

Email Address Validation Class Library

The email address validation library is available for both the Perl and PHP programming (scripting) languages. This class library provides a powerful tool that, when used responsibly, may be used to validate and verify Internet email addresses. Use of the library requires a working knowledge of the either the Perl or PHP programming language and the use class objects. It is assumed, but not required, the user will have a general knowledge of Internet standards and specifications relating to email addresses, DNS, and SMTP.

The library is composed of three primary class objects, the email address validation control object, the DNS resolver object, and the SMTP verification object. Additionally, the library contains supplemental helper objects to aid the primary objects during address validation. The email address validation object, `email_class`, is the overall library control object and is generally the only object your application need directly interface with in order to use the validation library.

The library includes two sample applications (scripts). A fully operational email address validation utility, `emailtest.php` (for PHP) or `emailtest.cgi` (for Perl). This script demonstrates the basic overall use of the library. Before the script can be used you will need to specify the name of your Name Server, your domain, and a valid email address from your domain. You will need to view the source code file for specific instructions regarding these settings. The second application is a fully functional test program `dnstest.php` (for PHP) or `dnstest.cgi` (for Perl). This utility demonstrates the use of the DNS class object.

Quentin Sager Consulting
20429 Ring Neck Road
Altoona, FL 32702
USA

www.quentinsagerconsulting.com
support@quentinsagerconsulting.com

Files Included With the Library

The library includes and uses the following files to perform email address validation. The actual files included with the library depend on the particular version however all versions use the same logical object definitions unless otherwise noted. By default all files should be placed in the same working directory or folder.

PHP edition source files.

The following source files are included with the PHP edition library only.

File	Description
email.class.php	The email validation object library “front end”. This is the only library source file your script must explicitly include through the PHP method require or require_once . This module will transparently include the additional library objects as required.
email.class.inc	Constant definitions used for the email validation class.
smtp.class.php	SMTP class used by the email validation class to communicate with remote mail servers.
dns.class.php	DNS query class used by the email validation class. Includes the primary DNS resolver class, constant definitions, and associated objects.
rfc822char.class.php	Character set definitions used by the email class parsing methods.
freemail.txt	External data file used to identify well known free email domains.
emailtest.php	Sample program/script to demonstrate the email validation class.
dnstest.php	Sample program/script to demonstrate the DNS lookup class.

Perl edition source files.

The following source files are included with the Perl edition library only.

File	Description
email_class.pm	The email validation object library “front end”. This is the only library source file your script must explicitly include through the Perl method use or require . This module will transparently include the additional library objects as required.
smtp_class.pm	SMTP class used by the email validation class to communicate with remote mail servers.
dns_resolver.pm	DNS query class used by the email validation class. Includes the primary DNS resolver class and related constant definitions.
dns_record.pm	DNS resolver helper object for common DNS record interface.
socket_class.pm	Internet socket wrapper class used by the SMTP and DNS objects.
rfc822char_class.pm	Character set definitions used by the email class parsing methods.
freemail.txt	External data file used to identify well known free email domains.
emailtest.cgi	Sample program/script to demonstrate the email validation class.
dnstest.cgi	Sample program/script to demonstrate the DNS lookup class.

The library has no particular installation requirements. To install, simply copy the the source code files to your working server or development directory. Directory requirements vary for web server install however Perl scripts are generally copied to the /cgi-bin or /scripts directory. PHP scripts generally have no server specific requirements and can usually be copied to any directory.

