**Internet and Web Programming**

**Course Code-CSE3002**

**Slot-F1**

**Digital Assignment -01**

**Name-Shadab Alam**

**Reg.No-18BCE2491**

**Code of Index.php**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<title>CEE LOGIN</title>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link rel="icon" type="phpmailer\_email/image/png" href="images/icons/favicon.ico"/>**

**<link rel="stylesheet" type="text/css" href="vendor/bootstrap/css/bootstrap.min.css">**

**<link rel="stylesheet" type="text/css" href="fonts/font-awesome-4.7.0/css/font-awesome.min.css">**

**<link rel="stylesheet" type="text/css" href="fonts/Linearicons-Free-v1.0.0/icon-font.min.css">**

**<link rel="stylesheet" type="text/css" href="vendor/animate/animate.css">**

**<link rel="stylesheet" type="text/css" href="vendor/css-hamburgers/hamburgers.min.css">**

**<link rel="stylesheet" type="text/css" href="vendor/select2/select2.min.css">**

**<link rel="stylesheet" type="text/css" href="daterangepicker/daterangepicker.css">**

**<link rel="stylesheet" type="text/css" href="css/util.css">**

**<link rel="stylesheet" type="text/css" href="css/main.css">**

**</head>**

**<body>**

**<div class="limiter">**

**<div class="container-login100">**

**<div class="wrap-login100">**

**<div class="login100-form-title" style="background-image: url(login-ui/images/bg-01.jpg);">**

**<span class="login100-form-title-1">**

**Send Email**

**</span><br>**

**</div><br>**

**<form action="send.php" method="post" enctype="multipart/form-data">**

**<label for="id" class="text-center">Enter Your Email**

**<span style="color: red;">\*</span>**

**</label>**

**<input type="text" id="id" name="email" class="form-control" placeholder="Enater Your Email" required>**

**<label for="id1" class="text-center">Enter Your Subject**

**<span style="color: red;">\*</span>**

**</label>**

**<input type="text" id="id1" name="subject" class="form-control" placeholder="Enater Your Subject" required="">**

**<label for="id4" class="text-center">Enter Your Message**

**<span style="color: red;">\*</span>**

**</label>**

**<textarea id="id4" name="message" class="form-control" placeholder="Enater Your Message" required=""></textarea>**

**<label for="id2" class="text-center">Attach Your File**

**<span style="color: red;">\*</span>**

**</label>**

**<input type="file" name="file" class="form-control" placeholder="Enater Your Message" required><br>**

**<input type="submit" name="send" class="btn btn-primary col-12" value="Send Email">**

**</form>**

**</div>**

**</div>**

**</div>**

**<div style="box-shadow:3px 8px 10px 3px">**

**</div>**

**</div>**

**<script src="login-ui/vendor/jquery/jquery-3.2.1.min.js"></script>**

**<script src="login-ui/vendor/animsition/js/animsition.min.js"></script>**

**<script src="login-ui/vendor/bootstrap/js/popper.js"></script>**

**<script src="login-ui/vendor/bootstrap/js/bootstrap.min.js"></script>**

**<script src="login-ui/vendor/select2/select2.min.js"></script>**

**<script src="login-ui/vendor/daterangepicker/moment.min.js"></script>**

**<script src="login-ui/vendor/daterangepicker/daterangepicker.js"></script>**

**<script src="login-ui/vendor/countdowntime/countdowntime.js"></script>**

**<script src="login-ui/js/main.js"></script>**

**</body>**

**</html>**

**Code Of Credential.php**

**<?php**

**define("EMAILID",'waquarahmad1213@gmail.com');**

**define('PASSWORD','waquar123@');**

**?>**

**Code Of send.php**

**<?php**

**require 'phpmailer/PHPMailerAutoload.php';**

**require 'credential.php';**

**if (isset($\_POST['send']))**

**{**

**$to=$\_POST['email'];**

**$subject=$\_POST['subject'];**

**$message=$\_POST['message'];**

**$mail = new PHPMailer;**

**//$mail->SMTPDebug = 2; // Enable verbose debug output**

**$mail->isSMTP(); // Set mailer to use SMTP**

**$mail->Host = 'smtp.gmail.com'; // Specify main and backup SMTP servers**

**$mail->SMTPAuth = true; // Enable SMTP authentication**

**$mail->Username = EMAILID; // SMTP username**

**$mail->Password = PASSWORD; // SMTP password**

**$mail->SMTPSecure = 'tls'; // Enable TLS encryption, `ssl` also accepted**

**$mail->Port = 587; // TCP port to connect to**

**$mail->setFrom(EMAILID,'WaquarAhmad');**

**$mail->addAddress($to); // Name is optional**

**$mail->addReplyTo(EMAILID,'Information');**

**//$mail->addCC('cc@example.com');**

**//$mail->addBCC('bcc@example.com');**

**$file\_name=$\_FILES["file"]["name"];**

**move\_uploaded\_file($\_FILES["file"]["tmp\_name"],$file\_name);**

**$mail->addAttachment($file\_name); // Add attachments**

**// Optional name**

**$mail->isHTML(true); // Set email format to HTML**

**$mail->Subject = $subject;**

**$mail->Body = '<div style="background-color:orange;border-radius:10px;padding:10px;box-shadow:3px 8px 10px 3px";border="1px" solid black;height="250px"><h1>'.$message.'</h1></div>';**

