XML expects UTF-8 encoded text. UTF-8 is compatible with regular ASCII, as long as you avoid characters from the upper half of the 256 character table. This means that special characters used in some European languages can be a problem. Think German umlauts or French accented characters.

If you use any characters of the upper half with ISO-8859-1, they won't be UTF-8 compatible.

RssMaker avoids this problem by allowing developers to specify the encoding in the resulting RSS document. If the website in question has HTML encoding such as *ü* for "ü", the *HTML::TreeBuilder* will convert the extracted link texts to ISO-8859-1. The ASCII code of this character is 252.

However, if the RSS file had specified

```
<?xml version="1.0"
encoding="utf-8"?>
```

and the "ü"s were ASCII 252 encoded, this would cause a problem. Developers can specify *encoding => "iso-8859-1"* for the *make* function to write the following to the XML document:

```
<?xml version="1.0"
encoding="iso-8859-1"?>
```

and 252 encoded "ü"s are correctly interpreted by the news aggregator.

RSS Feeding

Let's put RssMaker to the test and create an RSS feed of *books.perl.org*'s great web page. The site features reviews and ratings of books on Perl, and it's interesting to see when new books get added. Since this happens quite infrequently, having an alert system would be great.

bpo2rss shows how to quickly accomplish this task. The *make* function of the *RssMaker* module does the heavy-lifting. The *url* parameter specifies the URL for *books.perl.org*'s web page, containing links to recently discussed books. *output* specifies the name of the resulting RSS file. *title* is the title of the feed shown later in the news aggregator.

RssMaker calls the anonymous *filter* subroutine once per link. Each time it does so, *RssMaker* passes two parameters: the URL for the link and the matching text. The subroutine uses this information to check if the link is a headline that it should add to the feed. If the filter returns a 1, the link is added to the feed; if the filter returns a 0, the link is not added. In the case of *books.perl.org*'s site, *bpo2rss* simply checks if the URL matches the pattern */books/n* where *n* is a numeric value. That seems to be the *books.perl.org*'s convention on linking to book reviews. That's all it takes to complete the RSS feed.

It's even possible to modify the extracted link or its textual description before it is added to the RSS feed file, by using a simple trick: If you pass a parameter to a subroutine in Perl, this gives you both read and write access. Setting *$_[0]* in the function to a different value will modify the parameter passed in by the main program. When *make* calls *filter($url, $text)* and *filter* modifies *$_[0]* or *$_[1]*, then *$url* or *$text* will have changed in the *bpo2rss*, resulting in modified entries in the outgoing RSS feed file.

Aggregators

Services such as Blogline (*www.blogline.com*) run Web applications that allow registered users to subscribe to feeds and actively monitor these feeds for updates. My tip for a local tool is *Amphetadesk* ([2]), a Perl script that runs as an HTTP server on the local machine and displays an overview of headlines in your browser (Figure 1). ▶