Email Address Validation Class Definition

The Email Address validation object contains numerous methods to perform its validation. Though there are syntactic differences, the Perl and PHP versions share a common definition. Several methods are used internally by the object and should not be called directly. These internal methods are identified in the method descriptions.

```
class email_class
{
function email_class ();
function GetEmailLevel();
function GetEmailError();
function GetEmailErrorDescription();
function GetEmailDomain();
function GetEmailTLD();
function GetEmailDomainAddress();
function GetEmailLocalPart();
function GetEmailDisplayName();
function GetEmailAdress();
function GetMxNames();
function GetMxInfo();
function GetFromDomain();
function SetFromDomain($_lpFromDomain);
function GetFromEmail();
function SetFromEmail($_lpFromMbx);
function GetNameServer();
function SetNameServer($_lpNameServer);
function GetSmtpTrace();
function Accept($_bAccept,$_uiProperties);
function AcceptLiteral($_bAccept);
function AcceptPrivate($_bAccept);
function AcceptFree($_bAccept);
function AcceptRouting($_bAccept);
function RequireDnsMx($_bRequire);
function RequireTld($_bRequire);
function TestCatchall($_bTest);
function isCCTld();
function isGeneralTld();
function isDomainLiteral();
function isPrivateNetwork();
function isEmbeddedRouting();
function isFreeEmailDomain();
function isdomainname($parameter);
function isipaddress($parameter);
function collect_atext(&$pointer,$bRfc952=false);
function collect_digits(&$pointer);
function collect_qtext(&$pointer);
function collect_comment();
function collect_route();
function collect_domainliteral();
function collect_domain();
function collect_local_part($strict);
function CheckSyntax($s);
function CheckDns();
function CheckSmtp();
function Validate($address,$level = LEVEL_SYNTAX);
}
```

Email Validation Class Error and Status Codes

The object's error and status return codes provide considerable granularity to determine whether an email address is or could be valid. Many of these codes clearly identify error conditions while other codes provide a status condition.

Mnemonic	Value	Description
ERR_NOT_SET	-1	object has been instantiated but no address validated
ERR_SUCCESS	0	operation was successful
ERR_NO_ADDRESS	1	the address passed to validation routine was empty
ERR_UNBALANCED_COMMENT	2	unbalanced comment, missing closing parenthesis
ERR_UNBALANCED_QUOTE	3	unbalanced quoted text, missing closing quote
ERR_UNBALANCED_DOMAIN_LITERAL	4	unbalanced domain literal, missing closing bracket
ERR_UNBALANCED_ROUTE_SPEC	5	unbalanced route spec, missing closing angle bracket
ERR_UNBALANCED_MAILROUTE_SPEC	6	unbalanced mail routing spec, missing closing delimiter
ERR_MAILROUTE_SPECIFIED	7	address spec contains routing information
ERR_DOMAIN_LITERAL_SPECIFIED	8	address contains a domain literal, and domain literals are not accepted
ERR_FREE_MAIL_DOMAIN	9	address specifies a free email domain address
ERR_GROUP_SPECIFIED	10	address group lists are not supported
ERR_NO_DOMAIN	11	email address is missing domain
ERR_NO_LOCAL_PART	12	email address is missing local part (recipient)
ERR_ADDRESS_TOO_LONG	13	total length of email address is greater than 256 characters
ERR_LOCAL_PART_TOO_LONG	14	length of email address local part is greater than 64 characters
ERR_DOMAIN_TOO_LONG	15	length of domain name is greater than 255 characters
ERR_LABEL_TOO_LONG	16	length of domain name component is greater than 63 characters
ERR_LABEL_TOO_SHORT	17	length of domain name component is less than 2 characters
ERR_INVALID_QUOTED_TEXT	18	misplaced quoted string, not allowed in domain name
ERR_INVALID_QUOTED_LITERAL	19	misplaced quoted literal, not allowed in domain name
ERR_TOO_MANY_OCTETS	20	domain literal contains too many octets
ERR_TOO_FEW_OCTETS	21	domain literal does not contain enough octets
ERR_INVALID_IP_ADDRESS	22	domain literal specifies an invalid IP address
ERR_MULTICAST_IP	23	domain literal specifies a multicast IP address
ERR_BROADCAST_IP	24	domain literal specifies a network broadcast IP address
ERR_RESERVED_IP	25	domain literal specifies a class E reserved IP address
ERR_NETWORK_IP	26	domain literal specifies network IP address only
ERR_HOST_IP	27	domain literal IP address specifies this host on this network only
ERR_LOCAL_HOST	28	domain literal specifies reserved local host IP address
ERR_MISSING_OCTET	29	domain literal expected octet before delimiter
ERR_MISSING_OCTET_DELIMITER	30	domain literal expected delimiter before octet
ERR_MISSING_ATOM	31	expected atom before delimiter
ERR_MISSING_ATOM_DELIMITER	32	expected delimiter before atom
ERR_CANT_MIX_ATOM_LITERAL	33	both atom and literal used in domain spec
ERR_DOMAIN_PRIVATE	34	domain specifies a private unreachable domain
ERR_TLD_REQUIRED	35	domain must be from a recognized TLD
ERR_DOMAIN_TLDOONLY	36	only a top level domain has been specified
ERR_BAD_DOMAIN_LABEL	37	invalid first or last character in a domain name label
ERR_INVALID_CHARACTER	38	character is invalid for context
ERR_SYNTAX	39	syntax is incorrect
ERR_DNS_FAILURE	40	name server was unable to process this query due to a problem with the name server
ERR_DNS_REFUSED	41	name server refused query operation
ERR_DNS_UNSUPPORTED	42	name server does not support the requested query
ERR_DNS_NO_CONNECT	43	error connecting to DNS name server
ERR_DNS_NOT_DOMAIN	44	the queried domain does not exist
ERR_DNS_NO_MX_RECORD	45	the queried domain does not have an MX record
ERR_DNS_BOGUS_MX_RECORD	46	the queried domain MX records do not have associated A records

