## NAME

Mail::Message::Construct::Rebuild − modify a Mail::Message

## SYNOPSIS

```
my $cleanup = $msg->rebuild;
```

## DESCRIPTION

Modifying existing messages is a pain, certainly if this has to be done in an automated fashion. The problems are especially had when multiparts have to be created or removed. The**r ebuild()** method tries to simplify this task and add some standard features.

## METHODS

### Constructing a message

$obj->**rebuild**(%options)

Reconstruct an existing message into something new. Returned is a new message when there were modifications made, `undef` if the message has no body left, or the original message when no modifications had to be made.

Examples of use: you have a message which only contains html, and you want to translate it into a multipart which contains the original html and the textual translation of it. Or, you have a message with parts flagged to be deleted, and you want those changes be incorparted in the memory structure. Another possibility: clear all the resent groups (see Mail::Message::Head::ResentGroup) from the header, before it is written to file.

Reconstructing is a hazardous task, where multi level multiparts and nested messages come into play. The rebuild method tries to simplify handing these messages for you.

```
 -Option         --Default
  extra_rules      []
  keep_message_id  <false>
  rules            <see text>
```

extra_rules => ARRAY

The standard set of rules, which is the default for the `rules` option, is a moderest setting. In stead of copying that list into a full set of rules of your own, you can also specify only some additional rules which will be prependend to the default rule set.

The order of the rules is respected, which means that you do not always need to rewrite the whole rule is (see `rule` option). For instance, the extra rule of `removeDeletedParts` returns an `undef`, which means that it cancels the effect of the default rule `replaceDeletedParts`.

keep_message_id => BOOLEAN

The message-id is an unique identification of the message: no two messages with different content shall exist anywhere. However in practice, when a message is changed during transmission, the id is often incorrectly not changed. This may lead to complications in application which see both messages with the same id.

rules => ARRAY

The ARRAY is a list of rules, which each describe an action which will be called on each part which is found in the message. Most rules probably won't match, but some will bring changes to the content. Rules can be specified as method name, or as code reference. See the ''DETAILS'' chapter in this manual page, and **recursiveRebuildPart()**.

By default, only the relatively safe transformations are performed: `replaceDeletedParts`, `descendMultiparts`, `descendNested`, `flattenMultiparts`, `flattenEmptyMultiparts`. In the future, more safe transformations may be added to this list.

example:

```
# remove all deleted parts
my $cleaned = $msg->rebuild(keep_message_id => 1);
$folder->addMessage($cleaned) if defined $cleaned;

# Replace deleted parts by a place-holder
my $cleaned = $msg->rebuild
  ( keep_message_id => 1
  , extra_rules => [ 'removeEmpty', 'flattenMultiparts' ]
  );
```

**Internals**

$obj->**recursiveRebuildPart**($part, %options)

```
-Option--Default
 rules   <required>
```

rules => ARRAY-OF-RULES

Rules are method names which can be called on messages and message parts objects. The ARRAY can also list code references which can be called. In any case, each rule will be called the same way:

```
$code->(MESSAGE, PART)
```

The return can be `undef` or any complex construct based on a Mail::Message::Part or coerceable into such a part. For each part, all rules are called in sequence. When a rule returns a changed object, the rules will start all over again, however `undef` will immediately stop it.

## DETAILS

### Rebuilding a message

Modifying an existing message is a complicated job. Not only do you need to know what you are willing to change, but you have to take care about multiparts (possibly nested in multiple levels), rfc822 encapsulated messages, header field consistency, and so on. The **rebuild()** method let you focus on the task, and takes care of the rest.

The **rebuild()** method uses rules to transform the one message into an other. If one or more of the rules apply, a new message will be returned. A simple numeric comparison tells whether the message has changed. For example

```
print "No change"
   if $message == $message->rebuild;
```

Transformation is made with a set of rules. Each rule performs only a small step, which makes is easily configurable. The rules are ordered, and when one makes a change to the result, the result will be passed to all the rules again until no rule makes a change on the part anymore. A rule may also return`undef` in which case the part will be removed from the (resulting) message.

*General rules*

This sections describes the general configuration rules: all quite straight forward transformations on the message structure. The rules marked with (*) are used by default.

- descendMultiparts (*)

  Apply the rules to the parts of (possibly nested) multiparts, not only to the top-level message.

- descendNested (*)

  Apply the rules to the `message/rfc822` encapsulated message as well.

- flattenEmptyMultiparts (*)

  Multipart messages which do not have any parts left are replaced by a single part which contains the preamble, epilogue and a brief explanation.

- flattenMultiparts (*)

  When a multipart contains only one part, that part will take the place of the multipart: the removal of a level of nesting. This way, the preamble and epilogue of the multipart (which do not have a meaning, officially) are lost.

- flattenNesting

  Remove the `message/rfc822` encapsulation. Only the content related lines of the encapsulated body are preserved one level higher. Other information will be lost, which is often not too bad.

- removeDeletedParts

  All parts which are flagged for deletion are removed from the message without leaving a trace. If a nested message is encountered which has its encapsulated content flagged for deletion, it will be removed as a whole.

- removeEmptyMultiparts

  Multipart messages which do not have any parts left are removed. The information in preamble and epiloge is lost.

- removeEmptyBodies

  Simple message bodies which do not contain any lines of content are removed. This will loose the information which is stored in the headers of these bodies.

- replaceDeletedParts (*)

  All parts of the message which are flagged for deletion are replace by a message which says that the part is deleted.

You can specify a selection of these rules with rebuild(rules) and rebuild(extra_rules).

*Conversion rules*

This section describes the rules which try to be smart with the content. Please contribute with ideas and implementations.

- removeHtmlAlternativeToText

  When a multipart alternative is encountered, which contains both a plain text and an html part, then the html part is deleted. Especially useful in combination with the `flattenMultiparts` rule.

- textAlternativeForHtml

  Any `text/html` part which is not accompanied by an alternative plain text part will have one added. You must have a working Mail::Message::Convert::HtmlFormatText, which means that HTML::TreeBuilder and HTML::FormatText must be installed on your system.

  When you are planning to create an automatic html to plain text filter for your email, then have a look at *https://github.com/logological/mimestrip*

  **. Example: using parameter with `textAlternativeForHtml`**

  ```
  my $result = $msg->rebuild
    ( extra_rules => [ 'textAlternativeForHtml' ]
    , textAlternativeForHtml => { leftmargin => 0 }
    );
  ```

- removeExtraAlternativeText

  [2.110] When a multipart alternative is encountered, deletes all its parts except for the last part (the preferred part in accordance with RFC2046). In practice, this normally results in the alternative plain text part being deleted of an html message. Useful in combination with the `flattenMultiparts` rule.

*Adding your own rules*

If you have designed your own rule, please consider contributing this to Mail::Box; it may be useful for other people as well.

Each rule is called

```
my $new = $code->($message, $part, %options)
```

where the `%options` are defined by the `rebuild()` method internals. At least the `rules` option is passed, which is a full expansion of all the rules which will be applied.

Your subroutine shall return `$part` if no changes are needed, `undef` if the part should be removed, and any newly constructed `Mail::Message::Part` when a change is required. It is easiest to start looking at the source code of this package, and copy from a comparible routine.

When you have your own routine, you simply call:

```
my $rebuild_message = $message->rebuild
 ( extra_rules => [ \&my_own_rule, 'other_rule' ] );
```

## DIAGNOSTICS

Error: No rebuild rule `$name` defined.

## 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