NAME

perlfaq9 - Networking ($Revision: 1.16 $, $Date: 2004/10/30 12:20:59 $)

DESCRIPTION

This section deals with questions related to networking, the internet, and a few on the web.

What is the correct form of response from a CGI script?

(Alan Flavell <flavell+www@a5.ph.gla.ac.uk> answers...)

The Common Gateway Interface (CGI) specifies a software interface between a program ("CGI script") and a web server (HTTPD). It is not specific to Perl, and has its own FAQs and tutorials, and usenet group, comp.infosystems.www.authoring.cgi

The original CGI specification is at: http://hoohoo.ncsa.uiuc.edu/cgi/

Current best-practice RFC draft at: http://CGI-Spec.Golux.Com/

Other relevant documentation listed in: http://www.perl.org/CGI_MetaFAQ.html

These Perl FAQs very selectively cover some CGI issues. However, Perl programmers are strongly advised to use the CGI.pm module, to take care of the details for them.

The similarity between CGI response headers (defined in the CGI specification) and HTTP response headers (defined in the HTTP specification, RFC2616) is intentional, but can sometimes be confusing.

The CGI specification defines two kinds of script: the "Parsed Header" script, and the "Non Parsed Header" (NPH) script. Check your server documentation to see what it supports. "Parsed Header" scripts are simpler in various respects. The CGI specification allows any of the usual newline representations in the CGI response (it's the server's job to create an accurate HTTP response based on it). So "\n" written in text mode is technically correct, and recommended. NPH scripts are more tricky: they must put out a complete and accurate set of HTTP transaction response headers; the HTTP specification calls for records to be terminated with carriage-return and line-feed, i.e ASCII \015\012 written in binary mode.

Using CGI.pm gives excellent platform independence, including EBCDIC systems. CGI.pm selects an appropriate newline representation ($CGI::CRLF) and sets binmode as appropriate.

My CGI script runs from the command line but not the browser. (500 Server Error)

Several things could be wrong. You can go through the "Troubleshooting Perl CGI scripts" guide at

```
http://www.perl.org/troubleshooting_CGI.html
```

If, after that, you can demonstrate that you've read the FAQs and that your problem isn't something simple that can be easily answered, you'll probably receive a courteous and useful reply to your question if you post it on comp.infosystems.www.authoring.cgi (if it's something to do with HTTP or the CGI protocols). Questions that appear to be Perl questions but are really CGI ones that are posted to comp.lang.perl.misc are not so well received.

The useful FAQs, related documents, and troubleshooting guides are listed in the CGI Meta FAQ:

```
http://www.perl.org/CGI_MetaFAQ.html
```

How can I get better error messages from a CGI program?

Use the CGI::Carp module. It replaces `warn` and `die`, plus the normal Carp modules `carp`, `croak`, and `confess` functions with more verbose and safer versions. It still sends them to the normal server error log.

```
    use CGI::Carp;
```

```
    warn "This is a complaint";
    die "But this one is serious";
```

The following use of CGI::Carp also redirects errors to a file of your choice, placed in a BEGIN block to catch compile-time warnings as well:

```
    BEGIN {
        use CGI::Carp qw(carpout);
        open(LOG, ">>/var/local/cgi-logs/mycgi-log")
            or die "Unable to append to mycgi-log: $!\n";
        carpout(*LOG);
    }
```

You can even arrange for fatal errors to go back to the client browser, which is nice for your own debugging, but might confuse the end user.

```
    use CGI::Carp qw(fatalsToBrowser);
    die "Bad error here";
```

Even if the error happens before you get the HTTP header out, the module will try to take care of this to avoid the dreaded server 500 errors. Normal warnings still go out to the server error log (or wherever you've sent them with `carpout`) with the application name and date stamp prepended.

How do I remove HTML from a string?

The most correct way (albeit not the fastest) is to use HTML::Parser from CPAN. Another mostly correct way is to use HTML::FormatText which not only removes HTML but also attempts to do a little simple formatting of the resulting plain text.