Quentin Sager Consulting

Software and Data Solutions™

ERR_DNS_UNREACHABLE	47	the queried domain does not have any MX or A records
ERR_DNS_NO_RECORDS	48	no DNS records could be found for the domain
ERR_DNS_COULD_NOT_RESOLVE	49	could not resolve domain literal
ERR_SMTP_ACCEPTED	50	SMTP server accepted the mailbox name
ERR_SMTP_AMBIGUOUS	51	SMTP server rejected the mailbox name, ambiguous
ERR_SMTP_MBX_INVALID	52	SMTP server rejected the mailbox name, invalid or unknown user
ERR_SMTP_MBX_FULL	53	SMTP server rejected, the mailbox name is valid but full
ERR_SMTP_MBX_BUSY	54	SMTP server rejected, the mailbox name is valid but busy
ERR_SMTP_TRY_MBX	55	user not local, server will not forward
ERR_SMTP_FORWARD_MBX	56	user not local, server will forward
ERR_SMTP_MBX_SYNTAX	57	SMTP server rejected, the mailbox name syntax is wrong
ERR_SMTP_NO_VERIFY	58	could not verify user
ERR_SMTP_NO_SERVICE	59	no SMTP service available
ERR_SMTP_SERVER_ERR	60	SMTP server could not process due to server error
ERR_SMTP_SERVER_FULL	61	SMTP server could not process out of storage
ERR_SMTP_SERVER_SHUTDOWN	62	SMTP server shutdown
ERR_SMTP_SERVER_BLOCK	63	SMTP server is blocking or not accepting mail from us
ERR_SMTP_CATCHALL	64	SMTP server accepts catch all mailbox names
ERR_SMTP_NO_CONNECT	65	could not connect to SMTP server
ERR_SMTP_NO_SERVERS	66	no SMTP mail servers

Email Address Validation Class Method Descriptions

The primary interface to the email address validation library is the **email_class** object. With the single exception of the class constructor name both the Perl and PHP versions use identical method names and parameters. Access to the class methods through either language is syntactically similar through the use of a class object reference variable.

Syntax Note:

Several methods require parameters passed by reference (i.e. pointers to the data). In Perl these parameters are passed using this syntax: **\\$pointer**, for PHP we use this syntax **&\$pointer**.

PHP constructor

:: email_class(\$lpNameServer,\$lpFromDomain,\$lpFromMbx,\$lpCatchall)

Perl constructor

:: new(\$lpNameServer,\$lpFromDomain,\$lpFromMbx,\$lpCatchall)

Class constructor. Initializes the class on object instantiation and optionally sets working values from one or more passed parameters.

