NAME

Mail::Message::Construct::Build − building a Mail::Message from components

SYNOPSIS

```
 my $msg1 = Mail::Message->build
   ( From => 'me', data => "only two\nlines\n");

 my $msg2 = Mail::Message->buildFromBody($body);

 Mail::Message->build
   ( From     => 'me@myhost.com'
   , To       => 'you@yourhost.com'
   , Subject  => "Read our folder!"

   , data     => \@lines
   , file     => 'folder.pdf'
   )->send(via => 'postfix');
```

DESCRIPTION

Complex functionality on Mail::Message objects is implemented in different files which are autoloaded. This file implements the building of messages from various simpler components.

METHODS

Constructing a message

Mail::Message−>**build**([$message|$part|$body], `$content`)

Simplified message object builder. In case a`$message` or message `$part` is specified, a ne w message is created with the same body to start with, but new headers. A `$body` may be specified as well. However, there are more ways to add data simply.

The `$content` is a list of key-value pairs and header field objects. The keys which start with a capital are used as header-lines. Lower-cased fields are used for other purposes as listed below. Each field may be used more than once. Pairs where the value is `undef` are ignored.

If more than one `data`, `file`, and `attach` is specified, a multi-parted message is created. Some `Content-*` fields are treated separately: to enforce the content lines of the produced message body **after** it has been created. For instance, to explicitly state that you wish a `multipart/alternative` in stead of the default `multipart/mixed`. If you wish to specify the type per datum, you need to start playing with Mail::Message::Body objects yourself.

This `build` method will use **buildFromBody()** when the body object has been constructed. Together, they produce your message.

```
 -Option--Default
  attach  undef
  data    undef
  file    undef
  files   [ ]
  head    undef
```

attach => BODY|PART|MESSAGE|ARRAY

One attachment to the message. Each attachment can be full `$message`, a `$part`, or a `$body`. Any `$message` will get encapsulated into a `message/rfc822` body. You can specify many items (may be of different types) at once.

```
 attach => $folder->message(3)->decoded  # body
 attach => $folder->message(3)           # message
 attach => [ $msg1, $msg2->part(6), $msg3->body ];
```

data => STRING|ARRAY−OF−LINES

The text for one part, specified as one STRING, or an ARRAY of lines. Each line, including the last, must be terminated by a newline. This argument is passed to Mail::Message::Body::new(data) to construct one.

```
  data => [ "line 1\n", "line 2\n" ]     # array of lines
  data => <<'TEXT'                       # string
 line 1
 line 2
 TEXT
```

file => FILENAME|FILEHANDLE|IOHANDLE|ARRAY

Create a body where the data is read from the specified FILENAME, FILEHANDLE, or object of type IO::Handle. Also this body is used to create a Mail::Message::Body. [2.119] You may even pass more than one file at once: 'file' and 'files' option are equivalent.

```
 my $in = IO::File−>new('/etc/passwd', 'r');

 file  => 'picture.jpg'                   # filename
 file  => \*MYINPUTFILE                   # file handle
 file  => $in                             # any IO::Handle
 files => [ 'picture.jpg', \*MYINPUTFILE, $in ]

 open my $in, '<:raw', '/etc/passwd';    # alternative for IO::File
```

files => ARRAY-OF-FILE

Alias for option `file`.

head => HEAD

Start with a prepared header, otherwise one is created.

example:

```
 my $msg = Mail::Message−>build
  ( From    => 'me@home.nl'
  , To      => Mail::Address−>new('your name', 'you@yourplace.aq')
  , Cc      => 'everyone@example.com'
  , Subject => "Let's talk",
  , $other_message−>get('Bcc')

  , data   => [ "This is\n", "the first part of\n", "the message\n" ]
  , file   => 'myself.gif'
  , file   => 'you.jpg'
  , attach => $signature
  );

 my $msg = Mail::Message−>build
  ( To     => 'you'
  , 'Content−Type' => 'text/html'
  , data   => "<html></html>"
  );
```

Mail::Message−>**buildFromBody**($body, [$head], `$headers`)

Shape a message around a `$body`. Bodies have information about their content in them, which is used to construct a header for the message. You may specify a `$head` object which is pre-initialized, or one is created for you (also when `$head` is `undef`). Next to that, more `$headers` can be specified which are stored in that header.

Header fields are added in order, and before the header lines as defined by the body are taken. They may be supplied as key-value pairs or Mail::Message::Field objects. In case of a key-value pair, the field's name is to be used as key and the value is a string, address (Mail::Address object), or array of addresses.

