NAME

Mail::Box::Thread::Node − one node in a message thread

INHERITANCE

```
 Mail::Box::Thread::Node
   is a Mail::Reporter
```

SYNOPSIS

```
 my $node = Mail::Box::Thread::Node−>new;
 $node−>addMessage($message);
 ...
```

DESCRIPTION

The `Mail::Box::Thread::Node` maintains one node in the linked list of threads. Each node contains one message, and a list of its follow-ups. Next to that, it refers to its own ancestor and contains information about the trustworthiness of that relationship.

To complicate things a little, because the thread-manager can maintain multiple folders, and merge there content, you may find the same message in more folders. All versions of the same message (based on message-id) are stored in the same node.

Extends ‘‘DESCRIPTION’’ in Mail::Reporter.

METHODS

Extends ‘‘METHODS’’ in Mail::Reporter.

Constructors

Extends ‘‘Constructors’’ in Mail::Reporter.

Mail::Box::Thread::Node−>**new**(%options)

You will not call this method yourself. The Mail::Box::Thread::Manager object will call it to construct `Mail::Box::Thread::Node` objects. Either a `message` or a `messageId` must be supplied.

```
 −Option    −−Defined in     −−Default
  dummy_type                   undef
  log         Mail::Reporter   'WARNINGS'
  message                      undef
  messageId                    undef
  trace       Mail::Reporter   'WARNINGS'
```

dummy_type => CLASS

Indicates the class name of dummy messages. Dummy messages are placeholders in a Mail::Box::Thread::Manager data structure.

log => LEVEL

message => MESSAGE

The MESSAGE which is stored in this node. The message must be a Mail::Box::Message.

messageId => MESSAGE-ID

The MESSAGE-ID for the message which is stored in this node. Only specify it when you don’t have the message yet.

trace => LEVEL

The thread node

`$obj`−>**addMessage**($message)

Add one message to the thread node. If the node contains a dummy, then the dummy is replaced. Otherwise, the messages is added to the end of the list.

`$obj`−>**expand**([BOOLEAN])

Returns whether this (part of the) folder has to be shown expanded or not. This is simply done by a label, which means that most folder types can store this.

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

Returns true if the message is a dummy. A dummy is a "hole" in a thread which has follow-ups but does not have a message.

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

Get the message which is stored in this thread node. NOTE: the same message may be located in many folders at the same time, and these folders may be controlled by the same thread manager.

In scalar context, this method returns the first instance of the message that is not deleted. If all instances are flagged for deletion, then you get the first deleted message. When the open folders only contain references to the message, but no instance, you get a dummy message (see Mail::Message::Dummy).

In list context, all instances of the message which have been found are returned.

example:

```
my $threads = $mgr->threads(folders => [$draft, $sent]);
my $node    = $draft->message(1)->thread;

foreach my $instance ($node->message) {
   print "Found in ", $instance->folder, ".\n";
}

print "Subject is ", $node->message->subject, ".\n";
```

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

Return the message-id related to this thread node. Each of the messages listed in this node will have the same ID.

The thread order

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

Returns the list of follow-ups to this thread node. This list may contain parsed, not-parsed, and dummy messages.

`$obj->`**followedBy**($threads)

Register that the `$threads` are follow-ups to this message. These follow-ups need not be related to each other in any way other than sharing the same parent.

Defining the same relation more than once will not cause information to be duplicated.

`$obj->`**follows**($thread, `$quality`)

Register that the current thread is a reply to the specified `$thread`. The `$quality` of the relation is specified by the second argument. The method returns`undef` if the link is not accepted in order to avoid circular references.

The relation may be specified more than once, but only the most confident relation is used. For example, if a reply ($quality equals `REPLY`) is specified, later calls to the follow method will have no effect. If `follows` is called with a `$quality` that matches the current quality, the new thread overrides the previous.

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

Returns the message(s) to which the message in this node replies. In scalar context, this method will return the message to which the message in this node replies. This message object may be a dummy message.