Parameters:

\$ lpNameServer:

Optional – defaults to “A.ROOT-SERVERS.NET”. Specifies the DNS server name to use for realtime lookup. See :: SetNameServer method.

\$ lpFromDomain:

Optional – defaults to “localhost.localdomain “. Specifies the originating domain (your domain name) used during SMTP queries. See :: SetFromDomain method.

\$ lpFromMbx:

Optional – defaults to “postmaster@localhost.localdomain”. Specifies the originating mailbox name used during SMTP queries. See :: SetFromEmail method.

\$ lpCatchall:

Optional – defaults to “catchall”. Specifies the recipient name used during SMTP catch all testing. See :: SetCatchallRecipient method.

Returns:

none

:: SetDataPath (\$lpDataPath)

Set the directory path used by the validation object to locate the **freemail.txt** data file used during the validation process. The default object value is “./” meaning the current directory. The default value may work for your environment but you will generally want to explicitly set it.

For example, if you are using the Perl version in a CGI environment and you placed the library in the /cgi-bin directory the default value should generally work. If however you have placed the library in a directory name per off your /cgi-bin directory (i.e. **/cgi-bin/perl**) you will want to set the data path value to **perl/**.

Parameters:

\$lpDataPath:

Absolute or relative path to the directory containing the **freemail.txt** data file. **The path name must include a trailing directory delimiter.**

Returns:

none

::SetNameServer(\$_lpNameServer)

Set the name of the name server to use for initial real-time DNS queries. This value should ideally be set to your domain's name server. If this value is not explicitly set it will default to the first name server of the root Internet name servers.

Parameters:

\$_lpNameServer

Name server host name to connect with to perform DNS queries. This name will generally be the name server that services your domain.

Returns:

none

::GetNameServer()

Return the name of the current name server used for initial DNS queries, see SetNameServer method.

Parameters:

none

Returns:

Current name server being used for top level DNS queries.

::SetFromDomain(\$_lpFromDomain)

Set the domain name to be used for SMTP identification. This value should be set to your domain name (not the domain name of the email address being validated). It is used during SMTP validation as the parameter to the SMTP HELO command which identifies who you are to the remote mail server.

Parameters:

\$_lpFromDomain:

Generally this will be something like *yourdomain.com*. Current practice of many SMTP mail servers is to perform a DNS query to retrieve the DNS PTR record to verify the sender's IP address is valid for the specified domain name.

Returns:

none

::GetFromDomain()

Return the current domain name used for SMTP identification, see SetFromDomain method.

Parameters:

none

Returns:

Current domain name used as the HELO command parameter when performing SMTP validation.

::SetFromEmail(\$_lpFromMbx)

Set the sender email address to be used during SMTP validation. This value should be set to your email address (not the email address being validated). It is used during SMTP validation as the parameter to the SMTP MAIL FROM command which identifies who you are to the remote mail server. The email address being validated is always used as the parameter to the SMTP RCPT TO command.

Parameters:

\$_lpFromMbx:

Generally this will be something like *webmaster@yourdomain.com*. This should be a valid email address and should ideally be from your domain.

Returns:

none

::GetFromEmail()

Return the current email address used to identify the sender during SMTP identification, see SetFromEmail method.

Parameters:

none

Returns:

Current email address used as the MAIL FROM command parameter when performing SMTP validation.

:: SetCatchallRecipient (\$lpCatchall)

Set the recipient (mailbox) name to be used to perform SMTP catch all testing. This value is used to determine if the remote mail server will accept all mail sent to the domain. This value is appended with the domain of the current email address being validated.

Parameters:

\$lpCatchall:

Any unique name or value (potentially bogus) to determine if the remote server accepts all mail. This value should not include domain information.

Returns:

none

:: GetCatchallRecipient ()

Return the current recipient name used for catch all validation.

Parameters:

none

Returns:

The catch all recipient name.

::GetEmailDomain()

Return the domain name portion of the last email address validated. This value is set during the syntax check step of the address validation.

Parameters:

none

Returns:

Email address domain name.