Listing 1: RssMaker.pm

```
001 ################################
002 # RssMaker -- Generate a RSS
003 #         feed of a web page
004 # Mike Schilli, 2004
005 # (m@perlmeister.com)
006 ################################
007 package RssMaker;
008
009 use warnings;
010 use strict;
011
012 use LWP::UserAgent;
013 use HTTP::Request::Common;
014 use XML::RSS;
015 use HTML::Entities
016           qw(decode_entities);
017 use URI::URL;
018 use HTTP::Date;
019 use DateTime;
020 use HTML::TreeBuilder;
021 use Log::Log4perl qw(:easy);
022
023 ################################
024 sub make {
025 ################################
026   my (%o) = @_;
027
028   $o{url}
029     || LOGDIE "url missing";
030   $o{title}
031     || LOGDIE
032     "title missing";
033   $o{output}   ||= "out.rdf";
034   $o{filter}   ||= sub { 1 };
035   $o{encoding} ||= 'utf-8';
036
037   my $ua =
038     LWP::UserAgent->new();
039
040   INFO "Fetching $o{url}";
041   my $resp =
042     $ua->request(
043     GET $o{url} );
044
045   LOGDIE "Error fetching ",
046          "$o{url}"
047       if $resp->is_error();
048
049   my $http_time =
050     $resp->header(
051     'last-modified');
052
053   $http_time ||=
054     time2str( time() );
055
056   INFO "Last modified: ",
057        $http_time;
058
059   my $mtime =
060     str2time($http_time);
061
062   my $isotime =
063     DateTime->from_epoch(
064     epoch => $mtime);
065
066   DEBUG "Last modified:",
067         $isotime;
068
069   my $rss =
070     XML::RSS->new(
071     encoding =>
072       $o{encoding} );
073
074   $rss->channel(
075     title => $o{title},
076     link  => $o{url},
077     dc    => {
078       date => $isotime . "Z"
079     },
080   );
081
082   foreach(exlinks(
083       $resp->content(),
084       $o{url})) {
085
086     my ($lurl, $text) = @$_;
087
088     $text =
089       decode_entities($text);
090
091     if ($o{filter}->(
092             $lurl, $text)) {
093
094       INFO "Adding rss ",
095       "entry: $text $lurl";
096
097       $rss->add_item(
098         title => $text,
099         link  => $lurl);
100     }
101   }
102
103   INFO "Saving output in ",
104        "$o{output}";
105   $rss->save( $o{output} )
106     or LOGDIE "Cannot write",
107        " to", " $o{output}";
108 }
109
110 ################################
111 sub exlinks {
112 ################################
113   my ($html, $base_url) = @_;
114
115   my @links = ();
116
117   my $tree =
118     HTML::TreeBuilder->new();
119
120   $tree->parse($html)
121     or return ();
122
123   for(@{$tree->extract_links(
124         'a')}) {
125     my ($link, $element,
126         $attr, $tag) = @$_;
127
128     next
129       unless $attr eq "href";
130
131     my $uri =
132       URI->new_abs( $link,
133       $base_url );
134
135     next
136       unless length $element
137       ->as_trimmed_text();
138
139     push @links,
140       [
141       URI->new_abs(
142         $link, $base_url
143       ),
144       $element
145         ->as_trimmed_text()
146       ];
147   }
148
149   return @links;
150 }
151
152 1;
```

If you want to check if the RSS file fulfills the strict rules of the standard, you can validate the file online:

```
http://feeds.archive.org/validator/
```

has a free realtime service and gives you a really neat looking seal of approval if your feed checks out okay (Figure 2).

RssMaker.pm uses Log4perl in *easy* mode for debugging, *LWP::UserAgent* to fetch the URLs and *XML:RSS* to create the RSS file. *decode_entities* from *HTML::Entities* decodes HTML escape sequences such as *ü*. The *exlinks* function in *RssMaker.pm* provides link extraction using *HTML::TreeBuilder*. *as_trimmed_text()* digs the text out of HTML's <A> link tags.

Atomic Time

The *RSS* standard looks set to be replaced by a new standard called *Atom* sometime in the near future. The usual committees are already working on this problem. If the *Atom* clients listed at [6] reach a critical mass, CPAN will probably have an *AtomMaker* module with similar functionality to *RssMaker* to match. It will then use the *XML::Atom* module, which today is already available on CPAN. At present, many popular clients do not support the *Atom* format, and some of the listed clients are extremely buggy. [3] gives you an introduction to *Atom*, and there is a simple tutorial at [4].