**$mail->AltBody = 'This is the body in plain text for non-HTML mail clients';**

**if(!$mail->send()) {**

**echo 'Message could not be sent.';**

**echo 'Mailer Error: ' . $mail->ErrorInfo;**

**} else {**

**echo '**

**<div class="row">**

**<div class="col-6 m-auto">**

**<div class="alert alert-success text-center alert-dismissible fade show" role="alert">**

**Thanks:) Your message has been sent Successfully**

**<button type="button" class="close" data-dismiss="alert" aria-label="Close">**

**<span aria-hidden="true">&times;</span>**

**</button>**

**</div>**

**</div>**

**</div>';**

**}**

**}**

**?>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title></title>**

**<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">**

**<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>**

**<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>**

**<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>**

**</head>**

**<body class="bg-info">**

**<div class="container-fluid mt-5">**

**<div class="row">**

**<div class="col-6 m-auto">**

**<h1><a href="index.php" class="btn btn-primary">Send Again Mail</a></h1>**

**</div>**

**</div>**

**</div>**

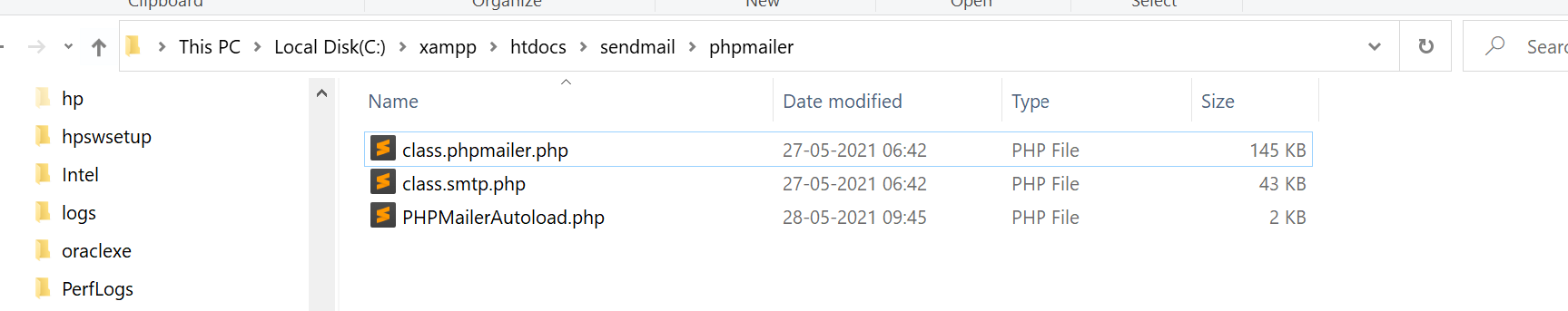
**</body>**

**</html>**

**Now the all php code are in Sendmail folder;**

**5:Now the again create folder name(phpmailer)**

**and inside the phpmailer folder are three php code**

**1:**

**Code of class.phpmailer.php**

**<?php**

**/\*\***

**\* PHPMailer - PHP email creation and transport class.**

**\* PHP Version 5**

**\* @package PHPMailer**

**\* @link https://github.com/PHPMailer/PHPMailer/ The PHPMailer GitHub project**

**\* @author Marcus Bointon (Synchro/coolbru) <phpmailer@synchromedia.co.uk>**

**\* @author Jim Jagielski (jimjag) <jimjag@gmail.com>**

**\* @author Andy Prevost (codeworxtech) <codeworxtech@users.sourceforge.net>**

**\* @author Brent R. Matzelle (original founder)**

**\* @copyright 2012 - 2014 Marcus Bointon**

**\* @copyright 2010 - 2012 Jim Jagielski**

**\* @copyright 2004 - 2009 Andy Prevost**

**\* @license http://www.gnu.org/copyleft/lesser.html GNU Lesser General Public License**

**\* @note This program is distributed in the hope that it will be useful - WITHOUT**

**\* ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or**

**\* FITNESS FOR A PARTICULAR PURPOSE.**

**\*/**

**/\*\***

**\* PHPMailer - PHP email creation and transport class.**

**\* @package PHPMailer**

**\* @author Marcus Bointon (Synchro/coolbru) <phpmailer@synchromedia.co.uk>**

**\* @author Jim Jagielski (jimjag) <jimjag@gmail.com>**

**\* @author Andy Prevost (codeworxtech) <codeworxtech@users.sourceforge.net>**

**\* @author Brent R. Matzelle (original founder)**

**\*/**

**class PHPMailer**

**{**

**/\*\***

**\* The PHPMailer Version number.**

**\* @var string**

**\*/**

**public $Version = '5.2.28';**

**/\*\***

**\* Email priority.**

**\* Options: null (default), 1 = High, 3 = Normal, 5 = low.**

**\* When null, the header is not set at all.**

**\* @var integer**

**\*/**

**public $Priority = null;**

**/\*\***

**\* The character set of the message.**

**\* @var string**

**\*/**

**public $CharSet = 'iso-8859-1';**

**/\*\***

**\* The MIME Content-type of the message.**

**\* @var string**

**\*/**

**public $ContentType = 'text/plain';**

**/\*\***

**\* The message encoding.**

**\* Options: "8bit", "7bit", "binary", "base64", and "quoted-printable".**

**\* @var string**

**\*/**

**public $Encoding = '8bit';**

**/\*\***

**\* Holds the most recent mailer error message.**

**\* @var string**

**\*/**

**public $ErrorInfo = '';**

**/\*\***

**\* The From email address for the message.**

**\* @var string**

**\*/**

**public $From = 'root@localhost';**

**/\*\***

**\* The From name of the message.**

**\* @var string**

**\*/**

**public $FromName = 'Root User';**

**/\*\***

**\* The Sender email (Return-Path) of the message.**

**\* If not empty, will be sent via -f to sendmail or as 'MAIL FROM' in smtp mode.**

**\* @var string**

**\*/**

**public $Sender = '';**

**/\*\***

**\* The Return-Path of the message.**

**\* If empty, it will be set to either From or Sender.**

**\* @var string**

**\* @deprecated Email senders should never set a return-path header;**

**\* it's the receiver's job (RFC5321 section 4.4), so this no longer does anything.**

**\* @link https://tools.ietf.org/html/rfc5321#section-4.4 RFC5321 reference**

**\*/**

**public $ReturnPath = '';**

**/\*\***

**\* The Subject of the message.**

**\* @var string**

**\*/**

**public $Subject = '';**

**/\*\***

**\* An HTML or plain text message body.**

**\* If HTML then call isHTML(true).**

**\* @var string**

**\*/**

**public $Body = '';**

**/\*\***

**\* The plain-text message body.**

**\* This body can be read by mail clients that do not have HTML email**

**\* capability such as mutt & Eudora.**

**\* Clients that can read HTML will view the normal Body.**

**\* @var string**

**\*/**

**public $AltBody = '';**

**/\*\***

**\* An iCal message part body.**

**\* Only supported in simple alt or alt\_inline message types**

**\* To generate iCal events, use the bundled extras/EasyPeasyICS.php class or iCalcreator**

**\* @link http://sprain.ch/blog/downloads/php-class-easypeasyics-create-ical-files-with-php/**

**\* @link http://kigkonsult.se/iCalcreator/**

**\* @var string**

**\*/**

**public $Ical = '';**

**/\*\***

**\* The complete compiled MIME message body.**

**\* @access protected**

**\* @var string**

**\*/**

**protected $MIMEBody = '';**

**/\*\***

**\* The complete compiled MIME message headers.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $MIMEHeader = '';**

**/\*\***

**\* Extra headers that createHeader() doesn't fold in.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $mailHeader = '';**

**/\*\***

**\* Word-wrap the message body to this number of chars.**

**\* Set to 0 to not wrap. A useful value here is 78, for RFC2822 section 2.1.1 compliance.**

**\* @var integer**

**\*/**

**public $WordWrap = 0;**

**/\*\***

**\* Which method to use to send mail.**

**\* Options: "mail", "sendmail", or "smtp".**

**\* @var string**

**\*/**

**public $Mailer = 'mail';**

**/\*\***

**\* The path to the sendmail program.**

**\* @var string**

**\*/**

**public $Sendmail = '/usr/sbin/sendmail';**

**/\*\***

**\* Whether mail() uses a fully sendmail-compatible MTA.**

**\* One which supports sendmail's "-oi -f" options.**

**\* @var boolean**

**\*/**

**public $UseSendmailOptions = true;**

**/\*\***

**\* Path to PHPMailer plugins.**

**\* Useful if the SMTP class is not in the PHP include path.**

**\* @var string**

**\* @deprecated Should not be needed now there is an autoloader.**

**\*/**

**public $PluginDir = '';**

**/\*\***

**\* The email address that a reading confirmation should be sent to, also known as read receipt.**

**\* @var string**

**\*/**

**public $ConfirmReadingTo = '';**

**/\*\***

**\* The hostname to use in the Message-ID header and as default HELO string.**

**\* If empty, PHPMailer attempts to find one with, in order,**

**\* $\_SERVER['SERVER\_NAME'], gethostname(), php\_uname('n'), or the value**

**\* 'localhost.localdomain'.**

**\* @var string**

**\*/**

**public $Hostname = '';**

**/\*\***

**\* An ID to be used in the Message-ID header.**

**\* If empty, a unique id will be generated.**

**\* You can set your own, but it must be in the format "<id@domain>",**

**\* as defined in RFC5322 section 3.6.4 or it will be ignored.**

**\* @see https://tools.ietf.org/html/rfc5322#section-3.6.4**

**\* @var string**

**\*/**

**public $MessageID = '';**

**/\*\***

**\* The message Date to be used in the Date header.**

**\* If empty, the current date will be added.**

**\* @var string**

**\*/**

**public $MessageDate = '';**

**/\*\***

**\* SMTP hosts.**

**\* Either a single hostname or multiple semicolon-delimited hostnames.**

**\* You can also specify a different port**

**\* for each host by using this format: [hostname:port]**

**\* (e.g. "smtp1.example.com:25;smtp2.example.com").**

**\* You can also specify encryption type, for example:**

**\* (e.g. "tls://smtp1.example.com:587;ssl://smtp2.example.com:465").**

**\* Hosts will be tried in order.**

**\* @var string**

**\*/**

**public $Host = 'localhost';**

**/\*\***

**\* The default SMTP server port.**

**\* @var integer**

**\* @TODO Why is this needed when the SMTP class takes care of it?**

**\*/**

**public $Port = 25;**

**/\*\***

**\* The SMTP HELO of the message.**

**\* Default is $Hostname. If $Hostname is empty, PHPMailer attempts to find**

**\* one with the same method described above for $Hostname.**

**\* @var string**

**\* @see PHPMailer::$Hostname**

**\*/**

**public $Helo = '';**

**/\*\***

**\* What kind of encryption to use on the SMTP connection.**

**\* Options: '', 'ssl' or 'tls'**

**\* @var string**

**\*/**

**public $SMTPSecure = '';**

**/\*\***

**\* Whether to enable TLS encryption automatically if a server supports it,**

**\* even if `SMTPSecure` is not set to 'tls'.**

**\* Be aware that in PHP >= 5.6 this requires that the server's certificates are valid.**

**\* @var boolean**

**\*/**

**public $SMTPAutoTLS = true;**

**/\*\***

**\* Whether to use SMTP authentication.**

**\* Uses the Username and Password properties.**

**\* @var boolean**

**\* @see PHPMailer::$Username**

**\* @see PHPMailer::$Password**

**\*/**

**public $SMTPAuth = false;**

**/\*\***

**\* Options array passed to stream\_context\_create when connecting via SMTP.**

**\* @var array**

**\*/**

**public $SMTPOptions = array();**

**/\*\***

**\* SMTP username.**

**\* @var string**

**\*/**

**public $Username = '';**

**/\*\***

**\* SMTP password.**

**\* @var string**

**\*/**

**public $Password = '';**

**/\*\***

**\* SMTP auth type.**

**\* Options are CRAM-MD5, LOGIN, PLAIN, NTLM, XOAUTH2, attempted in that order if not specified**

**\* @var string**

**\*/**

**public $AuthType = '';**

**/\*\***

**\* SMTP realm.**

**\* Used for NTLM auth**

**\* @var string**

**\*/**

**public $Realm = '';**

**/\*\***

**\* SMTP workstation.**

**\* Used for NTLM auth**

**\* @var string**

**\*/**

**public $Workstation = '';**

**/\*\***

**\* The SMTP server timeout in seconds.**

**\* Default of 5 minutes (300sec) is from RFC2821 section 4.5.3.2**

**\* @var integer**

**\*/**

**public $Timeout = 300;**

**/\*\***

**\* SMTP class debug output mode.**

**\* Debug output level.**

**\* Options:**

**\* \* `0` No output**

**\* \* `1` Commands**

**\* \* `2` Data and commands**

**\* \* `3` As 2 plus connection status**

**\* \* `4` Low-level data output**

**\* @var integer**

**\* @see SMTP::$do\_debug**

**\*/**

**public $SMTPDebug = 0;**

**/\*\***

**\* How to handle debug output.**

**\* Options:**

**\* \* `echo` Output plain-text as-is, appropriate for CLI**

**\* \* `html` Output escaped, line breaks converted to `<br>`, appropriate for browser output**

**\* \* `error\_log` Output to error log as configured in php.ini**

**\***

**\* Alternatively, you can provide a callable expecting two params: a message string and the debug level:**

**\* <code>**

**\* $mail->Debugoutput = function($str, $level) {echo "debug level $level; message: $str";};**

**\* </code>**

**\* @var string|callable**

**\* @see SMTP::$Debugoutput**

**\*/**

**public $Debugoutput = 'echo';**

**/\*\***

**\* Whether to keep SMTP connection open after each message.**

**\* If this is set to true then to close the connection**

**\* requires an explicit call to smtpClose().**

**\* @var boolean**

**\*/**

**public $SMTPKeepAlive = false;**

**/\*\***

**\* Whether to split multiple to addresses into multiple messages**

**\* or send them all in one message.**

**\* Only supported in `mail` and `sendmail` transports, not in SMTP.**

**\* @var boolean**

**\*/**

**public $SingleTo = false;**

**/\*\***

**\* Storage for addresses when SingleTo is enabled.**

**\* @var array**

**\* @TODO This should really not be public**

**\*/**

**public $SingleToArray = array();**

**/\*\***

**\* Whether to generate VERP addresses on send.**

**\* Only applicable when sending via SMTP.**

**\* @link https://en.wikipedia.org/wiki/Variable\_envelope\_return\_path**

**\* @link http://www.postfix.org/VERP\_README.html Postfix VERP info**

**\* @var boolean**

**\*/**

**public $do\_verp = false;**

**/\*\***

**\* Whether to allow sending messages with an empty body.**

**\* @var boolean**

**\*/**

**public $AllowEmpty = false;**

**/\*\***

**\* The default line ending.**

**\* @note The default remains "\n". We force CRLF where we know**

**\* it must be used via self::CRLF.**

**\* @var string**

**\*/**

**public $LE = "\n";**

**/\*\***

**\* DKIM selector.**

**\* @var string**

**\*/**

**public $DKIM\_selector = '';**

**/\*\***

**\* DKIM Identity.**

**\* Usually the email address used as the source of the email.**

**\* @var string**

**\*/**

**public $DKIM\_identity = '';**

**/\*\***

**\* DKIM passphrase.**

**\* Used if your key is encrypted.**

**\* @var string**

**\*/**

**public $DKIM\_passphrase = '';**

**/\*\***

**\* DKIM signing domain name.**

**\* @example 'example.com'**

**\* @var string**

**\*/**

**public $DKIM\_domain = '';**

**/\*\***

**\* DKIM private key file path.**

**\* @var string**

**\*/**

**public $DKIM\_private = '';**

**/\*\***

**\* DKIM private key string.**

**\* If set, takes precedence over `$DKIM\_private`.**

**\* @var string**

**\*/**

**public $DKIM\_private\_string = '';**

**/\*\***

**\* Callback Action function name.**

**\***

**\* The function that handles the result of the send email action.**

**\* It is called out by send() for each email sent.**

**\***

**\* Value can be any php callable: http://www.php.net/is\_callable**

**\***

**\* Parameters:**

**\* boolean $result result of the send action**

**\* array $to email addresses of the recipients**

**\* array $cc cc email addresses**

**\* array $bcc bcc email addresses**

**\* string $subject the subject**

**\* string $body the email body**

**\* string $from email address of sender**

**\* @var string**

**\*/**

**public $action\_function = '';**

**/\*\***

**\* What to put in the X-Mailer header.**

**\* Options: An empty string for PHPMailer default, whitespace for none, or a string to use**

**\* @var string**

**\*/**

**public $XMailer = '';**

**/\*\***

**\* Which validator to use by default when validating email addresses.**

**\* May be a callable to inject your own validator, but there are several built-in validators.**

**\* @see PHPMailer::validateAddress()**

**\* @var string|callable**

**\* @static**

**\*/**

**public static $validator = 'auto';**

**/\*\***

**\* An instance of the SMTP sender class.**

**\* @var SMTP**

**\* @access protected**

**\*/**

**protected $smtp = null;**

**/\*\***

**\* The array of 'to' names and addresses.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $to = array();**

**/\*\***

**\* The array of 'cc' names and addresses.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $cc = array();**

**/\*\***

**\* The array of 'bcc' names and addresses.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $bcc = array();**

**/\*\***

**\* The array of reply-to names and addresses.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $ReplyTo = array();**

