NAME

Mail::Box::Manager − manage a set of folders

INHERITANCE

```
 Mail::Box::Manager
   is a Mail::Reporter

 Mail::Box::Manager is extended by
   Mail::Box::Manage::User
```

SYNOPSIS

```
 use Mail::Box::Manager;
 my $mgr     = new Mail::Box::Manager;

 # Create folder objects.
 my $folder   = $mgr−>open(folder => $ENV{MAIL});
 my $message1 = $folder−>message(0);
 $mgr−>copyMessage('Draft', $message);

 my @messages = $folder−>message(0,3);
 $mgr−>moveMessage('Outbox', @messages, create => 1 );
 $mgr−>close($folder);

 # Create thread−detectors (see Mail::Box::Thread::Manager)
 my $t       = $mgr−>threads($inbox, $outbox);

 my $threads = $mgr−>threads(folder => $folder);
 foreach my $thread ($threads−>all)
 {   $thread−>print;
 }

 $mgr−>registerType(mbox => 'Mail::Box::MyType');
```

DESCRIPTION

The manager keeps track on a set of open folders and a set of message-thread supporting objects. You are not obliged to use this object (you can directly create a Mail::Box::Mbox if you prefer), but you will create more portable and safer code if you do use it.

Extends "DESCRIPTION" in Mail::Reporter.

METHODS

Extends "METHODS" in Mail::Reporter.

Constructors

Extends "Constructors" in Mail::Reporter.

Mail::Box::Manager−>**new**($args)

```
 −Option              −−Defined in     −−Default
  autodetect                            undef
  default_folder_type                   'mbox'
  folder_types                          <all standard types>
  folderdir                             [ '.' ]
  folderdirs                            <synonym for C<folderdir>>
  log                   Mail::Reporter   'WARNINGS'
  trace                 Mail::Reporter   'WARNINGS'
```

autodetect => TYPE|ARRAY−OF−TYPES

Select only a subset of the folder types which are implemented by MailBox to be detected automatically. This may improve the auto-detection of folder types. Normally, all folder types will be tried when a folder's name is incorrect, but this option limits the types which are checked and therefore may respond faster.

default_folder_type => NAME|CLASS
Specifies the default folder type for newly created folders. If this option is not specified, the most recently registered type is used (see **registerType()** and the new(folder_types) option.

folder_types => NEW-TYPE | ARRAY-OF-NEW-TYPES
Add one or more new folder types to the list of known types. The order is important: when you open a file without specifying its type, the manager will start trying the last added list of types, in order.

Each TYPE is specified as an array which contains name, class, and defaults for options which overrule the usual defaults. You may specify folder-specific defaults as OPTIONS. They override the settings of the manager.

folderdir => DIRECTORY
The default directory, or directories, where folders are located. The `Mail::Box::Manager` can autodetect the existing folder-types. There may be different kinds of folders opened at the same time, and messages can be moved between those types, although that may result in a loss of information depending on the folder types.

folderdirs => [DIRECTORIES]
log => LEVEL
trace => LEVEL

Attributes

`$obj`->**defaultFolderType**()
Returns the default folder type, some class name.

`$obj`->**folderTypes**()
Returns the list of currently defined folder types.

example:

```
 print join("\n", $manager->folderTypes), "\n";
```

`$obj`->**folderdir**()
In list context, this returns all folderdirs specified. In SCALAR context only the first.

`$obj`->**registerType**($type, `$class`, `%options`)
With `registerType` you can register one `$type` of folders. The `$class` is compiled automatically, so you do not need to `use` them in your own modules. The `$type` is just an arbitrary name.

The added types are prepended to the list of known types, so they are checked first when a folder is opened in autodetect mode.

example:

```
 $manager->registerType(mbox => 'Mail::Box::Mbox',
     save_on_exit => 0, folderdir => '/tmp');
```

Manage open folders

`$obj`->**close**($folder, `%options`)
`close` removes the specified folder from the list of open folders. Indirectly it will update the files on disk if needed (depends on the Mail::Box::new(save_on_exit) flag for each folder). `%options` are passed to **Mail::Box::close()** of the folder.

