Module Version: 3.10

NAME

Net::SMTP - Simple Mail Transfer Protocol Client

SYNOPSIS

```
use Net::SMTP;

# Constructors
$smtp = Net::SMTP->new('mailhost');
$smtp = Net::SMTP->new('mailhost', Timeout => 60);
```

DESCRIPTION

This module implements a client interface to the SMTP and ESMTP protocol, enabling a perl5 application to talk to SMTP servers. This documentation assumes that you are familiar with the concepts of the SMTP protocol described in RFC2821. With IO::Socket::SSL installed it also provides support for implicit and explicit TLS encryption, i.e. SMTPS or SMTP+STARTTLS.

The Net::SMTP class is a subclass of Net::Cmd and (depending on avaibility) of IO::Socket::IP, IO::Socket::INET6 or IO::Socket::INET.

EXAMPLES

This example prints the mail domain name of the SMTP server known as mailhost:

```
#!/usr/local/bin/perl -w

use Net::SMTP;

$smtp = Net::SMTP->new('mailhost');
print $smtp->domain,"\n";
$smtp->quit;
```

This example sends a small message to the postmaster at the SMTP server known as mailhost:

```
#!/usr/local/bin/perl -w

use Net::SMTP;

my $smtp = Net::SMTP->new('mailhost');
```

```
$smtp->mail($ENV{USER});
if ($smtp->to('postmaster')) {
 $smtp->data();
 $smtp->datasend("To: postmaster\n");
 $smtp->datasend("\n");
 $smtp->datasend("A simple test message\n");
 $smtp->dataend();
} else {
 print "Error: ", $smtp->message();
}

$smtp->quit;
```

CONSTRUCTOR

new ([HOST] [, OPTIONS])

This is the constructor for a new Net::SMTP object. `HOST` is the name of the remote host to which an SMTP connection is required.

On failure `undef` will be returned and `$@` will contain the reason for the failure.

`HOST` is optional. If `HOST` is not given then it may instead be passed as the `Host` option described below. If neither is given then the `SMTP_Hosts` specified in `Net::Config` will be used.

`OPTIONS` are passed in a hash like fashion, using key and value pairs. Possible options are:

Hello - SMTP requires that you identify yourself. This option specifies a string to pass as your mail domain. If not given localhost.localdomain will be used.

SendHello - If false then the EHLO (or HELO) command that is normally sent when constructing the object will not be sent. In that case the command will have to be sent manually by calling `hello()` instead.

Host - SMTP host to connect to. It may be a single scalar (hostname[:port]), as defined for the `PeerAddr` option in IO::Socket::INET, or a reference to an array with hosts to try in turn. The "host" method will return the value which was used to connect to the host. Format - `PeerHost` from IO::Socket::INET new method.

Port - port to connect to. Default - 25 for plain SMTP and 465 for immediate SSL.

SSL - If the connection should be done from start with SSL, contrary to later upgrade with `starttls`. You can use SSL arguments as documented in IO::Socket::SSL, but it will usually use the right arguments already.

LocalAddr and **LocalPort** - These parameters are passed directly to IO::Socket to allow binding the socket to a specific local address and port.

Domain - This parameter is passed directly to IO::Socket and makes it possible to enforce IPv4 connections even if IO::Socket::IP is used as super class. Alternatively **Family** can be used.

Timeout - Maximum time, in seconds, to wait for a response from the SMTP server (default: 120)

ExactAddresses - If true the all ADDRESS arguments must be as defined by `addr-spec` in RFC2822. If not given, or false, then Net::SMTP will attempt to extract the address from the value passed.