::GetEmailTLD()

Post validation method to retrieve the TLD (top level domain) portion of the last email address validated. This value is set during the syntax check step of the address validation.

Parameters:

none

Returns:

Email address top level domain.

::GetEmailDomainAddress()

Return the IP address of the email address domain. This value is set during the DNS check step of the address validation and is taken directly from the domain's DNS A (address) record.

Parameters:

none

Returns:

Email address domain IP address.

::GetEmailLocalPart()

Return the local part or actual recipient mailbox name portion of the last email address validated. This value is set during the syntax check step of the address validation.

Parameters:

none

Returns:

Email address local part (mailbox recipient).

::GetEmailDisplayName()

Return the user display name portion if any for the current email address. This value is set during the syntax check step of the address validation.

Parameters:

none

Returns:

Email address user display name if any.

::GetEmailRoute ()

Return the embedded routing portion if any for the current email address. This value is set during the syntax check step of the address validation.

Parameters:

none

Returns:

Embedded routing information if any.

::GetEmailLevel()

Post validation method to retrieve the last validation level completed. The last level completed is also returned directly from the ::Validate method. See ::Validate method for details

Parameters:

none

Returns:

Last validation level completed.

::GetEmailError()

Post validation method to retrieve the last validation detailed error or status code. There are a total of 68 possible values for this code. These values in combination with the validation level completed code are used to determine if the email is valid or invalid.

Parameters:

none

Returns:

Error code result of last validation.

::GetEmailErrorDescription()

Post validation method to retrieve error or status descriptive text for the last validation.

Parameters:

none

Returns:

Descriptive text of last validation result.

::GetEmailAddress()

Return the fully constructed email address (i.e. local part and domain name) of the last address validated. This is a final, *clean* version of the address which excludes any embedded routing information, comments, or user display name portions that may have been included in the original input.

Parameters:

none

Returns:

A clean version of the current validated email address. The returned address contains the local part and domain name only without comments (if any) and user display name (if any).

::GetMxNames()

Return the array (or reference to) the list of mail server names for the current email address domain. These values are set from the domain's DNS MX records during the DNS check step of the address validation.

Parameters:

none

Returns:

Array containing the DNS resolved mail server names for the current email address or false if list is empty.

::GetMxInfo()

Return the array (or reference to) the list of mail server names, server preference, and their IP addresses for the current email address domain. These values are set from the domain's DNS MX records during the DNS check step of the address validation. Each entry is stored as a single string with each *field* delimited by a single ~ character.

Example Entries:

```
mailserver1.testdomain.com~0~192.168.1.1  
mailserver2.testdomain.com~5~192.168.1.2
```

Parameters:

none

Returns:

Array containing the DNS resolved mail server names, preferences, and IP addresses for the current email address or false if list is empty.

::GetSmtTrace()

Return conversation log of the SMTP session for the last email address validated. This value is set during the SMTP check step of the address validation.

Parameters:

none

Relies upon execution of an SMTP validation.

Returns:

String containing the full SMTP validation conversation with the remote mail server.

::Accept(\$_bAccept,\$_uiProperty)

Set or clear additional validation properties. This method can be called directly or through the property specific wrapper methods. When called directly multiple properties can be set or cleared with a single call by or'ing the properties.

Property Mnemonic	Value	Description
B_ACCEPT_GROUP	0x0001	accept group names when set
B_ACCEPT_LITERAL	0x0002	accept domain literals when set
B_ACCEPT_PRIVATE	0x0004	accept private domains when set
B_ACCEPT_FREE	0x0008	accept free email domains when set
B_ACCEPT_ROUTING	0x0010	accept routing info in address spec

B_REQUIRE_DNS_MX	0x0020	require DNS MX record even if A record present
B_REQUIRE_TLD	0x0040	require a recognized TLD
B_TEST_CATCHALL	0x0080	perform a catchall SMTP test

Parameters:

\$_bAccept

True, set the property, false to clear the property.

\$_uiProperty

The property or properties to set or clear.

