NAME

Mail::Box−Cookbook − Examples how to use Mail::Box

DESCRIPTION

The Mail::Box package is a suite of classes for accessing and managing email folders in a folder-independent manner. This manual demonstrates a few simple applications. Please contribute with examples and fixes. It may also help to have a look at the programs included in the `scripts/` and the `examples/` directories of the distribution.

The Manager

For more details about all the packages which are involved in the Mail::Box suite you have to read Mail::Box−Overview. But you do not need to know much if you want to use the Mail::Box suite.

Please use the manager to open your folders. You will certainly benefit from it. The manager takes care of detecting which folder type you are using, and which folders are open. The latter avoids the accidental re-opening of an already open folder.

The `examples/open.pl` script contains mainly

```
my $mgr    = Mail::Box::Manager−>new;
my $folder = $mgr−>open($filename);
foreach my $message ($folder−>messages) {
    print $message−>get('Subject') || '<no subject>', "\n";
}
$folder−>close;
```

which shows all the most important functions. It will cause all subjects of the messages in the indicated folder to be listed. So: although the number of packages included in the Mail::Box module is huge, only little is needed for normal programs.

In stead of calling `close` on the folder, you may also call

```
$mgr−>closeAllFolders;
```

If you forget to close a folder, changes will not be written. This may change in the future.

Multi part messages

In early days of Internet, multi-part messages were very rare. However, in recent years, a large deal of all transmitted message have attachments. This makes handling of the bodies of messages a bit harder: when a message contains more than one part, which part is then the most important to read?

To complicate life, multi-parts can be nested: each part may be a multi-part by itself. This means that programs handling the message content must be recursive or skip multi-parts.

The central part of the `examples/multipart.pl` script reads:

```
foreach my $message ($folder−>messages) {
    show_type($message);
}

show_type($) {
    my $msg = shift;
    print $msg−>get('Content−Type'), "\n";

    if($msg−>isMultipart) {
        foreach my $part ($msg−>parts) {
            show_type($part);
        }
    }
}
```

Each part is a message by itself. It has a header and a body. A multipart message has a special body: it contains a list of parts and optionally also a preamble and an epilogue, which are respectively the lines

before and after the parts. These texts may be ignored, because they are only descriptive on how the multi-part was created.

Filter

The target is to select a few messages from one folder, to move them to an other. The `examples/takelarge.pl` script demonstrates how to achieve this. **Be war ned:** it will replace your input folder!

As abstract of the crucial part:

```
my $inbox = $mgr->open('inbox', access => 'rw');
my $large = $mgr->open('large', access => 'a', create => 1);

foreach my $message ($inbox->messages) {
    next if $message->size < $size;
    $mgr->moveMessage($large, $message);
}

$inbox->close;
$large->close;
```

The `inbox` is opened for read and write: first read all messages, and then write the smaller folder without moved messages back. The `large` folder is created if the file does not exist yet. In any case, messages will be added to the end of the folder.

The manager is needed to move the message: to unregister the message from the first folder, and reregister it in the second. You can move more messages at once, if you like. When you move to a folder which is not open, you even better do that: it will be faster:

```
my @move = grep {$_->size >= $size} $inbox->messages;
$mgr->moveMessage($large, @move);
```

In this example, the `size` of the message determines whether the message is moved or not. Of course, there are many other criteria you can use. For instance, use `timestamp` to find old messages:

```
use constant YEAR => 365 * 24 * 60 * 60;
my $now = time;
my @old = grep {$_->timestamp - $now > YEAR} $inbox->messages;
$mgr->moveMessage($oldbox, @old);
```

Create a reply

The complex message treatment is implemented in Mail::Message::Construct and automatically loaded when needed. It is sufficient to simply call `reply` on any message:

```
my $folder  = ...;
my $message = $folder->message(8);
my $reply   = $message->reply;

$folder->addMessage($reply);
$reply->print;
```

The method is quite complex, as demonstrated by `examples/reply.pl`, in which the construction of a reply-message is shown.

Three kinds of reply messages can be made: one which does not include the original message at all (NO), then one which inlines the original message quoted (INLINE), and as third possibility the original message as attachment (ATTACH).

The `include` parameter selects the kind of reply. When you reply to binary or multi-part messages, INLINE will automatically promoted to ATTACH. By default text will be stripped from the original senders signature. Multi-part messages are stripped from attachments which qualify as signature. In case a multi-part (after stripping) only contains one part, and that INLINE is requested, it will be 'flattened': the reply

may be a single-part.

Have a look at the parameters which can be passed to reply in Mail::Message::Construct. For a single-part reply, the return will be

```
prelude
quoted original
postlude
--
signature
```

A multipart body will be

```
part 1: prelude
        [ see attachment ]
        postlude
part 2: stripped original multipart
part 3: signature
```

Build a message

There are three ways to create a message which is not a reply:

- **Mail::Message::buildFromBody()**

  Start with creating a body, and transform that into a message.

- **Mail::Message::build()**

  create the whole message at once.

- **Mail::Message::read()**

  read a message from a file-handle, scalar, or array of lines.

All three methods are implemented in Mail::Message::Construct. Please, do yourself a favor, and give preference to the `build*` methods, over the `read`, because they are much more powerful. Use the`read` only when you have the message on STDIN or an array of lines which is supplied by an external program.

Very important to remember from now on: information about the content of the body (the `Content-*` lines in the header) is stored within the body object, for as long as the body is not contained with a message object.

For instance, `$message` method `decoded` returns the decoded body of the `$message`. It is a body object by itself, however outside a real message. Then you may want to play around with it, by concatenating some texts: again resulting in a new body. Each body contains the right`Content-*` information. Then, finally, you create a message specifying the body and extra header lines. At that moment you need to specify the source and destination addresses (the `From` and `To` lines>). At that moment, the body will automatically be encoded to be acceptable for mail folders and transmission programs.

```
my $body = Mail::Message::Body->new
 ( mime_type         => 'text/css'
 , transfer_encoding => '8bit'
 , data              => \@lines
 );
```

Above example creates a body, with explicitly stating what kind of data is stored in it. The default mime type is `text/plain`. The transfer encoding defaults to `none`. Each message will get encoded on the moment it is added to a message. The default encoding depends on the mime type.

To start with the first way to create a message. This solution provides maximum control over the message creation. Quite some work is hidden for you when executing the next line.

```
my $message = Mail::Message->buildFromBody
  ( $body
  , From => 'me@example.com'
  , To   => 'you@anywhere.net'
  , Cc   => [ Mail::Address->parse($groupalias) ]
  );
```

For header lines, you may specify a string, an address object (Mail::Address), or an array of such addresses. If you want to create multi-part messages, you need to create a multi-part body yourself first.

The second way of constructing a message uses the `build` method. A demonstration can be found in `examples/build.pl`. In only one class method call the header and the (possible multi-parted) body is created.

With the `data` option, you can specify one scalar which contains a whole body or an array of lines. Using the `file` option, a file-handle or filename specifies a body. The`attach` option refers to construed bodies and messages. Each option can be used as often as needed. If more than one source of data is provided, a multi-part message is produced.

```
my $message = Mail::Message->build
 ( From       => 'me@example.com'
 , To         => 'you@anywhere.net'
 , 'X-Mailer' => 'Automatic mailing system'
 , data       => \@lines
 , file       => 'logo.jpg'
 , attach     => $signature_body
 );
```

SEE ALSO

This module is part of Mail-Box distribution version 3.009, built on August 18, 2020. Website: *http://perl.overmeer.net/CPAN/*

LICENSE