Many folks attempt a simple-minded regular expression approach, like `s/<.*?>//g`, but that fails in many cases because the tags may continue over line breaks, they may contain quoted angle-brackets, or HTML comment may be present. Plus, folks forget to convert entities--like `<` for example.

Here's one "simple-minded" approach, that works for most files:

```
    #!/usr/bin/perl -p0777
    s/<(?:[^>'"]*|(['"]).*?\1)*>//gs
```

If you want a more complete solution, see the 3-stage striphtml program in http://www.cpan.org/authors/Tom_Christiansen/scripts/striphtml.gz .

Here are some tricky cases that you should think about when picking a solution:

```
    <IMG SRC = "foo.gif" ALT = "A > B">

    <IMG SRC = "foo.gif"
  ALT = "A > B">

    <!-- <A comment> -->

    <script>if (a<b && a>c)</script>

    <# Just data #>

    <![INCLUDE CDATA [ >>>>>>>>>>>> ]]>
```

If HTML comments include other tags, those solutions would also break on text like this:

```
<!-- This section commented out.
    <B>You can't see me!</B>
-->
```

How do I extract URLs?

You can easily extract all sorts of URLs from HTML with `HTML::SimpleLinkExtor` which 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.

Less complete solutions involving regular expressions can save you a lot of processing time if you know that the input is simple. One solution from Tom Christiansen runs 100 times faster than most module based approaches but only extracts URLs from anchors where the first attribute is HREF and there are no other attributes.

```
#!/usr/bin/perl -n00
# qxurl - tchrist@perl.com
print "$2\n" while m{
    < \s*
      A \s+ HREF \s* = \s* (["']) (.*?) \1
    \s* >
}gsix;
```

How do I download a file from the user's machine? How do I open a file on another machine?

In this case, download means to use the file upload feature of HTML forms. You allow the web surfer to specify a file to send to your web server. To you it looks like a download, and to the user it looks like an upload. No matter what you call it, you do it with what's known as **multipart/form-data** encoding. The CGI.pm module (which comes with Perl as part of the Standard Library) supports this in the start_multipart_form() method, which isn't the same as the startform() method.

See the section in the CGI.pm documentation on file uploads for code examples and details.

How do I make a pop-up menu in HTML?

Use the **<SELECT>** and **<OPTION>** tags. The CGI.pm module (available from CPAN) supports this widget, as well as many others, including some that it cleverly synthesizes on its own.

How do I fetch an HTML file?

One approach, if you have the lynx text-based HTML browser installed on your system, is this:

```
$html_code = `lynx -source $url`;
$text_data = `lynx -dump $url`;
```

The libwww-perl (LWP) modules from CPAN provide a more powerful way to do this. They don't require lynx, but like lynx, can still work through proxies:

```
# simplest version
use LWP::Simple;
$content = get($URL);

# or print HTML from a URL
use LWP::Simple;
```

```
getprint "http://www.linpro.no/lwp/";

# or print ASCII from HTML from a URL
# also need HTML-Tree package from CPAN
use LWP::Simple;
use HTML::Parser;
use HTML::FormatText;
my ($html, $ascii);
$html = get("http://www.perl.com/");
defined $html
    or die "Can't fetch HTML from http://www.perl.com/";
$ascii = HTML::FormatText->new->format(parse_html($html));
print $ascii;
```

How do I automate an HTML form submission?

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('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;

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

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

If you are writing a CGI script, you should be using the CGI.pm module that comes with perl, or some other equivalent module. The CGI module automatically decodes queries for you, and provides an escape() function to handle encoding.

The best source of detailed information on URI encoding is RFC 2396. Basically, the following substitutions do it:

```
s/([^\w()'*~!.-])/sprintf '%%%02x', ord $1/eg;   # encode

s/%([A-Fa-f\d]{2})/chr hex $1/eg;            # decode
```

However, you should only apply them to individual URI components, not the entire URI, otherwise you'll lose information and generally mess things up. If that didn't explain it, don't worry. Just go read section 2 of the RFC, it's probably the best explanation there is.

RFC 2396 also contains a lot of other useful information, including a regexp for breaking any arbitrary URI into components (Appendix B).