**/\*\***

**\* An array of all kinds of addresses.**

**\* Includes all of $to, $cc, $bcc**

**\* @var array**

**\* @access protected**

**\* @see PHPMailer::$to @see PHPMailer::$cc @see PHPMailer::$bcc**

**\*/**

**protected $all\_recipients = array();**

**/\*\***

**\* An array of names and addresses queued for validation.**

**\* In send(), valid and non duplicate entries are moved to $all\_recipients**

**\* and one of $to, $cc, or $bcc.**

**\* This array is used only for addresses with IDN.**

**\* @var array**

**\* @access protected**

**\* @see PHPMailer::$to @see PHPMailer::$cc @see PHPMailer::$bcc**

**\* @see PHPMailer::$all\_recipients**

**\*/**

**protected $RecipientsQueue = array();**

**/\*\***

**\* An array of reply-to names and addresses queued for validation.**

**\* In send(), valid and non duplicate entries are moved to $ReplyTo.**

**\* This array is used only for addresses with IDN.**

**\* @var array**

**\* @access protected**

**\* @see PHPMailer::$ReplyTo**

**\*/**

**protected $ReplyToQueue = array();**

**/\*\***

**\* The array of attachments.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $attachment = array();**

**/\*\***

**\* The array of custom headers.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $CustomHeader = array();**

**/\*\***

**\* The most recent Message-ID (including angular brackets).**

**\* @var string**

**\* @access protected**

**\*/**

**protected $lastMessageID = '';**

**/\*\***

**\* The message's MIME type.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $message\_type = '';**

**/\*\***

**\* The array of MIME boundary strings.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $boundary = array();**

**/\*\***

**\* The array of available languages.**

**\* @var array**

**\* @access protected**

**\*/**

**protected $language = array();**

**/\*\***

**\* The number of errors encountered.**

**\* @var integer**

**\* @access protected**

**\*/**

**protected $error\_count = 0;**

**/\*\***

**\* The S/MIME certificate file path.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $sign\_cert\_file = '';**

**/\*\***

**\* The S/MIME key file path.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $sign\_key\_file = '';**

**/\*\***

**\* The optional S/MIME extra certificates ("CA Chain") file path.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $sign\_extracerts\_file = '';**

