NAME

perlfaq9 - Web, Email and Networking

DESCRIPTION

This section deals with questions related to running web sites, sending and receiving email as well as general networking.

Should I use a web framework?

Yes. If you are building a web site with any level of interactivity (forms / users / databases), you will want to use a framework to make handling requests and responses easier.

If there is no interactivity then you may still want to look at using something like *Template Toolkit* or *Plack::Middleware::TemplateToolkit* so maintenance of your HTML files (and other assets) is easier.

Which web framework should I use?

There is no simple answer to this question. Perl frameworks can run everything from basic file servers and small scale intranets to massive multinational multilingual websites that are the core to international businesses.

Below is a list of a few frameworks with comments which might help you in making a decision, depending on your specific requirements. Start by reading the docs, then ask questions on the relevant mailing list or IRC channel.

Catalyst

Strongly object-oriented and fully-featured with a long development history and a large community and addon ecosystem. It is excellent for large and complex applications, where you have full control over the server.

Dancer

Young and free of legacy weight, providing a lightweight and easy to learn API. Has a growing addon ecosystem. It is best used for smaller projects and very easy to learn for beginners.

Mojolicious

Fairly young with a focus on HTML5 and real-time web technologies such as WebSockets.

Web::Simple

Currently experimental, strongly object-oriented, built for speed and intended as a toolkit for building micro web apps, custom frameworks or for tieing together existing Plack-compatible web applications with one central dispatcher.

All of these interact with or use *Plack* which is worth understanding the basics of when building a website in Perl (there is a lot of useful *Plack::Middleware*).

What is Plack and PSGI?

PSGI is the Perl Web Server Gateway Interface Specification, it is a standard that many Perl web frameworks use, you should not need to understand it to build a web site, the part you might want to use is *Plack*.

Plack is a set of tools for using the PSGI stack. It contains *middleware* components, a reference server and utilities for Web application frameworks. Plack is like Ruby's Rack or Python's Paste for WSGI.

You could build a web site using *Plack* and your own code, but for anything other than a very basic web site, using a web framework (that uses *Plack*) is a better option.

How do I remove HTML from a string?

Use *HTML::Strip*, or *HTML::FormatText* which not only removes HTML but also attempts to do a little simple formatting of the resulting plain text.

How do I extract URLs?

HTML::SimpleLinkExtor will extract URLs from HTML, it handles anchors, images, objects, frames, and many other tags that can contain a URL. If you need anything more complex, you can create your own subclass of *HTML::LinkExtor* or *HTML::Parser*. You might even use *HTML::SimpleLinkExtor* as an example for something specifically suited to your needs.

You can use *URI::Find* to extract URLs from an arbitrary text document.

How do I fetch an HTML file?

(contributed by brian d foy)

Use the libwww-perl distribution. The *LWP::Simple* module can fetch web resources and give their content back to you as a string:

```
use LWP::Simple qw(get);

my $html = get( "http://www.example.com/index.html" );
```

It can also store the resource directly in a file:

```
use LWP::Simple qw(getstore);

getstore( "http://www.example.com/index.html", "foo.html" );
```

If you need to do something more complicated, you can use *LWP::UserAgent* module to create your own user-agent (e.g. browser) to get the job done. If you want to simulate an interactive web browser, you can use the *WWW::Mechanize* module.

How do I automate an HTML form submission?

If you are doing something complex, such as moving through many pages and forms or a web site, you can use *WWW::Mechanize*. See its documentation for all the details.

If you're submitting values using the GET method, create a URL and encode the form using the `query_form` method:

```
use LWP::Simple;
use URI::URL;

my $url = url('L<http://www.perl.com/cgi-bin/cpan_mod')>;
$url->query_form(module => 'DB_File', readme => 1);
$content = get($url);
```

If you're using the POST method, create your own user agent and encode the content appropriately.

```
use HTTP::Request::Common qw(POST);
use LWP::UserAgent;

my $ua = LWP::UserAgent->new();
my $req = POST 'L<http://www.perl.com/cgi-bin/cpan_mod'>,
               [ module => 'DB_File', readme => 1 ];
my $content = $ua->request($req)->as_string;
```

How do I decode or create those %-encodings on the web?