How do I redirect to another page?

Specify the complete URL of the destination (even if it is on the same server). This is one of the two different kinds of CGI "Location:" responses which are defined in the CGI specification for a Parsed Headers script. The other kind (an absolute URLpath) is resolved internally to the server without any HTTP redirection. The CGI specifications do not allow relative URLs in either case.

Use of CGI.pm is strongly recommended. This example shows redirection with a complete URL. This redirection is handled by the web browser.

```
use CGI qw/:standard/;

my $url = 'http://www.cpan.org/';
print redirect($url);
```

This example shows a redirection with an absolute URLpath. This redirection is handled by the local web server.

```
my $url = '/CPAN/index.html';
print redirect($url);
```

But if coded directly, it could be as follows (the final "\n" is shown separately, for clarity), using either a complete URL or an absolute URLpath.

```
print "Location: $url\n";   # CGI response header
print "\n";                 # end of headers
```

How do I put a password on my web pages?

To enable authentication for your web server, you need to configure your web server. The configuration is different for different sorts of web servers---apache does it differently from iPlanet which does it differently from IIS. Check your web server documentation for the details for your particular server.

How do I edit my .htpasswd and .htgroup files with Perl?

The HTTPD::UserAdmin and HTTPD::GroupAdmin modules provide a consistent OO interface to these files, regardless of how they're stored. Databases may be text, dbm, Berkeley DB or any database with a DBI compatible driver. HTTPD::UserAdmin supports files used by the `Basic' and `Digest' authentication schemes. Here's an example:

```
 use HTTPD::UserAdmin ();
 HTTPD::UserAdmin
->new(DB => "/foo/.htpasswd")
->add($username => $password);
```

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

See the security references listed in the CGI Meta FAQ

```
http://www.perl.org/CGI_MetaFAQ.html
```

How do I parse a mail header?

For a quick-and-dirty solution, try this solution derived from *"split" in perlfunc*:

```
$/ = '';
$header = <MSG>;
$header =~ s/\n\s+/ /g;  # merge continuation lines
```

```
%head = ( UNIX_FROM_LINE, split /^([-\w]+):\s*/m, $header );
```

That solution doesn't do well if, for example, you're trying to maintain all the Received lines. A more complete approach is to use the Mail::Header module from CPAN (part of the MailTools package).

How do I decode a CGI form?

You use a standard module, probably CGI.pm. Under no circumstances should you attempt to do so by hand!

You'll see a lot of CGI programs that blindly read from STDIN the number of bytes equal to CONTENT_LENGTH for POSTs, or grab QUERY_STRING for decoding GETs. These programs are very poorly written. They only work sometimes. They typically forget to check the return value of the read() system call, which is a cardinal sin. They don't handle HEAD requests. They don't handle multipart forms used for file uploads. They don't deal with GET/POST combinations where query fields are in more than one place. They don't deal with keywords in the query string.

In short, they're bad hacks. Resist them at all costs. Please do not be tempted to reinvent the wheel. Instead, use the CGI.pm or CGI_Lite.pm (available from CPAN), or if you're trapped in the module-free land of perl1 .. perl4, you might look into cgi-lib.pl (available from http://cgi-lib.stanford.edu/cgi-lib/).

Make sure you know whether to use a GET or a POST in your form. GETs should only be used for something that doesn't update the server. Otherwise you can get mangled databases and repeated feedback mail messages. The fancy word for this is ``idempotency''. This simply means that there should be no difference between making a GET request for a particular URL once or multiple times. This is because the HTTP protocol definition says that a GET request may be cached by the browser, or server, or an intervening proxy. POST requests cannot be cached, because each request is independent and matters. Typically, POST requests change or depend on state on the server (query or update a database, send mail, or purchase a computer).

How do I check a valid mail address?

You can't, at least, not in real time. Bummer, eh?

Without sending mail to the address and seeing whether there's a human on the other end to answer you, you cannot determine whether a mail address is valid. Even if you apply the mail header standard, you can have problems, because there are deliverable addresses that aren't RFC-822 (the mail header standard) compliant, and addresses that aren't deliverable which are compliant.