**/\*\***

**\* The S/MIME password for the key.**

**\* Used only if the key is encrypted.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $sign\_key\_pass = '';**

**/\*\***

**\* Whether to throw exceptions for errors.**

**\* @var boolean**

**\* @access protected**

**\*/**

**protected $exceptions = false;**

**/\*\***

**\* Unique ID used for message ID and boundaries.**

**\* @var string**

**\* @access protected**

**\*/**

**protected $uniqueid = '';**

**/\*\***

**\* Error severity: message only, continue processing.**

**\*/**

**const STOP\_MESSAGE = 0;**

**/\*\***

**\* Error severity: message, likely ok to continue processing.**

**\*/**

**const STOP\_CONTINUE = 1;**

**/\*\***

**\* Error severity: message, plus full stop, critical error reached.**

**\*/**

**const STOP\_CRITICAL = 2;**

**/\*\***

**\* SMTP RFC standard line ending.**

**\*/**

**const CRLF = "\r\n";**

**/\*\***

**\* The maximum line length allowed by RFC 2822 section 2.1.1**

**\* @var integer**

**\*/**

**const MAX\_LINE\_LENGTH = 998;**

**/\*\***

**\* Constructor.**

**\* @param boolean $exceptions Should we throw external exceptions?**

**\*/**

**public function \_\_construct($exceptions = null)**

**{**

**if ($exceptions !== null) {**

**$this->exceptions = (boolean)$exceptions;**

**}**

**//Pick an appropriate debug output format automatically**

**$this->Debugoutput = (strpos(PHP\_SAPI, 'cli') !== false ? 'echo' : 'html');**

**}**

**/\*\***

**\* Destructor.**

**\*/**

**public function \_\_destruct()**

**{**

**//Close any open SMTP connection nicely**

**$this->smtpClose();**

**}**

**/\*\***

**\* Call mail() in a safe\_mode-aware fashion.**

**\* Also, unless sendmail\_path points to sendmail (or something that**

**\* claims to be sendmail), don't pass params (not a perfect fix,**

**\* but it will do)**

**\* @param string $to To**

**\* @param string $subject Subject**

**\* @param string $body Message Body**

**\* @param string $header Additional Header(s)**

**\* @param string $params Params**

**\* @access private**

**\* @return boolean**

**\*/**

**private function mailPassthru($to, $subject, $body, $header, $params)**

**{**

**//Check overloading of mail function to avoid double-encoding**

**if (ini\_get('mbstring.func\_overload') & 1) {**

**$subject = $this->secureHeader($subject);**

**} else {**

**$subject = $this->encodeHeader($this->secureHeader($subject));**

**}**

**//Can't use additional\_parameters in safe\_mode, calling mail() with null params breaks**

**//@link http://php.net/manual/en/function.mail.php**

**if (ini\_get('safe\_mode') or !$this->UseSendmailOptions or is\_null($params)) {**

**$result = @mail($to, $subject, $body, $header);**

**} else {**

**$result = @mail($to, $subject, $body, $header, $params);**

**}**

**return $result;**

**}**

**/\*\***

**\* Output debugging info via user-defined method.**

**\* Only generates output if SMTP debug output is enabled (@see SMTP::$do\_debug).**

**\* @see PHPMailer::$Debugoutput**

**\* @see PHPMailer::$SMTPDebug**

**\* @param string $str**

**\*/**

**protected function edebug($str)**

**{**

**if ($this->SMTPDebug <= 0) {**

**return;**

**}**

**//Avoid clash with built-in function names**

**if (!in\_array($this->Debugoutput, array('error\_log', 'html', 'echo')) and is\_callable($this->Debugoutput)) {**

**call\_user\_func($this->Debugoutput, $str, $this->SMTPDebug);**

**return;**

**}**

**switch ($this->Debugoutput) {**

**case 'error\_log':**

**//Don't output, just log**

**error\_log($str);**

**break;**

**case 'html':**

**//Cleans up output a bit for a better looking, HTML-safe output**

**echo htmlentities(**

**preg\_replace('/[\r\n]+/', '', $str),**

**ENT\_QUOTES,**

**'UTF-8'**

**)**

**. "<br>\n";**

**break;**

**case 'echo':**

**default:**

**//Normalize line breaks**

**$str = preg\_replace('/\r\n?/ms', "\n", $str);**

**echo gmdate('Y-m-d H:i:s') . "\t" . str\_replace(**

**"\n",**

**"\n \t ",**

**trim($str)**

**) . "\n";**

**}**

**}**

**/\*\***

**\* Sets message type to HTML or plain.**

**\* @param boolean $isHtml True for HTML mode.**

**\* @return void**

**\*/**

**public function isHTML($isHtml = true)**

**{**

**if ($isHtml) {**

**$this->ContentType = 'text/html';**

**} else {**

**$this->ContentType = 'text/plain';**

**}**

**}**

**/\*\***

**\* Send messages using SMTP.**

**\* @return void**

**\*/**

**public function isSMTP()**

**{**

**$this->Mailer = 'smtp';**

**}**

**/\*\***

**\* Send messages using PHP's mail() function.**

**\* @return void**

**\*/**

**public function isMail()**

**{**

**$this->Mailer = 'mail';**

**}**

**/\*\***

**\* Send messages using $Sendmail.**

**\* @return void**

**\*/**

**public function isSendmail()**

**{**

**$ini\_sendmail\_path = ini\_get('sendmail\_path');**

**if (!stristr($ini\_sendmail\_path, 'sendmail')) {**

**$this->Sendmail = '/usr/sbin/sendmail';**

**} else {**

**$this->Sendmail = $ini\_sendmail\_path;**

**}**

**$this->Mailer = 'sendmail';**

**}**

**/\*\***

**\* Send messages using qmail.**

**\* @return void**

**\*/**

**public function isQmail()**

**{**

**$ini\_sendmail\_path = ini\_get('sendmail\_path');**

**if (!stristr($ini\_sendmail\_path, 'qmail')) {**

**$this->Sendmail = '/var/qmail/bin/qmail-inject';**

**} else {**

**$this->Sendmail = $ini\_sendmail\_path;**

**}**

**$this->Mailer = 'qmail';**

**}**

**/\*\***

**\* Add a "To" address.**

**\* @param string $address The email address to send to**

**\* @param string $name**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\*/**

**public function addAddress($address, $name = '')**

**{**

**return $this->addOrEnqueueAnAddress('to', $address, $name);**

**}**

**/\*\***

**\* Add a "CC" address.**

**\* @note: This function works with the SMTP mailer on win32, not with the "mail" mailer.**

**\* @param string $address The email address to send to**

**\* @param string $name**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\*/**

**public function addCC($address, $name = '')**

**{**

**return $this->addOrEnqueueAnAddress('cc', $address, $name);**

**}**

**/\*\***

**\* Add a "BCC" address.**

**\* @note: This function works with the SMTP mailer on win32, not with the "mail" mailer.**

**\* @param string $address The email address to send to**

**\* @param string $name**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\*/**

**public function addBCC($address, $name = '')**

**{**

**return $this->addOrEnqueueAnAddress('bcc', $address, $name);**

**}**

**/\*\***

**\* Add a "Reply-To" address.**

**\* @param string $address The email address to reply to**

**\* @param string $name**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\*/**

**public function addReplyTo($address, $name = '')**

**{**

**return $this->addOrEnqueueAnAddress('Reply-To', $address, $name);**

**}**

**/\*\***

**\* Add an address to one of the recipient arrays or to the ReplyTo array. Because PHPMailer**

**\* can't validate addresses with an IDN without knowing the PHPMailer::$CharSet (that can still**

**\* be modified after calling this function), addition of such addresses is delayed until send().**

**\* Addresses that have been added already return false, but do not throw exceptions.**

**\* @param string $kind One of 'to', 'cc', 'bcc', or 'ReplyTo'**

**\* @param string $address The email address to send, resp. to reply to**

**\* @param string $name**

**\* @throws phpmailerException**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\* @access protected**

**\*/**

**protected function addOrEnqueueAnAddress($kind, $address, $name)**

**{**

**$address = trim($address);**

**$name = trim(preg\_replace('/[\r\n]+/', '', $name)); //Strip breaks and trim**

**if (($pos = strrpos($address, '@')) === false) {**

**// At-sign is misssing.**

**$error\_message = $this->lang('invalid\_address') . " (addAnAddress $kind): $address";**

**$this->setError($error\_message);**

**$this->edebug($error\_message);**

**if ($this->exceptions) {**

**throw new phpmailerException($error\_message);**

**}**

**return false;**

**}**

**$params = array($kind, $address, $name);**

**// Enqueue addresses with IDN until we know the PHPMailer::$CharSet.**

**if ($this->has8bitChars(substr($address, ++$pos)) and $this->idnSupported()) {**

**if ($kind != 'Reply-To') {**

**if (!array\_key\_exists($address, $this->RecipientsQueue)) {**

**$this->RecipientsQueue[$address] = $params;**

**return true;**

**}**

**} else {**

**if (!array\_key\_exists($address, $this->ReplyToQueue)) {**

**$this->ReplyToQueue[$address] = $params;**

**return true;**

**}**

**}**

**return false;**

**}**

**// Immediately add standard addresses without IDN.**

**return call\_user\_func\_array(array($this, 'addAnAddress'), $params);**

**}**

**/\*\***

**\* Add an address to one of the recipient arrays or to the ReplyTo array.**

**\* Addresses that have been added already return false, but do not throw exceptions.**

**\* @param string $kind One of 'to', 'cc', 'bcc', or 'ReplyTo'**

**\* @param string $address The email address to send, resp. to reply to**

**\* @param string $name**

**\* @throws phpmailerException**

**\* @return boolean true on success, false if address already used or invalid in some way**

**\* @access protected**

**\*/**

**protected function addAnAddress($kind, $address, $name = '')**

**{**

**if (!in\_array($kind, array('to', 'cc', 'bcc', 'Reply-To'))) {**

**$error\_message = $this->lang('Invalid recipient kind: ') . $kind;**

**$this->setError($error\_message);**

**$this->edebug($error\_message);**

**if ($this->exceptions) {**

**throw new phpmailerException($error\_message);**

**}**

**return false;**

**}**

**if (!$this->validateAddress($address)) {**

**$error\_message = $this->lang('invalid\_address') . " (addAnAddress $kind): $address";**

**$this->setError($error\_message);**

**$this->edebug($error\_message);**

**if ($this->exceptions) {**

**throw new phpmailerException($error\_message);**

**}**

**return false;**

**}**

**if ($kind != 'Reply-To') {**

**if (!array\_key\_exists(strtolower($address), $this->all\_recipients)) {**

**array\_push($this->$kind, array($address, $name));**

**$this->all\_recipients[strtolower($address)] = true;**

**return true;**

**}**

**} else {**

**if (!array\_key\_exists(strtolower($address), $this->ReplyTo)) {**

**$this->ReplyTo[strtolower($address)] = array($address, $name);**

**return true;**

**}**

**}**

**return false;**

**}**

**/\*\***

**\* Parse and validate a string containing one or more RFC822-style comma-separated email addresses**

**\* of the form "display name <address>" into an array of name/address pairs.**

**\* Uses the imap\_rfc822\_parse\_adrlist function if the IMAP extension is available.**

**\* Note that quotes in the name part are removed.