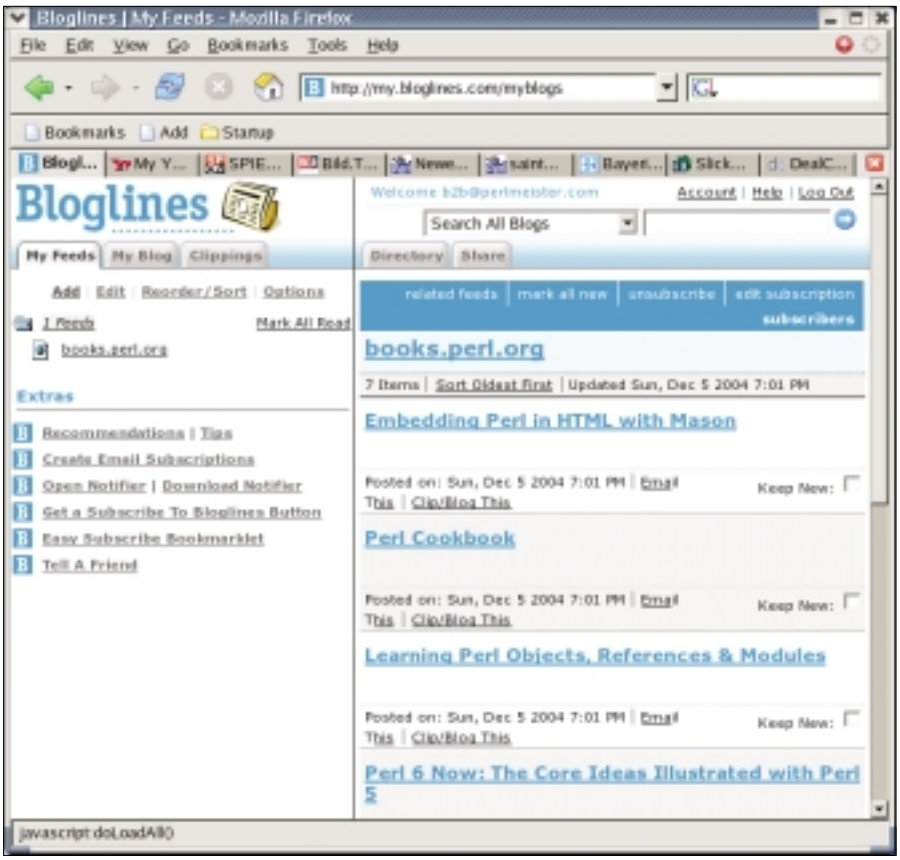

Figure 3: A newsfeed in Bloglines.

Installation

All of the modules required by *RssMaker.pm* are available from CPAN. You should set up any scraper scripts such as *bpo2rss* to run on your sysem as cronjobs, typically once a day. The resulting RSS files should *only* be published on the local Intranet, since publishing RSS files on the Internet could be interpreted as deep linking and might lead to legal problems.

During the debugging phase, it makes sense to set the *Log4perl* setting for the script to *$DEBUG*. The benefit of setting *Log4perl* to *$DEBUG* is that the *$DEBUG* value will allow you to monitor activities such as fetching, link extraction, and RSS feed generating on screen. In a production environment, you can use the *$ERROR* setting instead to remove any unwanted output and stop the cronjob from bombarding you with email messages. ■

Listing 2: bpo2rss

```
01 #!/usr/bin/perl
02 ##################################
03 # bpo2rss -- books.perl.org
04 #          RSS feed generator
05 # Mike Schilli, 2004
06 # (m@perlmeister.com)
07 ##################################
08 use warnings;
09 use strict;
10
11 use RssMaker;
12 use Log::Log4perl qw(:easy);
13
14 Log::Log4perl->easy_init(
15   $INFO);
16
17 my $url =
18   'http://books.perl.org/';
19
20 RssMaker::make(
21   url    => $url,
22   title  => "books.perl.org",
23   filter => sub {
24     my ( $link, $text ) = @_;
25
26     return 1
27       if $link =~
28         m#/book/\d+#;
29     return 0;
30   },
31   output => "bpo.rss",
32 );
```

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/51/Perl*

[2] Amphetadesk, "Syndicated Aggregator" *http://www.disobey.com/amphetadesk*

[3] Michael Fitzgerald, "XML Hacks", O'Reilly

[4] Reuven Lerner, "Aggregating with Atom", Linux-Journal 11/2004, p. 18ff.

[5] Ben Hammersley, "Content Syndication with RSS", O'Reilly 2003

[6] List of applications that support *Atom*: *http://atomenabled.org*