You can use the Email::Valid or RFC::RFC822::Address which check the format of the address, although they cannot actually tell you if it is a deliverable address (i.e. that mail to the address will not bounce). Modules like Mail::CheckUser and Mail::EXPN try to interact with the domain name system or particular mail servers to learn even more, but their methods do not work everywhere---especially for security conscious administrators.

Many are tempted to try to eliminate many frequently-invalid mail addresses with a simple regex, such as `/^[\w.-]+\@(?:[\w-]+\.)+\w+$/`. It's a very bad idea. However, this also throws out many valid ones, and says nothing about potential deliverability, so it is not suggested. Instead, see http://www.cpan.org/authors/Tom_Christiansen/scripts/ckaddr.gz , which actually checks against the full RFC spec (except for nested comments), looks for addresses you may not wish to accept mail to (say, Bill Clinton or your postmaster), and then makes sure that the hostname given can be looked up in the DNS MX records. It's not fast, but it works for what it tries to do.

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 that looks somewhat like:

```
Dear someuser@host.com,
```

```
Please confirm the mail address you gave us Wed May  6 09:38:41
MDT 1998 by replying to this message.  Include the string
"Rumpelstiltskin" in that reply, but spelled in reverse; that is,
start with "Nik...".  Once this is done, your confirmed address will
be entered into our records.
```

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, ask them to include the PIN in their reply. But if it bounces, or the message is included via a ``vacation'' script, it'll be there anyway. So it's best to ask them to mail back a slight alteration of the PIN, such as with the characters reversed, one added or subtracted to each digit, etc.

How do I decode a MIME/BASE64 string?

The MIME-Base64 package (available from CPAN) handles this as well as the MIME/QP encoding. Decoding BASE64 becomes as simple as:

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

The MIME-Tools package (available from CPAN) supports extraction with decoding of BASE64 encoded attachments and content directly from email messages.

If the string to decode is short (less than 84 bytes long) a more direct approach is to use the unpack() function's "u" format after minor transliterations:

```
tr#A-Za-z0-9+/##cd;                  # remove non-base64 chars
tr#A-Za-z0-9+/# -_#;                 # convert to uuencoded format
$len = pack("c", 32 + 0.75*length);  # compute length byte
print unpack("u", $len . $_);        # uudecode and print
```

How do I return the user's mail address?

On systems that support getpwuid, the $< variable, and the Sys::Hostname module (which is part of the standard perl distribution), you can probably try using something like this:

```
use Sys::Hostname;
$address = sprintf('%s@%s', scalar getpwuid($<), hostname);
```

Company policies on mail address can mean that this generates addresses that the company's mail system will not accept, so you should ask for users' mail addresses when this matters. Furthermore, not all systems on which Perl runs are so forthcoming with this information as is Unix.

The Mail::Util module from CPAN (part of the MailTools package) provides a mailaddress() function that tries to guess the mail address of the user. It makes a more intelligent guess than the code above, using information given when the module was installed, but it could still be incorrect. Again, the best way is often just to ask the user.

How do I send mail?

Use the `sendmail` program directly:

```
open(SENDMAIL, "|/usr/lib/sendmail -oi -t -odq")
                    or die "Can't fork for sendmail: $!\n";
print SENDMAIL <<"EOF";
From: User Originating Mail <me\@host>
```

```
To: Final Destination <you\@otherhost>
Subject: A relevant subject line

Body of the message goes here after the blank line
in as many lines as you like.
EOF
close(SENDMAIL)     or warn "sendmail didn't close nicely";
```

The **-oi** option prevents sendmail from interpreting a line consisting of a single dot as "end of message". The **-t** option says to use the headers to decide who to send the message to, and **-odq** says to put the message into the queue. This last option means your message won't be immediately delivered, so leave it out if you want immediate delivery.

Alternate, less convenient approaches include calling mail (sometimes called mailx) directly or simply opening up port 25 have having an intimate conversation between just you and the remote SMTP daemon, probably sendmail.