Most of the time you should not need to do this as your web framework, or if you are making a request, the *LWP* or other module would handle it for you.

To encode a string yourself, use the *URI::Escape* module. The `uri_escape` function returns the escaped string:

```
my $original = "Colon : Hash # Percent %";

my $escaped = uri_escape( $original );

print "$escaped\n"; # 'Colon%20%3A%20Hash%20%23%20Percent%20%25'
```

To decode the string, use the `uri_unescape` function:

```
my $unescaped = uri_unescape( $escaped );

print $unescaped; # back to original
```

Remember not to encode a full URI, you need to escape each component separately and then join them together.

How do I redirect to another page?

Most Perl Web Frameworks will have a mechanism for doing this, using the *Catalyst* framework it would be:

```
$c->res->redirect($url);
$c->detach();
```

If you are using Plack (which most frameworks do), then *Plack::Middleware::Rewrite* is worth looking at if you are migrating from Apache or have URL's you want to always redirect.

How do I put a password on my web pages?

See if the web framework you are using has an authentication system and if that fits your needs.

Alternativly look at *Plack::Middleware::Auth::Basic*, or one of the other *Plack authentication* options.

How do I make sure users can't enter values into a form that causes my CGI script to do bad things?

(contributed by brian d foy)

You can't prevent people from sending your script bad data. Even if you add some client-side checks, people may disable them or bypass them completely. For instance, someone might use a module such as *LWP* to submit to your web site. If you want to prevent data that try to use SQL injection or other sorts of attacks (and you should want to), you have to not trust any data that enter your program.

The *perlsec* documentation has general advice about data security. If you are using the *DBI* module, use placeholder to fill in data. If you are running external programs with `system` or `exec`, use the list forms. There are many other precautions that you should take, too many to list here, and most of them fall under the category of not using any data that you don't intend to use. Trust no one.

How do I parse a mail header?

Use the *Email::MIME* module. It's well-tested and supports all the craziness that you'll see in the real world (comment-folding whitespace, encodings, comments, etc.).

```
use Email::MIME;

my $message = Email::MIME->new($rfc2822);
my $subject = $message->header('Subject');
my $from    = $message->header('From');
```

If you've already got some other kind of email object, consider passing it to *Email::Abstract* and then using its cast method to get an *Email::MIME* object:

```
my $mail_message_object = read_message();
my $abstract = Email::Abstract->new($mail_message_object);
my $email_mime_object = $abstract->cast('Email::MIME');
```

How do I check a valid mail address?

(partly contributed by Aaron Sherman)

This isn't as simple a question as it sounds. There are two parts:

a) How do I verify that an email address is correctly formatted?

b) How do I verify that an email address targets a valid recipient?

Without sending mail to the address and seeing whether there's a human on the other end to answer you, you cannot fully answer part *b*, but the *Email::Valid* module will do both part *a* and part *b* as far as you can in real-time.

Our best advice for verifying a person's mail address is to have them enter their address twice, just as you normally do to change a password. This usually weeds out typos. If both versions match, send mail to that address with a personal message. If you get the message back and they've followed your directions, you can be reasonably assured that it's real.

A related strategy that's less open to forgery is to give them a PIN (personal ID number). Record the address and PIN (best that it be a random one) for later processing. In the mail you send, include a link to your site with the PIN included. If the mail bounces, you know it's not valid. If they don't click on the link, either they forged the address or (assuming they got the message) following through wasn't important so you don't need to worry about it.

How do I decode a MIME/BASE64 string?

The *MIME::Base64* package handles this as well as the MIME/QP encoding. Decoding base 64 becomes as simple as:

```
use MIME::Base64;
my $decoded = decode_base64($encoded);
```

The *Email::MIME* module can decode base 64-encoded email message parts transparently so the developer doesn't need to worry about it.