If the message seems to be the first message of a thread, the value `undef` is returned. (Remember that some MUA are not adding reference information to the message's header, so you can never be sure a message is the start of a thread)

In list context, this method returns a second string value indicating the confidence that the messages are related. When extended thread discovery is enabled, then some heuristics are applied to determine if messages are related. Values for the STRING may be:

- `'REPLY'`

  This relation was directly derived from an ‘in−reply−to’ message header field. The relation has a high confidence.

- `'REFERENCE'`

  This relation is based on information found in a ‘Reference’ message header field. One message may reference a list of messages which precede it in the thread. The heuristic attempts to determine relationships between messages assuming that the references are in order. This relation has a lower confidence.

- `'GUESS'`

  The relation is a big guess, with low confidence. It may be based on a subject which seems to be related, or commonalities in the message’s body.

More constants may be added later.

example:

```
 my $question = $answer−>repliedTo;
 my ($question, $quality) = $answer−>repliedTo;
 if($question && $quality eq 'REPLY') { ... };
```

`$obj`−>**sortedFollowUps**([$prepare, [$compare]])
Returns the list of **followUps()**, but sorted. By default sorting is based on the estimated time of the reply. See **startTimeEstimate()**.

On the whole thread

Some convenience methods are added to threads, to simplify retrieving information from it.

`$obj`−>**endTimeEstimate**()
Returns a guess as to when the thread has ended (although you never know for sure whether there fill follow messages in the future).

`$obj`−>**ids**()
Returns all the ids in the thread starting at the current thread node.

example:

```
 $newfolder−>addMessages($folder−>ids($thread−>ids));
 $folder−>delete($thread−>ids);
```

`$obj`−>**numberOfMessages**()
Number of messages in the thread starting at the current thread node, but not counting the dummies.

`$obj`−>**recurse**(CODE)
Execute a function for all sub-threads. If the subroutine returns true, sub-threads are visited recursively. Otherwise, the current branch traversal is aborted. The routine is called with the thread-node as the only argument.

`$obj`−>**startTimeEstimate**()
Returns a guess as to when the thread was started. Each message contains various date specifications (each with various uncertainties resulting from timezones and out-of-sync clocks). One of these date specifications is used as the timestamp for the message. If the node contains a dummy message the lowest timestamp of the replies is returned. Otherwise the estimated timestamp of the node’s message is returned.

`$obj`−>**threadMessages**()
Returns all the messages in the thread starting at the current thread node. This list will not include dummies.

example:

```
 my @t = $folder->message(3)
               ->threadStart
               ->threadMessages;
```

$obj->**threadToString**([CODE])
Translate a thread into a string. The string will contain at least one line for each message which was found, but tries to fold dummies. This is useful for debugging, but most message readers will prefer to implement their own thread printer.

The optional CODE argument is a reference to a routine which will be called for each message in the thread. The routine will be called with the message as the first argument. The default shows the subject of the message. In the first example below, this routine is called seven times.

example:

```
 print $node->threadToString;
```

may result in

```
 Subject of this message
 |- Re: Subject of this message
 |-*- Re: Re: Subject of this message
 | |- Re(2) Subject of this message
 | |- [3] Re(2) Subject of this message
 | `- Re: Subject of this message (reply)
 `- Re: Subject of this message
```

The '*' represents a missing message (a "dummy" message). The '[3]' presents a folded thread with three messages.

```
 print $node->threadToString(\&show);

 sub show($) {
    my $message = shift;
    my $subject = $message->head->get('subject');
    length $subject ? $subject : '<no subject>';
 }
```

$obj->**totalSize**()
Returns the sum of the size of all the messages in the thread.

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, $callback])
Mail::Box::Thread::Node->**defaultTrace**([$level]|[$loglevel, $tracelevel]|[$level, $callback])
Inherited, see "Error handling" in Mail::Reporter

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

$obj->**log**([$level, [$strings]])
Mail::Box::Thread::Node->**log**([$level, [$strings]])
Inherited, see "Error handling" in Mail::Reporter

`$obj`−>**logPriority**($level)
Mail::Box::Thread::Node−>**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

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-Box distribution version 3.009, built on August 18, 2020. Website: *http://perl.overmeer.net/CPAN/*

LICENSE