**

**\* @param string $addrstr The address list string**

**\* @param bool $useimap Whether to use the IMAP extension to parse the list**

**\* @return array**

**\* @link http://www.andrew.cmu.edu/user/agreen1/testing/mrbs/web/Mail/RFC822.php A more careful implementation**

**\*/**

**public function parseAddresses($addrstr, $useimap = true)**

**{**

**$addresses = array();**

**if ($useimap and function\_exists('imap\_rfc822\_parse\_adrlist')) {**

**//Use this built-in parser if it's available**

**$list = imap\_rfc822\_parse\_adrlist($addrstr, '');**

**foreach ($list as $address) {**

**if ($address->host != '.SYNTAX-ERROR.') {**

**if ($this->validateAddress($address->mailbox . '@' . $address->host)) {**

**$addresses[] = array(**

**'name' => (property\_exists($address, 'personal') ? $address->personal : ''),**

**'address' => $address->mailbox . '@' . $address->host**

**);**

**}**

**}**

**}**

**} else {**

**//Use this simpler parser**

**$list = explode(',', $addrstr);**

**foreach ($list as $address) {**

**$address = trim($address);**

**//Is there a separate name part?**

**if (strpos($address, '<') === false) {**

**//No separate name, just use the whole thing**

**if ($this->validateAddress($address)) {**

**$addresses[] = array(**

**'name' => '',**

**'address' => $address**

**);**

**}**

**} else {**

**list($name, $email) = explode('<', $address);**

**$email = trim(str\_replace('>', '', $email));**

**if ($this->validateAddress($email)) {**

**$addresses[] = array(**

**'name' => trim(str\_replace(array('"', "'"), '', $name)),**

**'address' => $email**

**);**

**}**

**}**

**}**

**}**

**return $addresses;**

**}**

**/\*\***

**\* Set the From and FromName properties.**

**\* @param string $address**

**\* @param string $name**

**\* @param boolean $auto Whether to also set the Sender address, defaults to true**

**\* @throws phpmailerException**

**\* @return boolean**

**\*/**

**public function setFrom($address, $name = '', $auto = true)**

**{**

**$address = trim($address);**

**$name = trim(preg\_replace('/[\r\n]+/', '', $name)); //Strip breaks and trim**

**// Don't validate now addresses with IDN. Will be done in send().**

**if (($pos = strrpos($address, '@')) === false or**

**(!$this->has8bitChars(substr($address, ++$pos)) or !$this->idnSupported()) and**

**!$this->validateAddress($address)) {**

**$error\_message = $this->lang('invalid\_address') . " (setFrom) $address";**

**$this->setError($error\_message);**

**$this->edebug($error\_message);**

**if ($this->exceptions) {**

**throw new phpmailerException($error\_message);**

**}**

**return false;**

**}**

**$this->From = $address;**

**$this->FromName = $name;**

**if ($auto) {**

**if (empty($this->Sender)) {**

**$this->Sender = $address;**

**}**

**}**

**return true;**

**}**

**/\*\***

**\* Return the Message-ID header of the last email.**

**\* Technically this is the value from the last time the headers were created,**

**\* but it's also the message ID of the last sent message except in**

**\* pathological cases.**

**\* @return string**

**\*/**

**public function getLastMessageID()**

**{**

**return $this->lastMessageID;**

**}**

**/\*\***

**\* Check that a string looks like an email address.**

**\* @param string $address The email address to check**

**\* @param string|callable $patternselect A selector for the validation pattern to use :**

**\* \* `auto` Pick best pattern automatically;**

**\* \* `pcre8` Use the squiloople.com pattern, requires PCRE > 8.0, PHP >= 5.3.2, 5.2.14;**

**\* \* `pcre` Use old PCRE implementation;**

**\* \* `php` Use PHP built-in FILTER\_VALIDATE\_EMAIL;**

**\* \* `html5` Use the pattern given by the HTML5 spec for 'email' type form input elements.**

**\* \* `noregex` Don't use a regex: super fast, really dumb.**

**\* Alternatively you may pass in a callable to inject your own validator, for example:**

**\* PHPMailer::validateAddress('user@example.com', function($address) {**

**\* return (strpos($address, '@') !== false);**

**\* });**

**\* You can also set the PHPMailer::$validator static to a callable, allowing built-in methods to use your validator.**

**\* @return boolean**

**\* @static**

**\* @access public**

**\*/**

**public static function validateAddress($address, $patternselect = null)**

**{**

**if (is\_null($patternselect)) {**

**$patternselect = self::$validator;**

**}**

**if (is\_callable($patternselect)) {**

**return call\_user\_func($patternselect, $address);**

**}**

**//Reject line breaks in addresses; it's valid RFC5322, but not RFC5321**

**if (strpos($address, "\n") !== false or strpos($address, "\r") !== false) {**

**return false;**

**}**

**if (!$patternselect or $patternselect == 'auto') {**

**//Check this constant first so it works when extension\_loaded() is disabled by safe mode**

**//Constant was added in PHP 5.2.4**

**if (defined('PCRE\_VERSION')) {**

**//This pattern can get stuck in a recursive loop in PCRE <= 8.0.2**

**if (version\_compare(PCRE\_VERSION, '8.0.3') >= 0) {**

**$patternselect = 'pcre8';**

**} else {**

**$patternselect = 'pcre';**

**}**

**} elseif (function\_exists('extension\_loaded') and extension\_loaded('pcre')) {**

**//Fall back to older PCRE**

**$patternselect = 'pcre';**

**} else {**

**//Filter\_var appeared in PHP 5.2.0 and does not require the PCRE extension**

**if (version\_compare(PHP\_VERSION, '5.2.0') >= 0) {**

**$patternselect = 'php';**

**} else {**

**$patternselect = 'noregex';**

**}**

**}**

**}**

**switch ($patternselect) {**

**case 'pcre8':**

**/\*\***

**\* Uses the same RFC5322 regex on which FILTER\_VALIDATE\_EMAIL is based, but allows dotless domains.**

**\* @link http://squiloople.com/2009/12/20/email-address-validation/**

**\* @copyright 2009-2010 Michael Rushton**

**\* Feel free to use and redistribute this code. But please keep this copyright notice.**

**\*/**

**return (boolean)preg\_match(**

**'/^(?!(?>(?1)"?(?>\\\[ -~]|[^"])"?(?1)){255,})(?!(?>(?1)"?(?>\\\[ -~]|[^"])"?(?1)){65,}@)' .**

**'((?>(?>(?>((?>(?>(?>\x0D\x0A)?[\t ])+|(?>[\t ]\*\x0D\x0A)?[\t ]+)?)(\((?>(?2)' .**

**'(?>[\x01-\x08\x0B\x0C\x0E-\'\*-\[\]-\x7F]|\\\[\x00-\x7F]|(?3)))\*(?2)\)))+(?2))|(?2))?)' .**

**'([!#-\'\*+\/-9=?^-~-]+|"(?>(?2)(?>[\x01-\x08\x0B\x0C\x0E-!#-\[\]-\x7F]|\\\[\x00-\x7F]))\*' .**

**'(?2)")(?>(?1)\.(?1)(?4))\*(?1)@(?!(?1)[a-z0-9-]{64,})(?1)(?>([a-z0-9](?>[a-z0-9-]\*[a-z0-9])?)' .**

**'(?>(?1)\.(?!(?1)[a-z0-9-]{64,})(?1)(?5)){0,126}|\[(?:(?>IPv6:(?>([a-f0-9]{1,4})(?>:(?6)){7}' .**

**'|(?!(?:.\*[a-f0-9][:\]]){8,})((?6)(?>:(?6)){0,6})?::(?7)?))|(?>(?>IPv6:(?>(?6)(?>:(?6)){5}:' .**

**'|(?!(?:.\*[a-f0-9]:){6,})(?8)?::(?>((?6)(?>:(?6)){0,4}):)?))?(25[0-5]|2[0-4][0-9]|1[0-9]{2}' .**

**'|[1-9]?[0-9])(?>\.(?9)){3}))\])(?1)$/isD',**

**$address**

**);**

**case 'pcre':**

**//An older regex that doesn't need a recent PCRE**

**return (boolean)preg\_match(**

**'/^(?!(?>"?(?>\\\[ -~]|[^"])"?){255,})(?!(?>"?(?>\\\[ -~]|[^"])"?){65,}@)(?>' .**

**'[!#-\'\*+\/-9=?^-~-]+|"(?>(?>[\x01-\x08\x0B\x0C\x0E-!#-\[\]-\x7F]|\\\[\x00-\xFF]))\*")' .**

**'(?>\.(?>[!#-\'\*+\/-9=?^-~-]+|"(?>(?>[\x01-\x08\x0B\x0C\x0E-!#-\[\]-\x7F]|\\\[\x00-\xFF]))\*"))\*' .**

**'@(?>(?![a-z0-9-]{64,})(?>[a-z0-9](?>[a-z0-9-]\*[a-z0-9])?)(?>\.(?![a-z0-9-]{64,})' .**

**'(?>[a-z0-9](?>[a-z0-9-]\*[a-z0-9])?)){0,126}|\[(?:(?>IPv6:(?>(?>[a-f0-9]{1,4})(?>:' .**

**'[a-f0-9]{1,4}){7}|(?!(?:.\*[a-f0-9][:\]]){8,})(?>[a-f0-9]{1,4}(?>:[a-f0-9]{1,4}){0,6})?' .**

**'::(?>[a-f0-9]{1,4}(?>:[a-f0-9]{1,4}){0,6})?))|(?>(?>IPv6:(?>[a-f0-9]{1,4}(?>:' .**

**'[a-f0-9]{1,4}){5}:|(?!(?:.\*[a-f0-9]:){6,})(?>[a-f0-9]{1,4}(?>:[a-f0-9]{1,4}){0,4})?' .**

**'::(?>(?:[a-f0-9]{1,4}(?>:[a-f0-9]{1,4}){0,4}):)?))?(?>25[0-5]|2[0-4][0-9]|1[0-9]{2}' .**

**'|[1-9]?[0-9])(?>\.(?>25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){3}))\])$/isD',**

**$address**

**);**

**case 'html5':**

**/\*\***

**\* This is the pattern used in the HTML5 spec for validation of 'email' type form input elements.**

**\* @link http://www.whatwg.org/specs/web-apps/current-work/#e-mail-state-(type=email)**

**\*/**

**return (boolean)preg\_match(**

**'/^[a-zA-Z0-9.!#$%&\'\*+\/=?^\_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}' .**

**'[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)\*$/sD',**

**$address**

**);**

**case 'noregex':**

**//No PCRE! Do something \_very\_ approximate!**

**//Check the address is 3 chars or longer and contains an @ that's not the first or last char**

**return (strlen($address) >= 3**

**and strpos($address, '@') >= 1**

**and strpos($address, '@') != strlen($address) - 1);**

**case 'php':**

**default:**

**return (boolean)filter\_var($address, FILTER\_VALIDATE\_EMAIL);**

**}**

**}**

**/\*\***

**\* Tells whether IDNs (Internationalized Domain Names) are supported or not. This requires the**

**\* "intl" and "mbstring" PHP extensions.**

**\* @return bool "true" if required functions for IDN support are present**

**\*/**

**public function idnSupported()**

**{**

**// @TODO: Write our own "idn\_to\_ascii" function for PHP <= 5.2.**

**return function\_exists('idn\_to\_ascii') and function\_exists('mb\_convert\_encoding');**

**}**

**/\*\***

**\* Converts IDN in given email address to its ASCII form, also known as punycode, if possible.**

**\* Important: Address must be passed in same encoding as currently set in PHPMailer::$CharSet.**

**\* This function silently returns unmodified address if:**

**\* - No conversion is necessary (i.e. domain name is not an IDN, or is already in ASCII form)**

**\* - Conversion to punycode is impossible (e.g. required PHP functions are not available)**

**\* or fails for any reason (e.g. domain has characters not allowed in an IDN)**

**\* @see PHPMailer::$CharSet**

**\* @param string $address The email address to convert**

**\* @return string The encoded address in ASCII form**

**\*/**

**public function punyencodeAddress($address)**

**{**

**// Verify we have required functions, CharSet, and at-sign.**

**if ($this->idnSupported() and**

**!empty($this->CharSet) and**

**($pos = strrpos($address, '@')) !== false) {**

**$domain = substr($address, ++$pos);**

**// Verify CharSet string is a valid one, and domain properly encoded in this CharSet.**

**if ($this->has8bitChars($domain) and @mb\_check\_encoding($domain, $this->CharSet)) {**

**$domain = mb\_convert\_encoding($domain, 'UTF-8', $this->CharSet);**

**if (($punycode = defined('INTL\_IDNA\_VARIANT\_UTS46') ?**

**idn\_to\_ascii($domain, 0, INTL\_IDNA\_VARIANT\_UTS46) :**

**idn\_to\_ascii($domain)) !== false) {**

**return substr($address, 0, $pos) . $punycode;**

**}**

**}**

**}**

**return $address;**

**}**

**/\*\***

**\* Create a message and send it.**

**\* Uses the sending method specified by $Mailer.**

**\* @throws phpmailerException**

**\* @return boolean false on error - See the ErrorInfo property for details of the error.**

**\*/**

**public function send()**

**{**

**try {**

**if (!