Returns:

none

::AcceptLiteral(\$_bAccept)

Specifies whether to accept a domain literal for the email address domain. Literal domains such as *myaccount@[192.168.1.1]* are perfectly valid when specifying an email address however their use is generally discourage.

Parameters:

\$_bAccept

True, accept email addresses when the domain is specified using a domain literal instead of a named domain.

Returns:

none

::AcceptPrivate(\$_bAccept)

Specifies whether to accept private domain names (or domain literals using private IP addresses) in the email address. For example: *myaccount@mydomain.anywhere* and *myaccount@[192.168.1.1]* are syntactically correct however both email addresses use private domains that are unreachable when using normal Internet connections.

Parameters:

\$_bAccept

True, accept email addresses even if they do not specify a recognized top level domain.

Returns:

none

::AcceptFree(\$_bAccept)

Specifies whether to accept email addresses from well know free email address domains such as *yahoo.com* and *hotmail.com* even if the address passes syntax validation. Free email domains are identified by the object through the use of an external data file (supplied).

Parameters:

\$_bAccept

True, accept email addresses from free email domains.

Returns:

none

::AcceptRouting(\$_bAccept)

Specifies whether to accept email addresses that contain routing information in the address specification. The use of routing information within an email address specification has become virtually obsolete and is generally ignored however email processing software must still be able to parse it. If routing information is present it is tested for syntactic correctness and then removed from the final email address. For example, the email address `<@abc.com,def.com,ghi.com:johnqpublic@mydomain.com>` would reduce to `<johnqpublic@mydomain.com>`.

Parameters:

\$_bAccept

True, email address should be accepted even if it contains embedded routing information.

Returns:

none

::RequireDnsMx(\$_bRequire)

Specifies whether to accept an email address from a domain that has a valid DNS A record but does not have any DNS MX records. Most domains will contain MX records in their DNS entries however there is not particular requirement regarding this issue. RFC 2821 states if a domain has no DNS MX records but does have a DNS A (address) record then mail delivery should be attempted using the A record address.

Parameters:

\$_bRequire

True, the email domain must have valid DNS MX records to be considered valid.

Returns:

none

::RequireTld(\$_bRequire)

Specifies whether the address should be from a public top level (country coded or general) domain.

Parameters:

\$_bRequire

True, the domain name should be from a recognized top level domain.

Returns:

none

::TestCatchall(\$_bTest)

Specifies whether to perform a *catchall* address check before attempting SMTP validation of the email address. Setting this option causes the validation to begin with a known-bad test address. If the server accepts this address, the validation will halt and sets validation results to `ERR_SMTP_CATCHALL`. See `:: SetCatchallRecipient` method.

Parameters:

\$_bTest

True, a catchall name test should perform during SMTP validation.

Returns:

none

::isCCTld()

Post validation method to determine if the current email address is from a public country code top level domain. Country coded top level domains are currently two character codes generally based on their ISO 3166 two character country code.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isGeneralTld()

Post validation method to determine if the current email address is from a public general top level domain. Current top level domains include: aero, biz, com, coop, edu, gov, info, int, mil, museum, name, net, org, pro, and arpa.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isDomainLiteral()

Post validation method to determine if the current address specified its domain by name or IP literal address.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isPrivateNetwork()

Post validation method to determine if the current address specified a private domain.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isEmbeddedRouting()

Post validation method to determine if the current address contained routing information in its address specification.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isFreeEmailDomain()

Post validation method to determine if the current address specified a well know free email domain address.

Parameters:

none, relies upon object internal state

Returns:

true or false

::isdomainname(\$domain)

Internal method called by *::collect_domain* method to determine if the email address could be a valid domain name. It will check that the domain name and its components meet length and label requirements, determine whether it specifies a public or private domain and whether it specifies a well known free email address domain.

Parameters:

\$domain

domain name to validate

Returns:

object error code

::isipaddress(\$domain)

Internal method called by *::collect_domainliteral* method to determine if a domain literal could be a valid IPv4 address.

