NAME

Email::Valid – Check validity of Internet email addresses

VERSION

version 1.202

SYNOPSIS

```
use Email::Valid;
my $address = Email::Valid->address('maurice@hevanet.com');
print ($address ? 'yes' : 'no');
```

DESCRIPTION

This module determines whether an email address is well-formed, and optionally, whether a mail host exists for the domain.

Please note that there is no way to determine whether an address is deliverable without attempting delivery (for details, see perlfaq 9 <http://perldoc.perl.org/perlfaq9.html#How-do-I-check-a-valid-mail-address>).

PREREQUISITES

This module requires perl 5.004 or later and the Mail::Address module. Either the Net::DNS module or the nslookup utility is required for DNS checks. The Net::Domain::TLD module is required to check the validity of top level domains.

METHODS

Every method which accepts an <ADDRESS> parameter may be passed either a string or an instance of the Mail::Address class. All errors raise an exception.

new ([PARAMS])

This method is used to construct an Email::Valid object. It accepts an optional list of named parameters to control the behavior of the object at instantiation.

The following named parameters are allowed. See the individual methods below for details.

```
-mxcheck
-tldcheck
-fudge
-fqdn
-allow_ip
-local_rules
```

mx (<ADDRESS>|<DOMAIN>)

This method accepts an email address or domain name and determines whether a DNS record (A or MX) exists for it.

The method returns true if a record is found and undef if not.

Either the Net::DNS module or the nslookup utility is required for DNS checks. Using Net::DNS is the preferred method since error handling is improved. If Net::DNS is available, you can modify the behavior of the resolver (e.g. change the default tcp_timeout value) by manipulating the global Net::DNS::Resolver instance stored in `$Email::Valid::Resolver`.

rfc822 (<ADDRESS>)

This method determines whether an address conforms to the RFC822 specification (except for nested comments). It returns true if it conforms and undef if not.

fudge (<TRUE>|<FALSE>)

Specifies whether calls to *address()* should attempt to correct common addressing errors. Currently, this results in the removal of spaces in AOL addresses, and the conversion of commas to periods in Compuserve addresses. The default is false.

allow_ip (<TRUE>|<FALSE>)

Specifies whether a ''domain literal'' is acceptable as the domain part. That means addresses like: `rjbs@[1.2.3.4]`

The checking for the domain literal is stricter than the RFC and looser than checking for a valid IP address, *but this is subject to change*.

The default is true.

fqdn (<TRUE>|<FALSE>)
Specifies whether addresses passed to *address()* must contain a fully qualified domain name (FQDN). The default is true.

Please note! FQDN checks only occur for non-domain-literals. In other words, if you have set `allow_ip` and the address ends in a bracketed IP address, the FQDN check will not occur.

tld (<ADDRESS>)
This method determines whether the domain part of an address is in a recognized top-level domain.

Please note! TLD checks only occur for non-domain-literals. In other words, if you have set `allow_ip` and the address ends in a bracketed IP address, the TLD check will not occur.

local_rules (<TRUE>|<FALSE>)
Specifies whether addresses passed to *address()* should be tested for domain specific restrictions. Currently, this is limited to certain AOL restrictions that I'm aware of. The default is false.

mxcheck (<TRUE>|<FALSE>)
Specifies whether addresses passed to *address()* should be checked for a valid DNS entry. The default is false.

tldcheck (<TRUE>|<FALSE>)
Specifies whether addresses passed to *address()* should be checked for a valid top level domains. The default is false.

address (<ADDRESS>)
This is the primary method which determines whether an email address is valid. Its behavior is modified by the values of *mxcheck()*, *tldcheck()*, *local_rules()*, *fqdn()*, and *fudge()*. If the address passes all checks, the (possibly modified) address is returned as a string. Otherwise, undef is returned. In a list context, the method also returns an instance of the Mail::Address class representing the email address.

details ()
If the last call to *address()* returned undef, you can call this method to determine why it failed. Possible values are:

```
rfc822
localpart
local_rules
fqdn
mxcheck
tldcheck
```

If the class is not instantiated, you can get the same information from the global `$Email::Valid::Details`.

EXAMPLES

Let's see if the address 'maurice@hevanet.com' conforms to the RFC822 specification:

```
print (Email::Valid->address('maurice@hevanet.com') ? 'yes' : 'no');
```

Additionally, let's make sure there's a mail host for it:

```
print (Email::Valid->address( -address => 'maurice@hevanet.com',
                              -mxcheck => 1 ) ? 'yes' : 'no');
```

Let's see an example of how the address may be modified:

```
$addr = Email::Valid->address('Alfred Neuman <Neuman @ foo.bar>');
print "$addr\n"; # prints Neuman@foo.bar
```

Now let's add the check for top level domains:

```
$addr = Email::Valid->address( -address => 'Neuman@foo.bar',
                               -tldcheck => 1 );
print "$addr\n"; # doesn't print anything
```

Need to determine why an address failed?

```
unless(Email::Valid->address('maurice@hevanet')) {
  print "address failed $Email::Valid::Details check.\n";
}
```

If an error is encountered, an exception is raised. This is really only possible when performing DNS queries. Trap any exceptions by wrapping the call in an eval block:

```
eval {
  $addr = Email::Valid->address( -address => 'maurice@hevanet.com',
                                 -mxcheck => 1 );
};
warn "an error was encountered: $@" if $@;
```

CREDITS

Significant portions of this module are based on the ckaddr program written by Tom Christiansen and the RFC822 address pattern developed by Jeffrey Friedl. Neither were involved in the construction of this module; all errors are mine.

Thanks very much to the following people for their suggestions and bug fixes:

```
Otis Gospodnetic <otis@DOMINIS.com>
Kim Ryan <kimaryan@ozemail.com.au>
Pete Ehlke <pde@listserv.music.sony.com>
Lupe Christoph
David Birnbaum
Achim
Elizabeth Mattijsen (liz@dijkmat.nl)
```

SEE ALSO

Mail::Address, Net::DNS, Net::Domain::TLD, perlfaq9 <https://metacpan.org/pod/distribution/perlfaq/lib/perlfaq9.pod>

RFC822 <https://www.ietf.org/rfc/rfc0822.txt> – standard for the format of ARPA internet text messages. Superseded by RFC2822 <https://www.ietf.org/rfc/rfc2822.txt>.

AUTHOR

Maurice Aubrey <maurice@hevanet.com>

CONTRIBUTORS

- Alexandr Ciornii <alexchorny@gmail.com>
- Karel Miko <karel.miko@gmail.com>
- McA <McA@github.com>
- Michael Schout <mschout@gkg.net>
- Mohammad S Anwar <mohammad.anwar@yahoo.com>
- Neil Bowers <neil@bowers.com>
- Ricardo SIGNES <rjbs@cpan.org>
- Steve Bertrand <steveb@cpan.org>

- Svetlana <svetlana.wiczer@gmail.com>
- Troy Morehouse <troymore@nbnet.nb.ca>

COPYRIGHT AND LICENSE