Working with Perl modules

Lesson 4

Sending and accessing Mail

Sending Mail from Perl

- Several levels of perl support possible
  - Sending mail using sendmail
  - Sending mail using a mail reader
  - Sending mail with the MailTools suite
  - Sending mail with Net::SMTP
  - Coding the SMTP protocol steps manually
- Depending on the needs (speed, flexibility, development time) the method has to be chosen

Sending mail with sendmail

- Easy to understand, similar what you would do on the command line
  - Open a pipe to sendmail
  - Sending the mail (headers, body) down the pipe
  - Closing the pipe
  - Done
- Disadvantages:
  - OS dependent (Windows usually without sendmail)
  - High resource usage (forking a process)

Sending with sendmail: example

```
open (PIPE, "|/usr/lib/sendmail -t") or die
  "sendmail open failed";
print PIPE "From: testuser\@desy.de\n";
print PIPE "To: postmaster\@desy.de\n";
print PIPE "Subject: test of service\n\n";
print PIPE <<"EOF";
Das ist ein Test
Viele Gruesse, Testuser
EOF
close PIPE or die "sendmail failure";
```

Sending with a mail reader

- Same procedure as with sendmail
  - Most mail readers do have delivery mode
  - Replace in the previous example sendmail by e.g. **`mail`**
- Same drawbacks as sendmail
- Advantages
  - some clients are configured to send mail to a well known server, not to the local sendmail
  - No need to write headers, using options instead

```
open P,"|mail -s 'Test' account\@desy.de";
```

Sending with Mail::Mailer

- Part of the MailTools package (installed at DESY)
- Higher level of abstraction than Net::SMTP
  - Is using Net::SMTP and libnet
  - Knows about the default mail server, if libnet configured at install time (not done on purpose!)
- Advantages
  - Nearly as simple as sending with sendmail directly
  - Free choice of mail server
  - Independent of OS specific binaries, no process forked

Mail::Mailer example

```
use Mail::Mailer;
my $mailer = Mail::Mailer->new("smtp",
                Server=> "smtp.desy.de");
$mailer->open({From => "testuser\@desy.de",
                To   => ["postmaster\@desy.de"],
                Subject => "Test"});
print $mailer <<"EOF";
Das ist ein weiterer Test
EOF
$mailer->close or die "mail sending failure";
```

Sending mail with Net::SMTP

- Apart from doing it by hand most flexible
  - Fine grained control over SMTP protocol
  - Separate handling of header and envelope
  - Only dependent on libnet module (again: no default smtp server configured at DESY)
  - A little more verbose than Mail::Mailer
  - OS independent

Net::SMTP example

```
use Net::SMTP qw(smtp);
my $mailer = Net::SMTP->new("smtp.desy.de")
             or die $@;
$mailer->mail("testuser\@desy.de");
$mailer->to  ("postmaster\@desy.de");
$mailer->data();
$mailer->datasend("From: testuser\@desy.de\n");
$mailer->datasend("To: account\@desy.de"\n\n");
$mailer->datasend("Hallo\n");
$mailer->dataend();
$mailer->quit or die "mail sending failure";
```

Talking raw SMTP

- Making use of modules: Net::Telnet
- Works like using telnet from the command line
- You need to be familiar with the SMTP commands
  - See RFC 2821
- Net::Telnet not installed at DESY
  - There are other ways to do it

Composing a MIME message

- Several methods possible
  - MIME-tools provide mechanisms to do it
  - MIME::Lite is a bit simpler (but not installed at DESY)
- Sending MIME is usually a bad idea
  - Waste of resources
  - Mail protocol not designed to handle huge data

Compose script

```
use MIME::Entity;
$msg = MIME::Entity->build(
        Type=>"multipart/mixed",
        From=>"somebody\@desy.de",
        To=>"recipient\@desy.de",
        Subject=>"MIME example");
$msg->attach(Path=>"./mime_send.pl");
$msg->attach(Data=>"Hello world!");
open MAIL, "|/usr/lib/sendmail -t";
$msg->print(\*MAIL);
close MAIL;
```

Accessing Mail using IMAP

- For access to INBOX usage of the IMAP protocol
- Access granted only after authentication
  - Authentication by LOGIN (IMAP) requires password
  - May be forbidden on unencrypted channels
  - Authentication by AUTHENTICATE requires authentication method
- AUTHENTICATE implemented with Authen::SASL
  - Free choice of method (if supported by server)
  - Tested with Kerberos4 and Kerberos5

Authentication to IMAP

- Use of SASL can eliminate need for passwd dialog
  - Automation of INBOX related tasks possible
- After authentication more IMAP commands useable
  - e.g. fetching the headers
- Auth method is being called within Mail::IMAPClient
- Sample script "access_inbox.pl" demonstrates this technique

Sample script access_inbox.pl

```
use Mail::IMAPClient;
use Authen::SASL;
use MIME::Base64;
# make the three following vars persistent for gssapi_auth
my $gss_api_step = 0;
my ($sasl, $conn);
my $user = getusername();
my $host = 'mail1.ifh.de';
my $imap = Mail::IMAPClient->new(
     Server  => $host,
     User    => $user,
) or die "couldn't connect to $host port 143: $!\n";
$imap->authenticate('GSSAPI', \&gssapi_auth)
   or die "Could not authenticate:$@\n";
print "Done\n";
$imap->logout or die "Could not logout: $@\n";
exit;
```

access_inbox.pl (cont.)

```
sub getusername {
  return getpwuid($<);
}
sub gssapi_auth {
  $gss_api_step++;
  if ( $gss_api_step == 1 ) {
    $sasl = Authen::SASL->new(mechanism => 'GSSAPI',
      callback  => { user => \&getusername }
    );
    $conn = $sasl->client_new('imap', $host);
    my $mesg = $conn->client_start;
    return encode_base64($mesg, "");
  } else {
    my $mesg=$conn->client_step(decode_base64($_[0]));
    return encode_base64($mesg, '');
  }
}
```

Secure access to INBOX with SSL

- If Kerberos (4/5) not supported by server
  - Avoid clear text password authentication
  - Password would be transmitted unencrypted
  - Use SSL to encrypt the whole connection
- Module IO::Socket::SSL comes in handy
- Only few changes required to previous example
  - Should work according to documentation
  - Was not successful for me

Secure access of the INBOX

```
use IO::Socket::SSL;
use Mail::IMAPClient;
my $sock =IO::Socket::SSL->new(PeerAddr=>'mail1.ifh.de',
                               PeerPort=>'993',
                               Proto=>'tcp');
# insert here the password dialog from Lesson 1
my $imap = Mail::IMAPClient->new(
    Socket => $sock,
    User   => $user,
    Password => $pass
) or die "couldn't connect to $host port 143: $!\n";
# now you should set the connected state and call $imap->login
```

Access to local mail folders

- Demonstration for mbox (UNIX) format
- Using same package MailTools
- Usually no further authentication required
- Very well suited for automating some tasks
  - checking mails for spam using Spamassassin
  - displaying headers of mails (sorted e.g. by mail size)
  - extracting mail addresses

Access to mbox folder

```
use Mail::Util(qw(read_mbox));
use Mail::Header;

# read from STDIN
my $msgs = read_mbox('-');
my $mails = $#$msgs + 1;
my $i = 0;
print "$mails mails in folder, the first 9 are\n";

for my $msg ( @$msgs ) {
  last if $i++ >= 9;
  my $head = new Mail::Header($msg);
  print "$i ", $head->get("Subject");
}
```