The folder's messages will also be withdrawn from the known message threads. You may also close the folder directly. The manager will be informed about this event and take appropriate actions.

```
 −Option        −−Default
  close_by_self  <false>
```

close_by_self => BOOLEAN
  Used internally to avoid confusion about how the close was started. Do not change this.

example:

```
 my $inbox = $mgr−>open('inbox');
 $mgr−>close($inbox);
 $inbox−>close;        # alternative
```

`$obj`−>**closeAllFolders**(, `%options`)
  `closeAllFolders` calls **close()** for each folder managed by this object. It is called just before the program stops (before global cleanup).

`$obj`−>**isOpenFolder**($folder)
  Returns true if the `$folder` is currently open.

example:

```
 print "Yes\n" if $mgr−>isOpenFolder('Inbox');
```

`$obj`−>**open**([$foldername], `%options`)
  Open a folder which name is specified as first parameter or with the option flag `folder`. The folder type is autodetected unless the `type` is specified.

  `open` carries options for the manager which are described here, but may also have additional options for the folder type. For a description of the folder options, see the options to the constructor **Mail::Box::new()** for each type of mail box.

```
 −Option       −−Default
  authenticate  'AUTO'
  create        <false>
  folder        $ENV{MAIL}
  folderdir     '.'
  type          <first, usually C<mbox>>
```

authenticate => TYPE|ARRAY−OF−TYPES|'AUTO'
  The TYPE of authentication to be used, or a list of TYPES which the client prefers. The server may provide preferences as well, and that order will be kept. This option is only supported by a small subset of folder types, especially by POP and IMAP.

create => BOOLEAN
  Create the folder if it does not exist. By default, this is not done. The `type` option specifies which type of folder is created.

folder => NAME|URL
  Which folder to open, specified by NAME or special URL. The URL format is composed as

```
 type://username:password@hostname:port/foldername
```

  Like real URLs, all fields are optional and have smart defaults, as long as the string starts with a known folder type. Far from all folder types support all these options, but at least they are always split-out. Be warned that special characters in the password should be properly url-encoded.

  When you specify anything which does not match the URL format, it is passed directly to the `new` method of the folder which is opened.

folderdir => DIRECTORY
  The directory where the folders are usually stored.

type => FOLDERTYPENAME|FOLDERTYPE

Specify the type of the folder. If you do not specify this option while opening a folder for reading, the manager checks all registered folder types in order for the ability to open the folder. If you open a new folder for writing, then the default will be the most recently registered type. (If you add more than one type at once, the first of the list is used.)

Currently, the types are `mbox`, `mh`, `maildir`, `pop3`, `pop3s`, `imap4`, and `imap4s`. You may also use names `pop`, `pops`, `imap`, and `imaps`.

example: opening folders via the manager

```
 my $jack  = $manager->open(folder => '=jack',
    type => 'mbox');

 my $rcvd  = $manager->open('myMail',
    type => 'Mail::Box::Mbox', access => 'rw');

 my $inbox = $manager->open('Inbox')
    or die "Cannot open Inbox.\n";

 my $pop   = 'pop3://myself:secret@pop3.server.com:120/x';
 my $send  = $manager->open($url);

 my $send  = $manager->open(folder => '/x',
   type => 'pop3', username => 'myself', password => 'secret'
   server_name => 'pop3.server.com', server_port => '120');
```

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

Returns a list of all open folders.

Manage existing folders

`$obj`->**delete**($foldername, `%options`)

Remove the named folder. The`%options` are the same as those for **open()**.

The deletion of a folder can take some time. Dependent on the type of folder, the folder must be read first. For some folder-types this will be fast.

```
 -Option   --Default
  recursive  <folder's default>
```

recursive => BOOLEAN

Some folder can only be recursively deleted, other have more flexibility.

