Email with Perl

Stefan Hornburg (Racke)

Nordic Perl Workshop 2018, Oslo, 6th September

Contents

1 Introduction

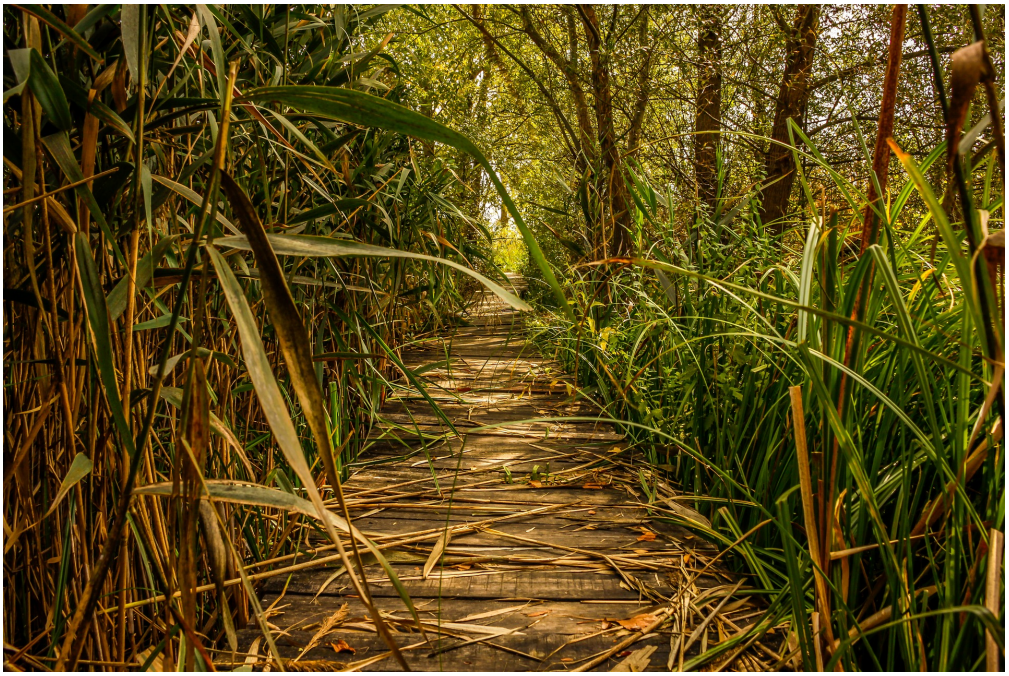
Text versus:

- HTML
- Attachments
- Embedded Images

Challenges:
Secluded island.

- Security
- Encryption
- SPAM
- Virus
- Counter Measures

So it is a jungle and we literally need to hack our way through it.



2 Overview

- Email structure
- Sending emails
- IMAP
- Parsing emails
- Use Cases

3 Email structure

Gone are the days of plain text emails, the majority is written in HTML with a plain text equivalent if you are lucky.

- Header
- Body
- Attachments
- HTML / Plaintext

A very nice feature of the Email::MIME module (more about that later) is the `debug_structure` method which gives you an overview about the different parts forming the email body.

```
Email::MIME->new( $message )->debug_structure;
```

```
+ multipart/mixed;
    + application/octet-stream; name=83AKE9.pdf
    + multipart/alternative;
        + text/plain; charset=UTF-8
        + text/html; charset=UTF-8
```

4 Sending emails

Too many ways to do it ...

- Net::SMTP
- MIME::Lite

Send:

- Email::Send (deprecated)

- Email::Sender
- Email::Sender::Simple

Misc:

- Email::MIME
- MIME::Entity
- Mail::Box

```
use Email::Sender::Simple qw(sendmail);
use Email::Simple;
use Email::Simple::Creator;
```

```
my $email = Email::Simple->create(
    header => [
        From    => '"Module Updates" <info@cpan.pm>',
        To      => '"Racke" <racke@racke.pm>',
        Subject => 'Changes in Module Foo::Bar',
    ],
    body => "Baz.\n",
);

sendmail($email);
```

Email::Simple unfortunately lacks a method for attachments. So we are simply using MIME::Entity instead.

```
my $email = MIME::Entity->build(
    From    => '"Module Updates" <info@cpan.pm>',
    To      => '"Racke" <racke@racke.pm>',
    Subject => 'Changes in Module Foo::Bar',
    Data => ["Baz.\n"],
);

$email->attach(
    Path => '../perlmail-de-handout.pdf',
    Type => 'application/pdf',
);

sendmail($email);
```

```
my $email = Email::Simple->create(
    header => [
        From    => '"Module Updates" <info@cpan.pm>',
        To      => '"Racke" <racke@racke.pm>',
        Subject =>  'Änderungen im Modul Foo::Bar',
    ],
    body => "Baz.\n",
);

sendmail($email);
```

