NAME

Mail::Box::Parser − reading and writing messages

INHERITANCE

```
Mail::Box::Parser
  is a Mail::Reporter

Mail::Box::Parser is extended by
  Mail::Box::Parser::C
  Mail::Box::Parser::Perl
```

SYNOPSIS

```
# Not instatiatiated itself
```

DESCRIPTION

The `Mail::Box::Parser` manages the parsing of folders. Usually, you won't need to know anything about this module, except the options which are involved with this code.

There are two implementations of this module planned:

- Mail::Box::Parser::Perl

  A slower parser which only uses plain Perl. This module is a bit slower, and does less checking and less recovery.

- Mail::Box::Parser::C

  A fast parser written in `C`. This package is released as separate module on CPAN, because the module distribution via CPAN can not handle XS files which are not located in the root directory of the module tree. If a C compiler is available on your system, it will be used automatically.

Extends ''DESCRIPTION'' in Mail::Reporter.

METHODS

Extends ''METHODS'' in Mail::Reporter.

Constructors

Extends ''Constructors'' in Mail::Reporter.

Mail::Box::Parser−>**new**(%options)

Create a parser object which can handle one file. For mbox-like mailboxes, this object can be used to read a whole folder. In case of MH-like mailboxes, each message is contained in a single file, so each message has its own parser object.

```
−Option  --Defined in     --Default
 file                       undef
 filename                   <required>
 log       Mail::Reporter   'WARNINGS'
 mode                       'r'
 trace     Mail::Reporter   'WARNINGS'
```

file => FILE-HANDLE

Any `IO::File` or `GLOB` which can be used to read the data from. In case this option is specified, the `filename` is informational only.

filename => FILENAME

The name of the file to be read.

log => LEVEL

mode => OPENMODE

File-open mode, which defaults to `'r'`, which means 'read−only'. See `perldoc −f open` for possible modes. Only applicable when no `file` is specified.

trace => LEVEL

The parser

`$obj`->**fileChanged**()

Returns whether the file which is parsed has changed after the last time **takeFileInfo()** was called.

`$obj`->**filename**()

Returns the name of the file this parser is working on.

`$obj`->**restart**()

Restart the parser on a certain file, usually because the content has changed.

`$obj`->**start**(%options)

Start the parser by opening a file.

```
-Option--Default
 file    undef
```

file => FILEHANDLE|undef

The file is already open, for instance because the data must be read from STDIN.

`$obj`->**stop**()

Stop the parser, which will include a close of the file. The lock on the folder will not be removed (is not the responsibility of the parser).

Parsing

`$obj`->**bodyAsFile**(`$fh` [$chars, [$lines]])

Try to read one message-body from the file, and immediately write it to the specified file-handle. Optionally, the predicted number of CHARacterS and/or `$lines` to be read can be supplied. These values may be `undef` and may be wrong.

The return is a list of three scalars: the location of the body (begin and end) and the number of lines in the body.

`$obj`->**bodyAsList**([$chars, [$lines]])

Try to read one message-body from the file. Optionally, the predicted number of CHARacterS and/or `$lines` to be read can be supplied. These values may be `undef` and may be wrong.

The return is a list of scalars, each containing one line (including line terminator), preceded by two integers representing the location in the file where this body started and ended.

`$obj`->**bodyAsString**([$chars, [$lines]])

Try to read one message-body from the file. Optionally, the predicted number of CHARacterS and/or `$lines` to be read can be supplied. These values may be `undef` and may be wrong.

The return is a list of three scalars, the location in the file where the body starts, where the body ends, and the string containing the whole body.

`$obj`->**bodyDelayed**([$chars, [$lines]])

Try to read one message-body from the file, but the data is skipped. Optionally, the predicted number of CHARacterS and/or `$lines` to be skipped can be supplied. These values may be `undef` and may be wrong.

The return is a list of four scalars: the location of the body (begin and end), the size of the body, and the number of lines in the body. The number of lines may be`undef`.

`$obj`->**filePosition**([$position])

Returns the location of the next byte to be used in the file which is parsed. When a `$position` is specified, the location in the file is moved to the indicated spot first.