Move messages to folders

`$obj`->**appendMessage**([$folder|$foldername], `$messages`, `%options`)

Append one or more messages to a folder (therefore, an `appendMessages()` is defined as well). You may specify a `$foldername` or an opened folder as the first argument. When the name is that of an open folder, it is treated as if the folder-object was specified, and not directly access the folder-files. You may also specify the foldername as part of the options list.

If a message is added to an already opened folder, it is only added to the structure internally in the program. The data will not be written to disk until a write of that folder takes place. When the name of an unopened folder is given, the folder is opened, the messages stored on disk, and then the folder is closed.

A message must be an instance of a Mail::Message. The actual message type does not have to match the folder type — the folder will try to resolve the differences with minimal loss of information. The coerced messages (how the were actually written) are returned as list.

The `%options` is a list of key/values, which are added to (overriding) the default options for the detected folder type.

example:

```
$mgr->appendMessage('=send', $message, folderdir => '/');
$mgr->appendMessage($received, $inbox->messages);

my @appended = $mgr->appendMessages($inbox->messages,
   folder => 'Drafts');
$_->label(seen => 1) foreach @appended;
```

`$obj->`**copyMessage**([$folder|$foldername], `$messages`, `%options`)

Copy a message from one folder into another folder. If the destination folder is already opened, **Mail::Box::copyTo()** is used. Otherwise, **Mail::Box::appendMessages()** is called.

You need to specify a folder's name or folder object as the first argument, or in the options list. The options are the same as those which can be specified when opening a folder.

```
-Option--Default
 share   <false>
```

share => BOOLEAN

Try to share the physical storage of the messages. The folder types may be different, but it all depends on the actual folder where the message is copied to. Silently ignored when not possible to share.

example:

```
my $drafts = $mgr->open(folder => 'Drafts');
my $outbox = $mgr->open(folder => 'Outbox');
$mgr->copyMessage($outbox, $drafts->message(0));

my @messages = $drafts->message(1,2);
$mgr->copyMessage('=Trash', @messages,
   folderdir => '/tmp', create => 1);

$mgr->copyMessage($drafts->message(1),
   folder => '=Drafts' folderdir => '/tmp',
   create => 1);
```

`$obj->`**moveMessage**([$folder|$foldername], `$messages`, `%options`)

Move a message from one folder to another.

BE WARNED that removals from a folder only take place when the folder is closed, so the message is only flagged to be deleted in the opened source folder.

BE WARNED that message labels may get lost when a message is moved from one folder type to an other. An attempt is made to translate labels, but there are many differences in interpretation by applications.

```
$mgr->moveMessage($received, $inbox->message(1))
```

is equivalent to

```
$mgr->copyMessage($received, $inbox->message(1), share => 1);
$inbox->message(1)->delete;

-Option--Default
 share   <true>
```

share => BOOLEAN

Manage message threads

`$obj`−>**threads**([$folders], `%options`)

Create a new object which keeps track of message threads. You can read about the possible options in Mail::Box::Thread::Manager. As`%options` specify one folder or an array of `$folders`. It is also permitted to specify folders before the options.

example:

```
 my $t1 = $mgr−>threads(folders => [ $inbox, $send ]);
 my $t2 = $mgr−>threads($inbox);
 my $t3 = $mgr−>threads($inbox, $send);
```

Internals

`$obj`−>**decodeFolderURL**($url)

Try to decompose a folder name which is specified as `$url` (see **open()**) into separate options. Special characters like @−sign, colon, and slash used in the user or password parts must be passed `$url`−encoded.

`$obj`−>**toBeThreaded**($folder, `$messages`)

Signal to the manager that all thread managers which are using the specified folder must be informed that new messages are coming in.

`$obj`−>**toBeUnthreaded**($folder, `$messages`)

Signal to the manager that all thread managers which are using the specified folder must be informed that new messages are or going out.