Debug - Enable debugging information

Example:

```
    $smtp = Net::SMTP->new('mailhost',
                           Hello => 'my.mail.domain',
                           Timeout => 30,
                           Debug   => 1,
                          );

    # the same
    $smtp = Net::SMTP->new(
                           Host => 'mailhost',
                           Hello => 'my.mail.domain',
                           Timeout => 30,
                           Debug   => 1,
                          );

    # the same with direct SSL
    $smtp = Net::SMTP->new('mailhost',
                           Hello => 'my.mail.domain',
                           Timeout => 30,
                           Debug   => 1,
                           SSL     => 1,
                          );

    # Connect to the default server from Net::config
    $smtp = Net::SMTP->new(
                           Hello => 'my.mail.domain',
                           Timeout => 30,
                          );
```

METHODS

Unless otherwise stated all methods return either a *true* or *false* value, with *true* meaning that the operation was a success. When a method states that it returns a value, failure will be returned as *undef* or an empty list.

`Net::SMTP` inherits from `Net::Cmd` so methods defined in `Net::Cmd` may be used to send commands to the remote SMTP server in addition to the methods documented here.

banner ()

Returns the banner message which the server replied with when the initial connection was made.

domain ()

Returns the domain that the remote SMTP server identified itself as during connection.

hello (DOMAIN)

Tell the remote server the mail domain which you are in using the EHLO command (or HELO if EHLO fails). Since this method is invoked automatically when the Net::SMTP object is constructed the user should normally not have to call it manually.

host ()

Returns the value used by the constructor, and passed to IO::Socket::INET, to connect to the host.

etrn (DOMAIN)

Request a queue run for the DOMAIN given.

starttls (SSLARGS)

Upgrade existing plain connection to SSL. You can use SSL arguments as documented in IO::Socket::SSL, but it will usually use the right arguments already.

auth (USERNAME, PASSWORD)

auth (SASL)

Attempt SASL authentication. Requires Authen::SASL module. The first form constructs a new Authen::SASL object using the given username and password; the second form uses the given Authen::SASL object.

mail (ADDRESS [, OPTIONS])

send (ADDRESS)

send_or_mail (ADDRESS)

send_and_mail (ADDRESS)

Send the appropriate command to the server MAIL, SEND, SOML or SAML. `ADDRESS` is the address of the sender. This initiates the sending of a message. The method `recipient` should be called for each address that the message is to be sent to.

The `mail` method can some additional ESMTP OPTIONS which is passed in hash like fashion, using key and value pairs. Possible options are:

```
Size        => <bytes>
Return      => "FULL" | "HDRS"
Bits        => "7" | "8" | "binary"
Transaction => <ADDRESS>
Envelope    => <ENVID>     # xtext-encodes its argument
ENVID       => <ENVID>     # similar to Envelope, but expects argument encoded
XVERP       => 1
AUTH        => <submitter> # encoded address according to RFC 2554
```

The `Return` and `Envelope` parameters are used for DSN (Delivery Status Notification).

The submitter address in `AUTH` option is expected to be in a format as required by RFC 2554, in an RFC2821-quoted form and xtext-encoded, or <> .

reset ()

Reset the status of the server. This may be called after a message has been initiated, but before any data has been sent, to cancel the sending of the message.

recipient (ADDRESS [, ADDRESS, [...]] [, OPTIONS])

Notify the server that the current message should be sent to all of the addresses given. Each address is sent as a separate command to the server. Should the sending of any address result in a failure then the process is aborted and a *false* value is returned. It is up to the user to call `reset` if they so desire.

The `recipient` method can also pass additional case-sensitive OPTIONS as an anonymous hash using key and value pairs. Possible options are:

```
Notify  => ['NEVER'] or ['SUCCESS','FAILURE','DELAY']  (see below)
ORcpt   => <ORCPT>
SkipBad => 1        (to ignore bad addresses)
```

If `SkipBad` is true the `recipient` will not return an error when a bad address is encountered and it will return an array of addresses that did succeed.

```
$smtp->recipient($recipient1,$recipient2);  # Good
$smtp->recipient($recipient1,$recipient2, { SkipBad => 1 });  # Good
$smtp->recipient($recipient1,$recipient2, { Notify => ['FAILURE','DELAY'], SkipBad => 1 });  # Good
@goodrecips=$smtp->recipient(@recipients, { Notify => ['FAILURE'], SkipBad => 1 });  # Good
$smtp->recipient("$recipient,$recipient2"); # BAD
```

Notify is used to request Delivery Status Notifications (DSNs), but your SMTP/ESMTP service may not respect this request depending upon its version and your site's SMTP configuration.