How do I find the user's mail address?

Ask them for it. There are so many email providers available that it's unlikely the local system has any idea how to determine a user's email address.

The exception is for organization-specific email (e.g. foo@yourcompany.com) where policy can be codified in your program. In that case, you could look at $ENV{USER}, $ENV{LOGNAME}, and getpwuid($<) in scalar context, like so:

```
my $user_name = getpwuid($<)
```

But you still cannot make assumptions about whether this is correct, unless your policy says it is. You really are best off asking the user.

How do I send email?

Use the *Email::MIME* and *Email::Sender::Simple* modules, like so:

```
# first, create your message
my $message = Email::MIME->create(
  header_str => [
    From    => 'you@example.com',
    To      => 'friend@example.com',
    Subject => 'Happy birthday!',
  ],
  attributes => {
    encoding => 'quoted-printable',
    charset  => 'utf-8',
  },
  body_str => "Happy birthday to you!\n",
);

use Email::Sender::Simple qw(sendmail);
sendmail($message);
```

By default, *Email::Sender::Simple* will try `sendmail` first, if it exists in your $PATH. This generally isn't the case. If there's a remote mail server you use to send mail, consider investigating one of the Transport classes. At time of writing, the available transports include:

Email::Sender::Transport::Sendmail

This is the default. If you can use the *mail(1)* or *mailx(1)* program to send mail from the machine where your code runs, you should be able to use this.

Email::Sender::Transport::SMTP

This transport contacts a remote SMTP server over TCP. It optionally uses SSL and can authenticate to the server via SASL.

Email::Sender::Transport::SMTP::TLS

This is like the SMTP transport, but uses TLS security. You can authenticate with this module as well, using any mechanisms your server supports after STARTTLS.

Telling *Email::Sender::Simple* to use your transport is straightforward.

```
sendmail(
  $message,
  {
    transport => $email_sender_transport_object,
  }
);
```

How do I use MIME to make an attachment to a mail message?

Email::MIME directly supports multipart messages. *Email::MIME* objects themselves are parts and can be attached to other *Email::MIME* objects. Consult the *Email::MIME* documentation for more information, including all of the supported methods and examples of their use.

How do I read email?

Use the *Email::Folder* module, like so:

```
use Email::Folder;

my $folder = Email::Folder->new('/path/to/email/folder');
while(my $message = $folder->next_message) {
  # next_message returns Email::Simple objects, but we want
  # Email::MIME objects as they're more robust
  my $mime = Email::MIME->new($message->as_string);
}
```

There are different classes in the *Email::Folder* namespace for supporting various mailbox types. Note that these modules are generally rather limited and only support **reading** rather than writing.

How do I find out my hostname, domainname, or IP address?

(contributed by brian d foy)

The *Net::Domain* module, which is part of the Standard Library starting in Perl 5.7.3, can get you the fully qualified domain name (FQDN), the host name, or the domain name.

```
use Net::Domain qw(hostname hostfqdn hostdomain);

my $host = hostfqdn();
```

The *Sys::Hostname* module, part of the Standard Library, can also get the hostname:

```
use Sys::Hostname;

$host = hostname();
```

The *Sys::Hostname::Long* module takes a different approach and tries harder to return the fully qualified hostname:

```
use Sys::Hostname::Long 'hostname_long';

my $hostname = hostname_long();
```

To get the IP address, you can use the `gethostbyname` built-in function to turn the name into a number. To turn that number into the dotted octet form (a.b.c.d) that most people expect, use the `inet_ntoa` function from the *Socket* module, which also comes with perl.

```
use Socket;

my $address = inet_ntoa(
    scalar gethostbyname( $host || 'localhost' )
);
```

How do I fetch/put an (S)FTP file?

Net::FTP, and *Net::SFTP* allow you to interact with FTP and SFTP (Secure FTP) servers.

How can I do RPC in Perl?

Use one of the RPC modules(*https://metacpan.org/search?q=RPC*).

AUTHOR AND COPYRIGHT