A `Date`, `Message-Id`, and `MIME-Version` field are added unless supplied.

example:

```
 my $type = Mail::Message::Field->new('Content-Type', 'text/html'
   , 'charset="us-ascii"');

 my @to   = ( Mail::Address->new('Your name', 'you@example.com')
            , 'world@example.info'
            );

 my $msg  = Mail::Message->buildFromBody
   ( $body
   , From => 'me@example.nl'
   , To   => \@to
   , $type
   );
```

DETAILS

Building a message

Rapid building

Most messages you need to construct are relatively simple. Therefore, this module provides a method to prepare a message with only one method call: **build()**.

*Compared to **MIME::Entity::build()***

The `build` method in MailBox is modelled after the `build` method as provided by MIMETools, but with a few simplifications:

When a keys starts with a capital, than it is always a header field
When a keys is lower-cased, it is always something else
You use the real field-names, not abbreviations
All field names are accepted
You may specify field objects between key-value pairs
A lot of facts are auto-detected, like content-type and encoding
You can create a multipart at once

Hum, reading the list above... what is equivalent? MIME::Entity is not that simple after all! Let's look at an example from MIME::Entity's manual page:

```
 ### Create the top-level, and set up the mail headers:
 $top = MIME::Entity->build(Type     => "multipart/mixed",
                            From     => 'me@myhost.com',
                            To       => 'you@yourhost.com',
                            Subject  => "Hello, nurse!");

 ### Attachment #1: a simple text document:
 $top->attach(Path=>"./testin/short.txt");

 ### Attachment #2: a GIF file:
 $top->attach(Path        => "./docs/mime-sm.gif",
              Type        => "image/gif",
              Encoding    => "base64");

 ### Attachment #3: text we'll create with text we have on-hand:
```

```
 $top->attach(Data => $contents);
```

The MailBox equivalent could be

```
 my $msg = Mail::Message->build
   ( From     => 'me@myhost.com'
   , To       => 'you@yourhost.com'
   , Subject  => "Hello, nurse!"

   , file     => "./testin/short.txt"
   , file     => "./docs/mime-sm.gif"
   , data     => $contents
   );
```

One of the simplifications is that MIME::Types is used to lookup the right content type and optimal transfer encoding. Good values for content-disposition and such are added as well.

build, starting with nothing

See **build()**.

buildFromBody, body becomes message

See **buildFromBody()**.

The Content− fields*

The various `Content-*` fields are not as harmless as they look. For instance, the ‘‘Content-Type’’ field will have an effect on the default transfer encoding.

When a message is built this way:

```
 my $msg = Mail::Message->build
  ( 'Content-Type' => 'video/mpeg3'
  , 'Content-Transfer-Encoding' => 'base64'
  , 'Content-Disposition' => 'attachment'
  , file => '/etc/passwd'
  );
```

then first a `text/plain` body is constructed (MIME::Types does not find an extension on the filename so defaults to `text/plain`), with no encoding. Only when that body is ready, the new type and requested encodings are set. The content of the body will get base64 encoded, because it is requested that way.

What basically happens is this:

```
 my $head = ...other header lines...;
 my $body = Mail::Message::Body::Lines->new(file => '/etc/passwd');
 $body->type('video/mpeg3');
 $body->transferEncoding('base64');
 $body->diposition('attachment');
 my $msg  = Mail::Message->buildFromBody($body, $head);
```

A safer way to construct the message is:

```
 my $body = Mail::Message::Body::Lines->new
  ( file              => '/etc/passwd'
  , mime_type         => 'video/mpeg3'
  , transfer_encoding => 'base64'
  , disposition       => 'attachment'
  );

 my $msg  = Mail::Message->buildFromBody
  ( $body
  , ...other header lines...
```

```
    );
```

In the latter program, you will immediately start with a body of the right type.

DIAGNOSTICS

Error: Only **build()** Mail::Message’s; they are not in a folder yet

You may wish to construct a message to be stored in a some kind of folder, but you need to do that in two steps. First, create a normal Mail::Message, and then add it to the folder. During this **Mail::Box::addMessage()** process, the message will get **coerce()**−d into the right message type, adding storage information and the like.

SEE ALSO

This module is part of Mail-Message distribution version 3.012, built on February 11, 2022. Website: *http://perl.overmeer.net/CPAN/*

LICENSE