$this->preSend()) {**

**return false;**

**}**

**return $this->postSend();**

**} catch (phpmailerException $exc) {**

**$this->mailHeader = '';**

**$this->setError($exc->getMessage());**

**if ($this->exceptions) {**

**throw $exc;**

**}**

**return false;**

**}**

**}**

**/\*\***

**\* Prepare a message for sending.**

**\* @throws phpmailerException**

**\* @return boolean**

**\*/**

**public function preSend()**

**{**

**try {**

**$this->error\_count = 0; // Reset errors**

**$this->mailHeader = '';**

**// Dequeue recipient and Reply-To addresses with IDN**

**foreach (array\_merge($this->RecipientsQueue, $this->ReplyToQueue) as $params) {**

**$params[1] = $this->punyencodeAddress($params[1]);**

**call\_user\_func\_array(array($this, 'addAnAddress'), $params);**

**}**

**if ((count($this->to) + count($this->cc) + count($this->bcc)) < 1) {**

**throw new phpmailerException($this->lang('provide\_address'), self::STOP\_CRITICAL);**

**}**

**// Validate From, Sender, and ConfirmReadingTo addresses**

**foreach (array('From', 'Sender', 'ConfirmReadingTo') as $address\_kind) {**

**$this->$address\_kind = trim($this->$address\_kind);**

**if (empty($this->$address\_kind)) {**

**continue;**

**}**

**$this->$address\_kind = $this->punyencodeAddress($this->$address\_kind);**

**if (!$this->validateAddress($this->$address\_kind)) {**

**$error\_message = $this->lang('invalid\_address') . ' (punyEncode) ' . $this->$address\_kind;**

**$this->setError($error\_message);**

**$this->edebug($error\_message);**

**if ($this->exceptions) {**

**throw new phpmailerException($error\_message);**

**}**

**return false;**

**}**

**}**

**// Set whether the message is multipart/alternative**

**if ($this->alternativeExists()) {**

**$this->ContentType = 'multipart/alternative';**

**}**

**$this->setMessageType();**

**// Refuse to send an empty message unless we are specifically allowing it**

**if (!$this->AllowEmpty and empty($this->Body)) {**

**throw new phpmailerException($this->lang('empty\_message'), self::STOP\_CRITICAL);**

**}**

**// Create body before headers in case body makes changes to headers (e.g. altering transfer encoding)**

**$this->MIMEHeader = '';**

**$this->MIMEBody = $this->createBody();**

**// createBody may have added some headers, so retain them**

**$tempheaders = $this->MIMEHeader;**

**$this->MIMEHeader = $this->createHeader();**

**$this->MIMEHeader .= $tempheaders;**

**// To capture the complete message when using mail(), create**

**// an extra header list which createHeader() doesn't fold in**

**if ($this->Mailer == 'mail') {**

**if (count($this->to) > 0) {**

**$this->mailHeader .= $this->addrAppend('To', $this->to);**

**} else {**

**$this->mailHeader .= $this->headerLine('To', 'undisclosed-recipients:;');**

**}**

**$this->mailHeader .= $this->headerLine(**

**'Subject',**

**$this->encodeHeader($this->secureHeader(trim($this->Subject)))**

**);**

**}**

**// Sign with DKIM if enabled**

**if (!empty($this->DKIM\_domain)**

**and !empty($this->DKIM\_selector)**

**and (!empty($this->DKIM\_private\_string)**

**or (!empty($this->DKIM\_private)**

**and self::isPermittedPath($this->DKIM\_private)**

**and file\_exists($this->DKIM\_private)**

**)**

**)**

**) {**

**$header\_dkim = $this->DKIM\_Add(**

**$this->MIMEHeader . $this->mailHeader,**

**$this->encodeHeader($this->secureHeader($this->Subject)),**

**$this->MIMEBody**

**);**

**$this->MIMEHeader = rtrim($this->MIMEHeader, "\r\n ") . self::CRLF .**

**str\_replace("\r\n", "\n", $header\_dkim) . self::CRLF;**

**}**

**return true;**

**} catch (phpmailerException $exc) {**

**$this->setError($exc->getMessage());**

**if ($this->exceptions) {**

**throw $exc;**

**}**

**return false;**

**}**

**}**

**/\*\***

**\* Actually send a message.**

**\* Send the email via the selected mechanism**

**\* @throws phpmailerException**

**\* @return boolean**

**\*/**

**public function postSend()**

**{**

**try {**

**// Choose the mailer and send through it**

**switch ($this->Mailer) {**

**case 'sendmail':**

**case 'qmail':**

**return $this->sendmailSend($this->MIMEHeader, $this->MIMEBody);**

**case 'smtp':**

**return $this->smtpSend($this->MIMEHeader, $this->MIMEBody);**

**case 'mail':**

**return $this->mailSend($this->MIMEHeader, $this->MIMEBody);**

**default:**

**$sendMethod = $this->Mailer.'Send';**

**if (method\_exists($this, $sendMethod)) {**

**return $this->$sendMethod($this->MIMEHeader, $this->MIMEBody);**

**}**

**return $this->mailSend($this->MIMEHeader, $this->MIMEBody);**

**}**

**} catch (phpmailerException $exc) {**

**$this->setError($exc->getMessage());**

**$this->edebug($exc->getMessage());**

**if ($this->exceptions) {**

**throw $exc;**

**}**

**}**

**return false;**

**}**

**/\*\***

**\* Send mail using the $Sendmail program.**

**\* @param string $header The message headers**

**\* @param string $body The message body**

**\* @see PHPMailer::$Sendmail**

**\* @throws phpmailerException**

**\* @access protected**

**\* @return boolean**

**\*/**

**protected function sendmailSend($header, $body)**

**{**

**// CVE-2016-10033, CVE-2016-10045: Don't pass -f if characters will be escaped.**

**if (!empty($this->Sender) and self::isShellSafe($this->Sender)) {**

**if ($this->Mailer == 'qmail') {**

**$sendmailFmt = '%s -f%s';**

**} else {**

**$sendmailFmt = '%s -oi -f%s -t';**

**}**

**} else {**

**if ($this->Mailer == 'qmail') {**

**$sendmailFmt = '%s';**

**} else {**

**$sendmailFmt = '%s -oi -t';**

**}**

**}**

**// TODO: If possible, this should be changed to escapeshellarg. Needs thorough testing.**

**$sendmail = sprintf($sendmailFmt, escapeshellcmd($this->Sendmail), $this->Sender);**

**if ($this->SingleTo) {**

**foreach ($this->SingleToArray as $toAddr) {**

**if (!@$mail = popen($sendmail, 'w')) {**

**throw new phpmailerException($this->lang('execute') . $this->Sendmail, self::STOP\_CRITICAL);**

**}**

**fputs($mail, 'To: ' . $toAddr . "\n");**

**fputs($mail, $header);**

**fputs($mail, $body);**

**$result = pclose($mail);**

**$this->doCallback(**

**($result == 0),**

**array($toAddr),**

**$this->cc,**

**$this->bcc,**

**$this->Subject,**

**$body,**

**$this->From**

**);**

**if ($result != 0) {**

**throw new phpmailerException($this->lang('execute') . $this->Sendmail, self::STOP\_CRITICAL);**

**}**

**}**

**} else {**

**if (!@$mail = popen($sendmail, 'w')) {**

**throw new phpmailerException($this->lang('execute') . $this->Sendmail, self::STOP\_CRITICAL);**

**}**

**fputs($mail, $header);**

**fputs($mail, $body);**

**$result = pclose($mail);**

**$this->doCallback(**

**($result == 0),**

**$this->to,**

**$this->cc,**

**$this->bcc,**

**$this->Subject,**

**$body,**

**$this->From**

**);**

**if ($result != 0) {**

**throw new phpmailerException($this->lang('execute') . $this->Sendmail, self::STOP\_CRITICAL);**

**}**

**}**

**return true;**

**}**

**/\*\***

**\* Fix CVE-2016-10033 and CVE-2016-10045 by disallowing potentially unsafe shell characters.**

**\***

**\* Note that escapeshellarg and escapeshellcmd are inadequate for our purposes, especially on Windows.**

**\* @param string $string The string to be validated**

**\* @see https://github.com/PHPMailer/PHPMailer/issues/924 CVE-2016-10045 bug report**

**\* @access protected**

**\* @return boolean**

**\*/**

**protected static function isShellSafe($string)**

**{**

**// Future-proof**

**if (escapeshellcmd($string) !== $string**

**or !in\_array(escapeshellarg($string), array("'$string'", "\"$string\""))**

**) {**

**return false;**

**}**

**$length = strlen($string);**

**for ($i = 0; $i < $length; $i++) {**

**$c = $string[$i];**

**// All other characters have a special meaning in at least one common shell, including = and +.**

**// Full stop (.) has a special meaning in cmd.exe, but its impact should be negligible here.**

**// Note that this does permit non-Latin alphanumeric characters based on the current locale.**

**if (!ctype\_alnum($c) && strpos('@\_-.', $c) === false) {**

**return false;**

**}**

**}**

**return true;**

**}**

**/\*\***

**\* Check whether a file path is of a permitted type.**

**\* Used to reject URLs and phar files from functions that access local file paths,**

**\* such as addAttachment.**

**\* @param string $path A relative or absolute path to a file.**

**\* @return bool**

**\*/**

**protected static function isPermittedPath($path)**

**{**

**return !preg\_match('#^[a-z]+://#i', $path);**

**}**

**/\*\***

**\* Send mail using the PHP mail() function.**

**\* @param string $header The message headers**

**\* @param string $body The message body**

**\* @link http://www.php.net/manual/en/book.mail.php**

**\* @throws phpmailerException**

**\* @access protected**

**\* @return boolean**

**\*/**

**protected function mailSend($header, $body)**

**{**

**$toArr = array();**

**foreach ($this->to as $toaddr) {**

**$toArr[] = $this->addrFormat($toaddr);**

**}**

**$to = implode(', ', $toArr);**

**$params = null;**

**//This sets the SMTP envelope sender which gets turned into a return-path header by the receiver**

**if (!empty($this->Sender) and $this->validateAddress($this->Sender)) {**

**// CVE-2016-10033, CVE-2016-10045: Don't pass -f if characters will be escaped.**

**if (self::isShellSafe($this->Sender)) {**

**$params = sprintf('-f%s', $this->Sender);**

**}**

**}**

**if (!empty($this->Sender) and !ini\_get('safe\_mode') and $this->validateAddress($this->Sender)) {**

**$old\_from = ini\_get('sendmail\_from');**

**ini\_set('sendmail\_from', $this->Sender);**

**}**

**$result = false;**

**if ($this->SingleTo and count($toArr) > 1) {**

**foreach ($toArr as $toAddr) {**

**$result = $this->mailPassthru($toAddr, $this->Subject, $body, $header, $params);**

**$this->doCallback($result, array($toAddr), $this->cc, $this->bcc, $this->Subject, $body, $this->From);**

**}**

**} else {**

**$result = $this->mailPassthru($to, $this->Subject, $body, $header, $params);**

**$this->doCallback($result, $this->to, $this->cc, $this->bcc, $this->Subject, $body, $this->From);**

**}**

**if (isset($old\_from)) {**

**ini\_set('sendmail\_from', $old\_from);**

**}**

**if (!$result) {**

**throw new phpmailerException($this->lang('instantiate'), self::STOP\_CRITICAL);**

**}**

**return true;**

**}**

**/\*\***

**\* Get an instance to use for SMTP operations.**

**\* Override this function to load your own SMTP implementation**

**\* @return SMTP**

**\*/**

**public function getSMTPInstance()**

**{**

**if (!is\_object($this->smtp)) {**

**$this->smtp = new SMTP;**

**}**

**return $this->smtp;**

**}**

**/\*\***

**\* Send mail via SMTP.**

**\* Returns false if there is a bad MAIL FROM, RCPT, or DATA input.**

**\* Uses the PHPMailerSMTP class by default.**

**\* @see PHPMailer::getSMTPInstance() to use a different class.**

**\* @param string $header The message headers**

**\* @param string $body The message body**

**\* @throws phpmailerException**

**\* @uses SMTP**

**\* @access protected**

**\* @return boolean**

**\*/**

**protected function smtpSend($header, $body)**

**{**

**$bad\_rcpt = array();**

**if (!