☺ Thunderbird

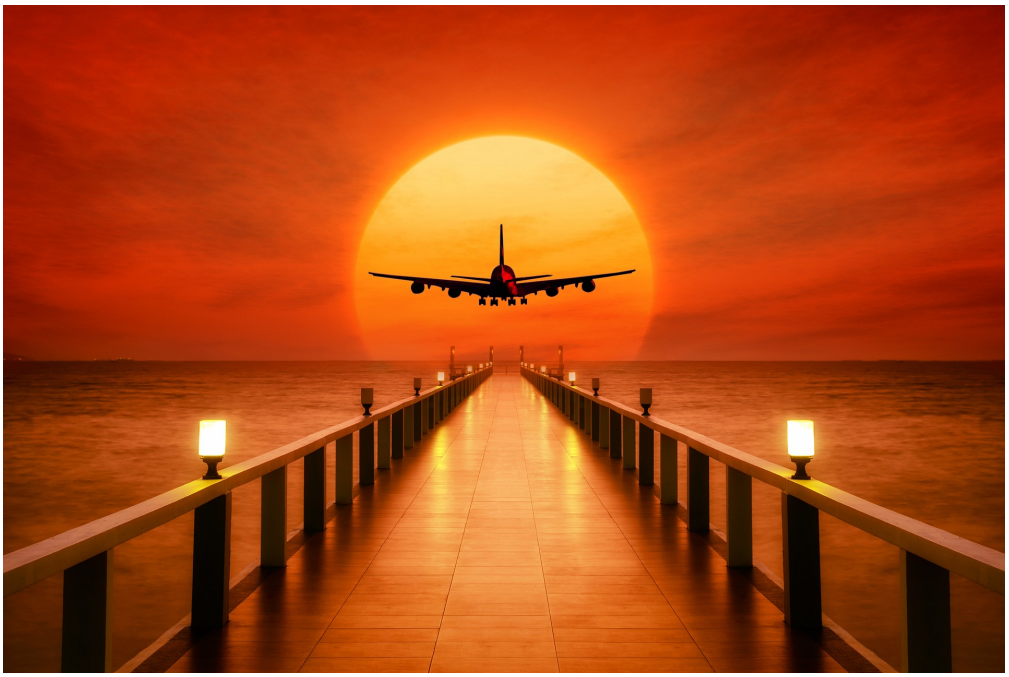
☹ K9 (Android)

```
my $email = Email::Simple->create(
    header => [
        From    => '"Module Updates" <info@cpan.pm>',
        To      => '"Racke" <racke@racke.pm>',
        Subject => encode('MIME-Header',
                          'Änderungen im Modul Foo',
                         ),
    ],
    body => "Bar.\n",
);

sendmail($email);
```

```
To: "Racke" <racke@racke.pm>
From: "Module Updates" <info@cpan.pm>
Subject: =?UTF-8?B?w4RuZGVydW5nZW4gaW4gTW9kdWxlIEZvbw==?=
```

```
my $email = Email::Simple->create(
    header => [
        From    => '"Module Updates" <info@cpan.pm>',
        To      => '"Racke" <racke@racke.pm>',
        Subject => encode('MIME-Header',
                          'Änderungen im Modul Foo::Bar',
                         ),
    ],
    body => encode('UTF-8', "Überflüssig"),
);

sendmail($email);
```

4.1 Transports



- Sendmail
- SMTP

- Maildir
- Redirect

4.1.1 Sendmail-Transport

Wenn nicht anders angegeben, wird der Sendmail-Transport verwendet, d.h. der auf dem Rechner installierte Mailserver.

Das ist nicht immer erwünscht, z.B. wegen dynamischer IP oder vorgebener Mailserver.

```
sendmail( $email );
```

4.1.2 Maildir-Transport

Beim Maildir-Transport wird das Verzeichnis Maildir im aktuellen Verzeichnis verwendet und angelegt soweit nicht vorhanden.