`$obj`->**lineSeparator**()

Returns the character or characters which are used to separate lines in the folder file. This is based on the first line of the file. UNIX systems use a single LF to separate lines. Windows uses a CR and a LF. Mac uses CR.

`$obj->`**popSeparator**()
Remove the last-pushed separator from the list which is maintained by the parser. This will return `undef` when there is none left.

`$obj->`**pushSeparator**(STRING|Regexp)
Add a boundary line. Separators tell the parser where to stop reading. A famous separator is the `From`-line, which is used in Mbox-like folders to separate messages. But also parts (*attachments*) is a message are divided by separators.

The specified STRING describes the start of the separator-line. The Regexp can specify a more complicated format.

`$obj->`**readHeader**()
Read the whole message-header and return it as list of field-value pairs. Mind that some fields will appear more than once.

The first element will represent the position in the file where the header starts. The follows the list of header field names and bodies.

example:

```
my ($where, @header) = $parser->readHeader;
```

`$obj->`**readSeparator**(%options)
Read the currently active separator (the last one which was pushed). The line (or `undef`) is returned. Blank-lines before the separator lines are ignored.

The return are two scalars, where the first gives the location of the separator in the file, and the second the line which is found as separator. A new separator is activated using **pushSeparator()**.

Internals

`$obj->`**closeFile**()
Close the file which was being parsed.

`$obj->`**defaultParserType**([$class])
Mail::Box::Parser->**defaultParserType**([$class])
Returns the parser to be used to parse all subsequent messages, possibly first setting the parser using the optional argument. Usually, the parser is autodetected; the `C`-based parser will be used when it can be, and the Perl-based parser will be used otherwise.

The `$class` argument allows you to specify a package name to force a particular parser to be used (such as your own custom parser). You have to `use` or `require` the package yourself before calling this method with an argument. The parser must be a sub-class of `Mail::Box::Parser`.

`$obj->`**openFile**($args)
Open the file to be parsed. `$args` is a ref-hash of options.

```
-Option  --Default
 filename  <required>
 mode      <required>
```

filename => FILENAME
mode => STRING

`$obj->`**takeFileInfo**()
Capture some data about the file being parsed, to be compared later.

Error handling

Extends ‘‘Error handling’’ in Mail::Reporter.

`$obj->`**AUTOLOAD**()
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj`->**addReport**($object)
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**defaultTrace**([$level]|[$loglevel, `$tracelevel`]|[$level, `$callback`])
Mail::Box::Parser->**defaultTrace**([$level]|[$loglevel, `$tracelevel`]|[$level, `$callback`])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**errors**()
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**log**([$level, [$strings]])
Mail::Box::Parser->**log**([$level, [$strings]])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**logPriority**($level)
Mail::Box::Parser->**logPriority**($level)
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**logSettings**()
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**notImplemented**()
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**report**([$level])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**reportAll**([$level])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**trace**([$level])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**warnings**()
Inherited, see ''Error handling'' in Mail::Reporter

Cleanup

Extends ''Cleanup'' in Mail::Reporter.

`$obj`->**DESTROY**()
Inherited, see ''Cleanup'' in Mail::Reporter

DIAGNOSTICS

Warning: File `$filename` changed during access.
When a message parser starts working, it takes size and modification time of the file at hand. If the folder is written, it checks whether there were changes in the file made by external programs.

Calling **Mail::Box::update()** on a folder before it being closed will read these new messages. But the real source of this problem is locking: some external program (for instance the mail transfer agent, like sendmail) uses a different locking mechanism as you do and therefore violates your rights.

Error: Filename or handle required to create a parser.
A message parser needs to know the source of the message at creation. These sources can be a filename (string), file handle object or GLOB. See new(filename) and new(file).

Error: Package `$package` does not implement `$method`.
Fatal error: the specific package (or one of its superclasses) does not implement this method where it should. This message means that some other related classes do implement this method however the class at hand does not. Probably you should investigate this and probably inform the author of the package.

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