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::Manager−>**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, [$strings]])
Mail::Box::Manager−>**log**([$level, [$strings]])

Inherited, see ‘‘Error handling’’ in Mail::Reporter

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

DETAILS

On many places in the documentation you can read that it is useful to have a manager object. There are two of them: the Mail::Box::Manager, which maintains a set of open folders, and an extension of it: the Mail::Box::Manage::User.

Managing open folders

It is useful to start your program by creating a folder manager object, an Mail::Box::Manager. The object takes a few burdons from your neck:

- autodetect the type of folder which is used.

  This means that your application can be fully folder type independent.

- autoload the required modules

  There are so many modules involved in MailBox, that it is useful to have some lazy autoloading of code. The manager knows which modules belong to which type of folder.

- avoid double openings

  Your programming mistakes may cause the same folder to be opened twice. The result of that could be very destructive. Therefore, the manager keeps track on all open folders and avoids the same folder to be opened for the second time.

- close folders at clean-up

  When the program is ending, the manager will cleanly close all folders which are still open. This is required, because the autodestruct sequence of Perl works in an unpredicatable order.

- message thread detection

  MailBox can discover message threads which span multiple folders. Any set of open folders may be grouped in a tree of replies on replies on replies. When a folder is closed, it will automatically be removed from the threads, and a new folder can dynamically be added to the structure.

The manager is really simplifying things, and should therefore be the base of all programs. However, it is possible to write useful programs without it.

Managing a user

One step further is the Mail::Box::Manage::User object (since MailBox v2.057), which not only keeps track on open folders, but also collects information about not-open folders.

The user class is, as the name says, targeted on managing one single user. Where the Mail::Box::Manager will open any set of folder files, probably from multiple users, the user class want one root folder directory.

In many aspects, the user manager simplifies the task for user-based servers and other user-centric applications by setting smart defaults.

DIAGNOSTICS

Error: Folder `$name` is already open.
: You cannot ask the manager for a folder which is already open. In some older releases (before MailBox 2.049), this was permitted, but then behaviour changed, because many nasty side-effects are to be expected. For instance, an **Mail::Box::update()** on one folder handle would influence the second, probably unexpectedly.

Error: Folder `$name` is not a Mail::Box; cannot add a message.
: The folder where the message should be appended to is an object which is not a folder type which extends Mail::Box. Probably, it is not a folder at all.

Warning: Folder does not exist, failed opening `$type` folder `$name`.
: The folder does not exist and creating is not permitted (see open(create)) or did not succeed. When you do not have sufficient access rights to the folder (for instance wrong password for POP3), this warning will be produced as well.

  The manager tried to open a folder of the specified type. It may help to explicitly state the type of your folder with the `type` option. There will probably be another warning or error message which is related to this report and provides more details about its cause. You may also have a look at new(autodetect) and new(folder_types).

Warning: Folder type `$type` is unknown, using autodetect.
: The specified folder type (see open(type), possibly derived from the folder name when specified as url) is not known to the manager. This may mean that you forgot to require the Mail::Box extension which implements this folder type, but probably it is a typo. Usually, the manager is able to figure-out which type to use by itself.

Error: Illegal folder URL '$url'.
: The folder name was specified as URL, but not according to the syntax. See **decodeFolderURL()** for an description of the syntax.

Error: No foldername specified to open.
: `open()` needs a folder name as first argument (before the list of options), or with the `folder` option within the list. If no name was found, the MAIL environment variable is checked. When even that does not result in a usable folder, then this error is produced. The error may be caused by an accidental odd-length option list.

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.

Error: Use **appendMessage()** to add messages which are not in a folder.
: You do not need to copy this message into the folder, because you do not share the message between folders.

Warning: Use **moveMessage()** or **copyMessage()** to move between open folders.
: The message is already part of a folder, and now it should be appended to a different folder. You need to decide between copy or move, which both will clone the message (not the body, because they are immutable).

Warning: Will never create a folder `$name` without having write access.
: You have set open(create), but only want to read the folder. Create is only useful for folders which have write or append access modes (see Mail::Box::new(access)).

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