Parameters:

\$domain

domain literal to validate

Returns:

object error code

::collect_atext(&\$pointer,\$bRfc952=false)

Internal method used to parse and extract an atom (name or label) from the current address or address component.

Parameters:

\$pointer

string reference to return the atom to

\$bRfc952

Boolean, when true the atom being collected should be tested using domain name label requirements. When false the atom should be tested with relaxed requirements (i.e. local part atom).

Returns:

object error code

::collect_digits(&\$pointer)

Internal method used to parse and extract a domain literal octet from the current address.

Parameters:

\$pointer
string reference to return the octet to

Returns:

object error code

::collect_qtext(&\$pointer)

Internal method to parse and extract quoted text from the current address.

Parameters:

\$pointer
string reference to return the quoted text to

Returns:

object error code

::collect_comment()

Internal method used to parse a comment from the current address. If present, comments are removed from the email address.

Parameters:

none, relies upon object internal state

Returns:

object error code

Example Address:

- with comments - "John Q. Public" (test account)<(local part)jqpublic@anydomain (my domain) . com
- comments removed - "John Q. Public" <jqpublic@anydomain.com

::collect_route()

Internal method called by *::collect_local_part* method to parse routing information from the current address specification. Routing information is removed from the final address. Though it is not used in the final address, if present, routing information is syntactically validated.

Parameters:

none, relies upon object internal state

Returns:

object error code

Example Address:

- "John Q. Public" <@xyz.com,@abc.com,@efg.com:jqpublic@anydomain.com>

::collect_domainliteral()

Internal method called by *::collect_domain* method to parse, extract, and validate a domain literal from the current address.

Parameters:

none, relies upon object internal state

Returns:

object error code

::collect_domain()

Internal method called by `::CheckSyntax` method to parse, identify, extract, syntactically validate the domain portion of the current email address. It can distinguish between named domains and domain literals.

Parameters:

none, relies upon object internal state

Returns:

object error code

Example Addresses:

- named domain – `jqpublic@anydomain.com`
- domain literal – `jqpublic@[192.168.1.1]`

::collect_local_part(\$strict)

Internal method called by `::CheckSyntax` method to parse and extract the local part of the current email address. This method may be called multiple times internally depending on the original entry. Its primary function is to extract the recipient mailbox name from the email address.

Examples Addresses:

- `jqpublic@anydomain.com` the local part is `jqpublic`.
- “John Q. Public” <`jqpublic@anydomain.com`> the local part is `jqpublic` the display name is `John Q. Public`

Parameters:

\$strict

Boolean to determine how to initially interpret the local part of the address. When false indicates the local part could be interpreted as a prefixed user display name or actual recipient mailbox name. When true, specifies the local part should be interpreted as the actual recipient mailbox.

Returns:

object error code

::CheckSyntax(\$address)

Performs full syntax and basic domain evaluation of the email address. Evaluation of the address is accomplished by parsing the address left to right to identify, separate, and validate its components. This step is a true parsing operation of the address applying current Internet standards.

Related Validation Properties	Effect
B_ACCEPT_LITERAL	If set, specifies the email address should be rejected if the domain is specified using domain literal syntax even if the address passes basic validation.
B_ACCEPT_PRIVATE	Private domains are basically unreachable through normal Internet communications (there are exceptions). If set, this property specifies the email address should be rejected even if it passes basic

	validation.
B_ACCEPT_FREE	If set, specifies the email address should be rejected even if it passes basic validation because it specifies a well known free email address domain.
B_ACCEPT_ROUTING	If set, specifies the email address should not be rejected if it contains embedded routing information. Though routing information if present is actually discarded it must still be syntactically correct.
B_REQUIRE_TLD	If set, requires a named domain to include a recognized TLD

Parameter:

\$address

email address to validate

Returns:

object error code

::CheckDns()

Performs a DNS query of the current email address domain to retrieve the mail server host names and addresses used by the domain. The query is performed according to RFC 2821 section 5 Address Resolution and Mail Handling and begins by first contacting the name server set through the *::SetNameServer* method requesting all DNS records for the email address domain.