Or you might be able use the CPAN module Mail::Mailer:

```
use Mail::Mailer;

$mailer = Mail::Mailer->new();
$mailer->open({ From    => $from_address,
                To      => $to_address,
                Subject => $subject,
              })
    or die "Can't open: $!\n";
print $mailer $body;
$mailer->close();
```

The Mail::Internet module uses Net::SMTP which is less Unix-centric than Mail::Mailer, but less reliable. Avoid raw SMTP commands. There are many reasons to use a mail transport agent like sendmail. These include queuing, MX records, and security.

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

This answer is extracted directly from the MIME::Lite documentation. Create a multipart message (i.e., one with attachments).

```
use MIME::Lite;

### Create a new multipart message:
$msg = MIME::Lite->new(
             From    =>'me@myhost.com',
             To      =>'you@yourhost.com',
             Cc      =>'some@other.com, some@more.com',
             Subject =>'A message with 2 parts...',
             Type    =>'multipart/mixed'
             );

### Add parts (each "attach" has same arguments as "new"):
$msg->attach(Type     =>'TEXT',
             Data     =>"Here's the GIF file you wanted"
             );
$msg->attach(Type     =>'image/gif',
             Path     =>'aaa000123.gif',
```

```
                    Filename =>'logo.gif'
                    );

    $text = $msg->as_string;
```

MIME::Lite also includes a method for sending these things.

```
    $msg->send;
```

This defaults to using *sendmail* but can be customized to use SMTP via *Net::SMTP*.

How do I read mail?

While you could use the Mail::Folder module from CPAN (part of the MailFolder package) or the Mail::Internet module from CPAN (part of the MailTools package), often a module is overkill. Here's a mail sorter.

```
    #!/usr/bin/perl

    my(@msgs, @sub);
    my $msgno = -1;
    $/ = '';                    # paragraph reads
    while (<>) {
        if (/^From /m) {
            /^Subject:\s*(?:Re:\s*)*(.*)/mi;
            $sub[++$msgno] = lc($1) || '';
        }
        $msgs[$msgno] .= $_;
    }
    for my $i (sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 .. $#msgs)) {
        print $msgs[$i];
    }
```

Or more succinctly,

```
    #!/usr/bin/perl -n00
    # bysub2 - awkish sort-by-subject
    BEGIN { $msgno = -1 }
    $sub[++$msgno] = (/^Subject:\s*(?:Re:\s*)*(.*)/mi)[0] if /^From/m;
    $msg[$msgno] .= $_;
    END { print @msg[ sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 ..
$#msg) ] }
```

How do I find out my hostname/domainname/IP address?

The normal way to find your own hostname is to call the `'hostname'` program. While sometimes expedient, this has some problems, such as not knowing whether you've got the canonical name or not. It's one of those tradeoffs of convenience versus portability.

The Sys::Hostname module (part of the standard perl distribution) will give you the hostname after which you can find out the IP address (assuming you have working DNS) with a gethostbyname() call.

```
    use Socket;
    use Sys::Hostname;
    my $host = hostname();
    my $addr = inet_ntoa(scalar gethostbyname($host || 'localhost'));
```

Probably the simplest way to learn your DNS domain name is to grok it out of /etc/resolv.conf, at least under Unix. Of course, this assumes several things about your resolv.conf configuration, including that it exists.

(We still need a good DNS domain name-learning method for non-Unix systems.)

How do I fetch a news article or the active newsgroups?

Use the Net::NNTP or News::NNTPClient modules, both available from CPAN. This can make tasks like fetching the newsgroup list as simple as

```
perl -MNews::NNTPClient
  -e 'print News::NNTPClient->new->list("newsgroups")'
```

How do I fetch/put an FTP file?

LWP::Simple (available from CPAN) can fetch but not put. Net::FTP (also available from CPAN) is more complex but can put as well as fetch.

How can I do RPC in Perl?

A DCE::RPC module is being developed (but is not yet available) and will be released as part of the DCE-Perl package (available from CPAN). The rpcgen suite, available from CPAN/authors/id/JAKE/, is an RPC stub generator and includes an RPC::ONC module.

AUTHOR AND COPYRIGHT