$this->smtpConnect($this->SMTPOptions)) {**

**throw new phpmailerException($this->lang('smtp\_connect\_failed'), self::STOP\_CRITICAL);**

**}**

**if (!empty($this->Sender) and $this->validateAddress($this->Sender)) {**

**$smtp\_from = $this->Sender;**

**} else {**

**$smtp\_from = $this->From;**

**}**

**if (!$this->smtp->mail($smtp\_from)) {**

**$this->setError($this->lang('from\_failed') . $smtp\_from . ' : ' . implode(',', $this->smtp->getError()));**

**throw new phpmailerException($this->ErrorInfo, self::STOP\_CRITICAL);**

**}**

**// Attempt to send to all recipients**

**foreach (array($this->to, $this->cc, $this->bcc) as $togroup) {**

**foreach ($togroup as $to) {**

**if (!$this->smtp->recipient($to[0])) {**

**$error = $this->smtp->getError();**

**$bad\_rcpt[] = array('to' => $to[0], 'error' => $error['detail']);**

**$isSent = false;**

**} else {**

**$isSent = true;**

**}**

**$this->doCallback($isSent, array($to[0]), array(), array(), $this->Subject, $body, $this->From);**

**}**

**}**

**// Only send the DATA command if we have viable recipients**

**if ((count($this->all\_recipients) > count($bad\_rcpt)) and !$this->smtp->data($header . $body)) {**

**throw new phpmailerException($this->lang('data\_not\_accepted'), self::STOP\_CRITICAL);**

**}**

**if ($this->SMTPKeepAlive) {**

**$this->smtp->reset();**

**} else {**

**$this->smtp->quit();**

**$this->smtp->close();**

**}**

**//Create error message for any bad addresses**

**if (count($bad\_rcpt) > 0) {**

**$errstr = '';**

**foreach ($bad\_rcpt as $bad) {**

**$errstr .= $bad['to'] . ': ' . $bad['error'];**

**}**

**throw new phpmailerException(**

**$this->lang('recipients\_failed') . $errstr,**

**self::STOP\_CONTINUE**

**);**

**}**

**return true;**

**}**

**/\*\***

**\* Initiate a connection to an SMTP server.**

**\* Returns false if the operation failed.**

**\* @param array $options An array of options compatible with stream\_context\_create()**

**\* @uses SMTP**

**\* @access public**

**\* @throws phpmailerException**

**\* @return boolean**

**\*/**

**public function smtpConnect($options = null)**

**{**

**if (is\_null($this->smtp)) {**

**$this->smtp = $this->getSMTPInstance();**

**}**

**//If no options are provided, use whatever is set in the instance**

**if (is\_null($options)) {**

**$options = $this->SMTPOptions;**

**}**

**// Already connected?**

**if ($this->smtp->connected()) {**

**return true;**

**}**

**$this->smtp->setTimeout($this->Timeout);**

**$this->smtp->setDebugLevel($this->SMTPDebug);**

**$this->smtp->setDebugOutput($this->Debugoutput);**

**$this->smtp->setVerp($this->do\_verp);**

**$hosts = explode(';', $this->Host);**

**$lastexception = null;**

**foreach ($hosts as $hostentry) {**

**$hostinfo = array();**

**if (!preg\_match(**

**'/^((ssl|tls):\/\/)\*([a-zA-Z0-9\.-]\*|\[[a-fA-F0-9:]+\]):?([0-9]\*)$/',**

**trim($hostentry),**

**$hostinfo**

**)) {**

**// Not a valid host entry**

**$this->edebug('Ignoring invalid host: ' . $hostentry);**

**continue;**

**}**

**// $hostinfo[2]: optional ssl or tls prefix**

**// $hostinfo[3]: the hostname**

**// $hostinfo[4]: optional port number**

**// The host string prefix can temporarily override the current setting for SMTPSecure**

**// If it's not specified, the default value is used**

**$prefix = '';**

**$secure = $this->SMTPSecure;**

**$tls = ($this->SMTPSecure == 'tls');**

**if ('ssl' == $hostinfo[2] or ('' == $hostinfo[2] and 'ssl' == $this->SMTPSecure)) {**

**$prefix = 'ssl://';**

**$tls = false; // Can't have SSL and TLS at the same time**

**$secure = 'ssl';**

**} elseif ($hostinfo[2] == 'tls') {**

**$tls = true;**

**// tls doesn't use a prefix**

**$secure = 'tls';**

**}**

**//Do we need the OpenSSL extension?**

**$sslext = defined('OPENSSL\_ALGO\_SHA1');**

**if ('tls' === $secure or 'ssl' === $secure) {**

**//Check for an OpenSSL constant rather than using extension\_loaded, which is sometimes disabled**

**if (!$sslext) {**

**throw new phpmailerException($this->lang('extension\_missing').'openssl', self::STOP\_CRITICAL);**

**}**

**}**

**$host = $hostinfo[3];**

**$port = $this->Port;**

**$tport = (integer)$hostinfo[4];**

**if ($tport > 0 and $tport < 65536) {**

**$port = $tport;**

**}**

**if ($this->smtp->connect($prefix . $host, $port, $this->Timeout, $options)) {**

**try {**

**if ($this->Helo) {**

**$hello = $this->Helo;**

**} else {**

**$hello = $this->serverHostname();**

**}**

**$this->smtp->hello($hello);**

**//Automatically enable TLS encryption if:**

**// \* it's not disabled**

**// \* we have openssl extension**

**// \* we are not already using SSL**

**// \* the server offers STARTTLS**

**if ($this->SMTPAutoTLS and $sslext and $secure != 'ssl' and $this->smtp->getServerExt('STARTTLS')) {**

**$tls = true;**

**}**

**if ($tls) {**

**if (!$this->smtp->startTLS()) {**

**throw new phpmailerException($this->lang('connect\_host'));**

**}**

**// We must resend EHLO after TLS negotiation**

**$this->smtp->hello($hello);**

**}**

**if ($this->SMTPAuth) {**

**if (!$this->smtp->authenticate(**

**$this->Username,**

**$this->Password,**

**$this->AuthType,**

**$this->Realm,**

**$this->Workstation**

**)**

**) {**

**throw new phpmailerException($this->lang('authenticate'));**

**}**

**}**

**return true;**

**} catch (phpmailerException $exc) {**

**$lastexception = $exc;**

**$this->edebug($exc->getMessage());**

**// We must have connected, but then failed TLS or Auth, so close connection nicely**

**$this->smtp->quit();**

**}**

**}**

**}**

**// If we get here, all connection attempts have failed, so close connection hard**

**$this->smtp->close();**

**// As we've caught all exceptions, just report whatever the last one was**

**if ($this->exceptions and !is\_null($lastexception)) {**

**throw $lastexception;**

**}**

**return false;**

**}**

**/\*\***

**\* Close the active SMTP session if one exists.**

**\* @return void**

**\*/**

**public function smtpClose()**

**{**

**if (is\_a($this->smtp, 'SMTP')) {**

**if ($this->smtp->connected()) {**

**$this->smtp->quit();**

**$this->smtp->close();**

**}**

**}**

**}**

**/\*\***

**\* Set the language for error messages.**

**\* Returns false if it cannot load the language file.**

**\* The default language is English.**

**\* @param string $langcode ISO 639-1 2-character language code (e.g. French is "fr")**

**\* @param string $lang\_path Path to the language file directory, with trailing separator (slash)**

**\* @return boolean**

**\* @access public**

**\*/**

**public function setLanguage($langcode = 'en', $lang\_path = '')**

**{**

**// Backwards compatibility for renamed language codes**

**$renamed\_langcodes = array(**

**'br' => 'pt\_br',**

**'cz' => 'cs',**

**'dk' => 'da',**

**'no' => 'nb',**

**'se' => 'sv',**

**'sr' => 'rs'**

**);**

**if (isset($renamed\_langcodes[$langcode])) {**

**$langcode = $renamed\_langcodes[$langcode];**

**}**

**// Define full set of translatable strings in English**

**$PHPMAILER\_LANG = array(**

**'authenticate' => 'SMTP Error: Could not authenticate.',**

**'connect\_host' => 'SMTP Error: Could not connect to SMTP host.',**

**'data\_not\_accepted' => 'SMTP Error: data not accepted.',**

**'empty\_message' => 'Message body empty',**

**'encoding' => 'Unknown encoding: ',**

**'execute' => 'Could not execute: ',**

**'file\_access' => 'Could not access file: ',**

**'file\_open' => 'File Error: Could not open file: ',**

**'from\_failed' => 'The following From address failed: ',**

**'instantiate' => 'Could not instantiate mail function.',**

**'invalid\_address' => 'Invalid address: ',**

**'mailer\_not\_supported' => ' mailer is not supported.',**

**'provide\_address' => 'You must provide at least one recipient email address.',**

**'recipients\_failed' => 'SMTP Error: The following recipients failed: ',**

**'signing' => 'Signing Error: ',**

**'smtp\_connect\_failed' => 'SMTP connect() failed.',**

**'smtp\_error' => 'SMTP server error: ',**

**'variable\_set' => 'Cannot set or reset variable: ',**

**'extension\_missing' => 'Extension missing: '**

**);**

**if (empty($lang\_path)) {**

**// Calculate an absolute path so it can work if CWD is not here**

**$lang\_path = dirname(\_\_FILE\_\_). DIRECTORY\_SEPARATOR . 'language'. DIRECTORY\_SEPARATOR;**

**}**

**//Validate $langcode**

**if (!preg\_match('/^[a-z]{2}(?:\_[a-zA-Z]{2})?$/', $langcode)) {**

**$langcode = 'en';**

**}**

**$foundlang = true;**

**$lang\_file = $lang\_path . 'phpmailer.lang-' . $langcode . '.php';**

**// There is no English translation file**

**if ($langcode != 'en') {**

**// Make sure language file path is readable**

**if (!self::isPermittedPath($lang\_file) or !is\_readable($lang\_file)) {**

**$foundlang = false;**

**} else {**

**// Overwrite language-specific strings.**

**// This way we'll never have missing translation keys.**

**$foundlang = include $lang\_file;**

**}**

**}**

**$this->language = $PHPMAILER\_LANG;**

**return (boolean)$foundlang; // Returns false if language not found**

**}**

**/\*\***

**\* Get the array of strings for the current language.**

**\* @return array**

**\*/**

**public function getTranslations()**

**{**

**return $this->language;**

**}**

**/\*\***

**\* Create recipient headers.**

**\* @access public**

**\* @param string $type**

**\* @param array $addr An array of recipient,**

**\* where each recipient is a 2-element indexed array with element 0 containing an address**

**\* and element 1 containing a name, like:**

**\* array(array('joe@example.com', 'Joe User'), array('zoe@example.com', 'Zoe User'))**

**\* @return string**

**\*/**

**public function addrAppend($type, $addr)**

**{**

**$addresses = array();**

**foreach ($addr as $address) {**

**$addresses[] = $this->addrFormat($address);**

**}**

**return $type . ': ' . implode(', ', $addresses) . $this->LE;**

**}**

**/\*\***

**\* Format an address for use in a message header.**

**\* @access public**

**\* @param array $addr A 2-element indexed array, element 0 containing an address, element 1 containing a name**

**\* like array('joe@example.com', 'Joe User')**

**\* @return string**

**\*/**

**public function addrFormat($addr)**

**{**

**if (empty($addr[1])) { // No name provided**

**return $this->secureHeader($addr[0]);**

**} else {**

**return $this->encodeHeader($this->secureHeader($addr[1]), 'phrase') . ' <' . $this->secureHeader(**

**$addr[0]**

**) . '>';**

**}**

**}**

**/\*\***

**\* Word-wrap message.**

**\* For use with mailers that do not automatically perform wrapping**

**\* and for quoted-printable encoded messages.**

**\* Original written by philippe.