Deshalb verwenden wir hier das Verzeichnis Maildir im Homeverzeichnis.

```
use Email::Sender::Transport::Maildir qw();
use File::HomeDir;
use Path::Tiny;
```

```
my $maildir = path(File::HomeDir->my_home)
    ->child('Maildir');

my $transport = Email::Sender::Transport::Maildir->new(
   dir => $maildir
);

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

4.1.3 Test-Transport

4.1.4 Redirect-Transport

```
$transport_orig = Email::Sender::Transport::Sendmail->new;

$transport = Email::Sender::Transport::Redirect->new({
  transport => $transport_orig,
  redirect_address => 'racke@linuxia.de',
});

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

```
To: racke@linuxia.de
Cc: racke@linuxia.de
From: "Module Updates" <info@cpan.pm>
Subject: Hello world!
Date: Tue, 3 Apr 2018 08:45:59 +0200
X-Intercepted-To: "Racke" <racke@racke.pm>
X-Intercepted-Cc: "Info" <info@racke.pm>
X-Email-Sender-From: info@cpan.pm
X-Email-Sender-To: racke@linuxia.de
X-Email-Sender-To: racke@linuxia.de
Lines: 1

Here we go.
```

4.2 Embedding images

- Images as links
- Base64 encoded
- CID Inline

4.3 Create emails from templates

```
my $html = template $template, $tokens,
   { layout => 'email' };

my $f    = HTML::FormatText::WithLinks->new;
my $text = $f->parse($html);
```

Dancer::Plugin2::Email uses MIME::Entity and Email::Sender.

```
email {
    %args,
    body   => encode( 'UTF-8', $text ),
    type   => 'text',
    attach => {
       Charset  => 'utf-8',
       Data     => encode( 'UTF-8', $html ),
       Encoding => "quoted-printable",
       Type     => "text/html"
    },
    multipart => 'alternative',
};
```

5 IMAP

- Modules
- Login
- Select folder
- Search emails
- Download emails

5.1 IMAP Modules

- Net::IMAP::Client
- Net::IMAP::Simple

For example, Net::IMAP::Simple has several search helpers.

- Constructor
  - server
  - ssl / use_ssl
- Supported methods
- Return values
  - select
  - search

5.2 Login

```
my $imap = Net::IMAP::Client->new(
    server => 'mail.linuxia.pm',
    ssl => 1,
    port => 993,
    user => 'racke@racke.pm',
    pass => 'nevairbe',
);

$imap->login;
```

login rejected

5.3 Select folder

IMAP servers use different separators for the folder hierarchy.

Select folder (Dbmail):

```
$imap->select('INBOX/Consulting/Bahn');
```

Select folder (Courier):

```
$imap->select('INBOX.Consulting.Bahn');
```

```
$imap->separator;
```

5.4 Search emails

It is mandatory to select a folder first, otherwise the search fail with an error message like `Error in IMAP command received by server`.

All emails in selected folder:

```
$ids = $imap->search( 'ALL' , 'DATE' );
```

Search emails in selected folder:

```
$ids = $imap->search( { subject => 'Perl' }, 'DATE' );
```

5.5 Folders

List of folders:

```
$imap->folders
```

```
[
  'INBOX/Sympa',
  'INBOX',
  'Sent',
  'Drafts',
  'Trash'
]
```

5.6 Headers

```
$summary = $imap->get_summaries($ids);
```

5.7 Download emails

Download complete email:

```
$message = $imap->get_rfc822_body($id);
```

6 Processing emails

6.1 Modules for processing

- Email::MIME
- Email::Simple
- Read email from IMAP:

```
my $message = $imap->get_rfc822_body($id);
```

- Read email from file:

```
use Path::Tiny;
my $message = path('email.eml')->slurp;
```

- Parse email:

```
my $parser = Email::MIME->new($message);
```

Subject header:

```
print $parser->header_str('Subject');
```

```
Umlaute with Perl: ü ä ö
```

Subject header:

```
print $parser->header_raw('Subject');
```

```
=?UTF-8?B?VW1sYXV0ZSB3aXRoIFBlcmw6IMO8IMOkIMO2?=
```

The header will be already decoded.
Walk MIME parts:

```
my @out;

$parser->walk_parts(sub {
  my ($part) = @_;
  return if $part->subparts;

  if ($part->content_type =~ m{text/html}i) {
    push @out, $part->body_str;
  }
});
```

6.2 Extract train ticket

```
$ids = $imap->search('FROM buchungsbestaetigung@bahn.de');

for my $imap_id (@$ids) {
    my $message = $imap->get_rfc822_body($imap_id);
    my $parser = Email::MIME->new($message);
    my @out;

    ...
}

$parser->walk_parts(
  sub {
    my ($part) = @_;
    return if $part->subparts;

    if ($part->content_type =~ m{application/octet-stream}i){
      push @out, [ $part->filename, $part->body ];
    }
  }
);

$out[0][1] > io($out[0][0]);
```

7 Use Cases

- Dancer2::Plugin::Email (Email::Sender)
- Helpdesk::Integration
- Sympa
- Spamassassin

7.1 Helpdesk::Integration

- Request Tracker
  - Web-Interface
  - Email-Interface
- Search IMAP folder
- Parse emails
- Create/update ticket through REST API

8 Finish

8.1 Questions

8.2 The end