Leaving out the Notify option usually defaults an SMTP service to its default behavior equivalent to ['FAILURE'] notifications only, but again this may be dependent upon your site's SMTP configuration.

The NEVER keyword must appear by itself if used within the Notify option and "requests that a DSN not be returned to the sender under any conditions."

```
{Notify => ['NEVER']}

$smtp->recipient(@recipients, { Notify => ['NEVER'], SkipBad => 1 });  # Good
```

You may use any combination of these three values 'SUCCESS','FAILURE','DELAY' in the anonymous array reference as defined by RFC3461 (see http://www.ietf.org/rfc/rfc3461.txt for more information. Note: quotations in this topic from same.).

A Notify parameter of 'SUCCESS' or 'FAILURE' "requests that a DSN be issued on successful delivery or delivery failure, respectively."

A Notify parameter of 'DELAY' "indicates the sender's willingness to receive delayed DSNs. Delayed DSNs may be issued if delivery of a message has been delayed for an unusual amount of time (as determined by the Message Transfer Agent (MTA) at which the message is delayed), but the final delivery status (whether successful or failure) cannot be determined. The absence of the DELAY keyword in a NOTIFY parameter requests that a "delayed" DSN NOT be issued under any conditions."

```
{Notify => ['SUCCESS','FAILURE','DELAY']}

$smtp->recipient(@recipients, { Notify => ['FAILURE','DELAY'], SkipBad => 1 });  # Good
```

ORcpt is also part of the SMTP DSN extension according to RFC3461. It is used to pass along the original recipient that the mail was first sent to. The machine that generates a DSN will use this address to inform the sender, because he can't know if recipients get rewritten by mail servers. It is expected to be in a format as required by RFC3461, xtext-encoded.

to (ADDRESS [, ADDRESS [...]])

cc (ADDRESS [, ADDRESS [...]])

bcc (ADDRESS [, ADDRESS [...]])

Synonyms for `recipient`.

data ([DATA])

Initiate the sending of the data from the current message.

DATA may be a reference to a list or a list and must be encoded by the caller to octets of whatever encoding is required, e.g. by using the Encode module's encode() function.

If specified the contents of DATA and a termination string ".\r\n" is sent to the server. The result will be true if the data was accepted.

If DATA is not specified then the result will indicate that the server wishes the data to be sent. The data must then be sent using the datasend and dataend methods described in Net::Cmd.

bdat (DATA)

bdatlast (DATA)

Use the alternate DATA command "BDAT" of the data chunking service extension defined in RFC1830 for efficiently sending large MIME messages.

expand (ADDRESS)

Request the server to expand the given address Returns an array which contains the text read from the server.

verify (ADDRESS)

Verify that ADDRESS is a legitimate mailing address.

Most sites usually disable this feature in their SMTP service configuration. Use "Debug => 1" option under new() to see if disabled.

help ([$subject])

Request help text from the server. Returns the text or undef upon failure

quit ()

Send the QUIT command to the remote SMTP server and close the socket connection.

can_inet6 ()

Returns whether we can use IPv6.

can_ssl ()

Returns whether we can use SSL.

ADDRESSES

Net::SMTP attempts to DWIM with addresses that are passed. For example an application might extract The From: line from an email and pass that to mail(). While this may work, it is not recommended. The application should really use a module like Mail::Address to extract the mail address and pass that.

If ExactAddresses is passed to the constructor, then addresses should be a valid rfc2821-quoted address, although Net::SMTP will accept the address surrounded by angle brackets.

```
funny user@domain      WRONG
"funny user"@domain    RIGHT, recommended
<"funny user"@domain>  OK
```

SEE ALSO

Net::Cmd, IO::Socket::SSL

AUTHOR

Graham Barr <*gbarr@pobox.com*>

Steve Hay <*shay@cpan.org*> is now maintaining libnet as of version 1.22_02

COPYRIGHT