This method is dependent on values set by the *::CheckSyntax* method. If no email address has been validated it will exit setting the object's error code to ERR_NO_ADDRESS.

Related Validation Property	Effect
B_REQUIRE_DNS_MX	Require at least one DNS MX record even if an A record is present. A valid domain will always have a DNS A (address) record however it may not have MX records. Per RFC 2821 if the domain does not have DNS mail exchange records then email delivery should be attempted through the domain's address record.

Parameters:

none, relies upon object internal state

Returns:

object error code

::CheckSmtp()

Performs SMTP validation of the current email address by contacting its mail servers and asking them to verify the recipient. Verification is accomplished by initiating an email transfer request from your domain to the current email address and interpreting the SMPT protocol RCPT command result code. At this point the SMTP session is reset canceling the conversation thus no email is actually sent to the email address.

This method is dependent on values set by the *::CheckSyntax* and *::CheckDns* methods. If no email address has been validated it will exit setting the object's error code to ERR_NO_ADDRESS. If no mail servers have been found (or initialized) by a DNS query the method will exit setting the object's error code to ERR_SMTP_NO_SERVERS;

Related Validation Property	Effect
B_TEST_CATCHALL	SMTP verification includes a catchall mailbox test. This test is used to determine whether the email server for the domain will accept any email address sent to the domain.

Parameters:

none, relies upon object internal state

Returns:

object error code

::Validate(\$address,\$level = LEVEL_SYNTAX)

Validate the specified email address. The actual validation performed depends upon the level of validation requested and whether errors are detected at a particular step. The validation property methods may be called prior to the validation call to tighten address requirements. For example, you can specify whether or not to accept email addresses from well know free email domains. Thus an address such as *anyuser@yahoo.com* would pass syntactic validation but would be rejected because of its domain.

Whether or not an email address is valid or invalid is not always clear cut. The return value from this method is the last level of validation performed. This value and the object's error code should both be viewed to make your decision about a particular email address.

Parameters:

\$address

the email address to validate

\$level

validation level to perform, accepted values are

- LEVEL_SYNTAX (1) perform syntax validation only (default)
- LEVEL_DNS (2) perform both syntax and DNS lookup validation
- LEVEL_SMTP (3) perform syntax, DNS lookup, and SMTP validation

Returns:

- LEVEL_INVALID, address clearly invalid
- LEVEL_SYNTAX, syntax validation complete
- LEVEL_DNS, DNS validation complete
- LEVEL_SMTP, validation complete

Email Address Validation Class References

The email address validation library was written to and conforms with the following Internet standards and recommendations relating to email addresses, domain names, SMTP and DNS protocols.

Reference	Description
RFC 821	Simple Mail Transfer Protocol (Status: STANDARD) Obsoletes RFC 788 Obsoleted by RFC 2821 Also STD0010
RFC 822	Standard for the format of ARPA Internet text messages (Status: STANDARD) Obsoletes RFC 733 Obsoleted by RFC 2822 Updated by RFC 1123, RFC 1138, RFC 1148, RFC 1327, RFC 2156 Also STD0011
RFC 1123	Requirements for Internet Hosts
RFC 1918	Address Allocation for Private Internets (Status: BEST CURRENT PRACTICE) Obsoletes RFC 1627, RFC 1597
RFC 2821	Simple Mail Transfer Protocol (Status: PROPOSED STANDARD) Obsoletes RFC 821, RFC 974, RFC 1869
RFC 2822	Internet Message Format (Status: PROPOSED STANDARD) Obsoletes RFC 822
RFC 1035	Domain names - implementation and specification (Status: STANDARD) Obsoletes RFC 973, RFC 882, RFC 883 Updated by RFC 1101, RFC 1183, RFC 1348, RFC 1876, RFC 1982, RFC 1995, RFC 1996, RFC 2065, RFC 2136, RFC 2181, RFC 2137, RFC 2308, RFC 2535, RFC 2845, RFC 3425 Also STD0013
RFC 1597	Address Allocation for Private Internets (Status: INFORMATIONAL) Obsoleted by RFC 1918
STD 13	