**

**\* @param string $message The message to wrap**

**\* @param integer $length The line length to wrap to**

**\* @param boolean $qp\_mode Whether to run in Quoted-Printable mode**

**\* @access public**

**\* @return string**

**\*/**

**public function wrapText($message, $length, $qp\_mode = false)**

**{**

**if ($qp\_mode) {**

**$soft\_break = sprintf(' =%s', $this->LE);**

**} else {**

**$soft\_break = $this->LE;**

**}**

**// If utf-8 encoding is used, we will need to make sure we don't**

**// split multibyte characters when we wrap**

**$is\_utf8 = (strtolower($this->CharSet) == 'utf-8');**

**$lelen = strlen($this->LE);**

**$crlflen = strlen(self::CRLF);**

**$message = $this->fixEOL($message);**

**//Remove a trailing line break**

**if (substr($message, -$lelen) == $this->LE) {**

**$message = substr($message, 0, -$lelen);**

**}**

**//Split message into lines**

**$lines = explode($this->LE, $message);**

**//Message will be rebuilt in here**

**$message = '';**

**foreach ($lines as $line) {**

**$words = explode(' ', $line);**

**$buf = '';**

**$firstword = true;**

**foreach ($words as $word) {**

**if ($qp\_mode and (strlen($word) > $length)) {**

**$space\_left = $length - strlen($buf) - $crlflen;**

**if (!$firstword) {**

**if ($space\_left > 20) {**

**$len = $space\_left;**

**if ($is\_utf8) {**

**$len = $this->utf8CharBoundary($word, $len);**

**} elseif (substr($word, $len - 1, 1) == '=') {**

**$len--;**

**} elseif (substr($word, $len - 2, 1) == '=') {**

**$len -= 2;**

**}**

**$part = substr($word, 0, $len);**

**$word = substr($word, $len);**

**$buf .= ' ' . $part;**

**$message .= $buf . sprintf('=%s', self::CRLF);**

**} else {**

**$message .= $buf . $soft\_break;**

**}**

**$buf = '';**

**}**

**while (strlen($word) > 0) {**

**if ($length <= 0) {**

**break;**

**}**

**$len = $length;**

**if ($is\_utf8) {**

**$len = $this->utf8CharBoundary($word, $len);**

**} elseif (substr($word, $len - 1, 1) == '=') {**

**$len--;**

**} elseif (substr($word, $len - 2, 1) == '=') {**

**$len -= 2;**

**}**

**$part = substr($word, 0, $len);**

**$word = substr($word, $len);**

**if (strlen($word) > 0) {**

**$message .= $part . sprintf('=%s', self::CRLF);**

**} else {**

**$buf = $part;**

**}**

**}**

**} else {**

**$buf\_o = $buf;**

**if (!$firstword) {**

**$buf .= ' ';**

**}**

**$buf .= $word;**

**if (strlen($buf) > $length and $buf\_o != '') {**

**$message .= $buf\_o . $soft\_break;**

**$buf = $word;**

**}**

**}**

**$firstword = false;**

**}**

**$message .= $buf . self::CRLF;**

**}**

**return $message;**

**}**

**/\*\***

**\* Find the last character boundary prior to $maxLength in a utf-8**

**\* quoted-printable encoded string.**

**\* Original written by Colin Brown.**

**\* @access public**

**\* @param string $encodedText utf-8 QP text**

**\* @param integer $maxLength Find the last character boundary prior to this length**

**\* @return integer**

**\*/**

**public function utf8CharBoundary($encodedText, $maxLength)**

**{**

**$foundSplitPos = false;**

**$lookBack = 3;**

**while (!$foundSplitPos) {**

**$lastChunk = substr($encodedText, $maxLength - $lookBack, $lookBack);**

**$encodedCharPos = strpos($lastChunk, '=');**

**if (false !== $encodedCharPos) {**

**// Found start of encoded character byte within $lookBack block.**

**// Check the encoded byte value (the 2 chars after the '=')**

**$hex = substr($encodedText, $maxLength - $lookBack + $encodedCharPos + 1, 2);**

**$dec = hexdec($hex);**

**if ($dec < 128) {**

**// Single byte character.**

**// If the encoded char was found at pos 0, it will fit**

**// otherwise reduce maxLength to start of the encoded char**

**if ($encodedCharPos > 0) {**

**$maxLength = $maxLength - ($lookBack - $encodedCharPos);**

**}**

**$foundSplitPos = true;**

**} elseif ($dec >= 192) {**

**// First byte of a multi byte character**

**// Reduce maxLength to split at start of character**

**$maxLength = $maxLength - ($lookBack - $encodedCharPos);**

**$foundSplitPos = true;**

**} elseif ($dec < 192) {**

**// Middle byte of a multi byte character, look further back**

**$lookBack += 3;**

**}**

**} else {**

**// No encoded character found**

**$foundSplitPos = true;**

**}**

**}**

**return $maxLength;**

**}**

**/\*\***

**\* Apply word wrapping to the message body.**

**\* Wraps the message body to the number of chars set in the WordWrap property.**

**\* You should only do this to plain-text bodies as wrapping HTML tags may break them.**

**\* This is called automatically by createBody(), so you don't need to call it yourself.**

**\* @access public**

**\* @return void**

**\*/**

**public function setWordWrap()**

**{**

**if ($this->WordWrap < 1) {**

**return;**

**}**

**switch ($this->message\_type) {**

**case 'alt':**

**case 'alt\_inline':**

**case 'alt\_attach':**

**case 'alt\_inline\_attach':**

**$this->AltBody = $this->wrapText($this->AltBody, $this->WordWrap);**

**break;**

**default:**

**$this->Body = $this->wrapText($this->Body, $this->WordWrap);**

**break;**

**}**

**}**

**/\*\***

**\* Assemble message headers.**

**\* @access public**

**\* @return string The assembled headers**

**\*/**

**public function createHeader()**

**{**

**$result = '';**

**$result .= $this->headerLine('Date', $this->MessageDate == '' ? self::rfcDate() : $this->MessageDate);**

**// To be created automatically by mail()**

**if ($this->SingleTo) {**

**if ($this->Mailer != 'mail') {**

**foreach ($this->to as $toaddr) {**

**$this->SingleToArray[] = $this->addrFormat($toaddr);**

**}**

**}**

**} else {**

**if (count($this->to) > 0) {**

**if ($this->Mailer != 'mail') {**

**$result .= $this->addrAppend('To', $this->to);**

**}**

**} elseif (count($this->cc) == 0) {**

**$result .= $this->headerLine('To', 'undisclosed-recipients:;');**

**}**

**}**

**$result .= $this->addrAppend('From', array(array(trim($this->From), $this->FromName)));**

**// sendmail and mail() extract Cc from the header before sending**

**if (count($this->cc) > 0) {**

**$result .= $this->addrAppend('Cc', $this->cc);**

**}**

**// sendmail and mail() extract Bcc from the header before sending**

**if ((**

**$this->Mailer == 'sendmail' or $this->Mailer == 'qmail' or $this->Mailer == 'mail'**

**)**

**and count($this->bcc) > 0**

**) {**

**$result .= $this->addrAppend('Bcc', $this->bcc);**

**}**

**if (count($this->ReplyTo) > 0) {**

**$result .= $this->addrAppend('Reply-To', $this->ReplyTo);**

**}**

**// mail() sets the subject itself**

**if ($this->Mailer != 'mail') {**

**$result .= $this->headerLine('Subject', $this->encodeHeader($this->secureHeader($this->Subject)));**

**}**

**// Only allow a custom message ID if it conforms to RFC 5322 section 3.6.4**

**// https://tools.ietf.org/html/rfc5322#section-3.6.4**

**if ('' != $this->MessageID and preg\_match('/^<.\*@.\*>$/', $this->MessageID)) {**

**$this->lastMessageID = $this->MessageID;**

**} else {**

**$this->lastMessageID = sprintf('<%s@%s>', $this->uniqueid, $this->serverHostname());**

**}**

**$result .= $this->headerLine('Message-ID', $this->lastMessageID);**

**if (!is\_null($this->Priority)) {**

**$result .= $this->headerLine('X-Priority', $this->Priority);**

**}**

**if ($this->XMailer == '') {**

**$result .= $this->headerLine(**

**'X-Mailer',**

**'PHPMailer ' . $this->Version . ' (https://github.com/PHPMailer/PHPMailer)'**

**);**

**} else {**

**$myXmailer = trim($this->XMailer);**

**if ($myXmailer) {**

**$result .= $this->headerLine('X-Mailer', $myXmailer);**

**}**

**}**

**if ($this->ConfirmReadingTo != '') {**

**$result .= $this->headerLine('Disposition-Notification-To', '<' . $this->ConfirmReadingTo . '>');**

**}**

**// Add custom headers**

**foreach ($this->CustomHeader as $header) {**

**$result .= $this->headerLine(**

**trim($header[0]),**

**$this->encodeHeader(trim($header[1]))**

**);**

**}**

**if (!$this->sign\_key\_file) {**

**$result .= $this->headerLine('MIME-Version', '1.0');**

**$result .= $this->getMailMIME();**

**}**

**return $result;**

**}**

**/\*\***

**\* Get the message MIME type headers.**

**\* @access public**

**\* @return string**

**\*/**

**public function getMailMIME()**

**{**

**$result = '';**

**$ismultipart = true;**

**switch ($this->message\_type) {**

**case 'inline':**

**$result .= $this->headerLine('Content-Type', 'multipart/related;');**

**$result .= $this->textLine("\tboundary=\"" . $this->boundary[1] . '"');**

**break;**

**case 'attach':**

**case 'inline\_attach':**

**case 'alt\_attach':**

**case 'alt\_inline\_attach':**

**$result .= $this->headerLine('Content-Type', 'multipart/mixed;');**

**$result .= $this->textLine("\tboundary=\"" . $this->boundary[1] . '"');**

**break;**

**case 'alt':**

**case 'alt\_inline':**

**$result .= $this->headerLine('Content-Type', 'multipart/alternative;');**

**$result .= $this->textLine("\tboundary=\"" . $this->boundary[1] . '"');**

**break;**

**default:**

**// Catches case 'plain': and case '':**

**$result .= $this->textLine('Content-Type: ' . $this->ContentType . '; charset=' . $this->CharSet);**

**$ismultipart = false;**

**break;**

**}**

**// RFC1341 part 5 says 7bit is assumed if not specified**

**if ($this->Encoding != '7bit') {**

**// RFC 2045 section 6.4 says multipart MIME parts may only use 7bit, 8bit or binary CTE**

**if ($ismultipart) {**

**if ($this->Encoding == '8bit') {**

**$result .= $this->headerLine('Content-Transfer-Encoding', '8bit');**

**}**

**// The only remaining alternatives are quoted-printable and base64, which are both 7bit compatible**

**} else {**

**$result .= $this->headerLine('Content-Transfer-Encoding', $this->Encoding);**

**}**

**}**

**if ($this->Mailer != 'mail') {**

**$result .= $this->LE;**

**}**

**return $result;**

**}**

**/\*\***

**\* Returns the whole MIME message.**

**\* Includes complete headers and body.**

**\* Only valid post preSend().**

**\* @see PHPMailer::preSend()**

**\* @access public**

**\* @return string**

**\*/**

**public function getSentMIMEMessage()**

**{**

**return rtrim($this->MIMEHeader . $this->mailHeader, "\n\r") . self::CRLF . self::CRLF . $this->MIMEBody;**

**}**

**/\*\***

**\* Create unique ID**

**\* @return string**

**\*/**

**protected function generateId() {**

**return md5(uniqid(time()));**

**}**

**/\*\***

**\* Assemble the message body.**

**\* Returns an empty string on failure.**

**\* @access public**

**\* @throws phpmailerException**

**\* @return string The assembled message body**

**\*/**

**public function createBody()**

**{**

**$body = '';**

**//Create unique IDs and preset boundaries**

**$this->uniqueid = $this->generateId();**

**$this->boundary[1] = 'b1\_' . $this->uniqueid;**

**$this->boundary[2] = 'b2\_' . $this->uniqueid;**

**$this->boundary[3] = 'b3\_' . $this->uniqueid;**

**if ($this->sign\_key\_file) {**

**$body .= $this->getMailMIME() . $this->LE;**

**}**

**$this->setWordWrap();**

**$bodyEncoding = $this->Encoding;**

**$bodyCharSet = $this->CharSet;**

**//Can we do a 7-bit downgrade?**

**if ($bodyEncoding == '8bit' and !$this->has8bitChars($this->Body)) {**

**$bodyEncoding = '7bit';**

**//All ISO 8859, Windows codepage and UTF-8 charsets are ascii compatible up to 7-bit**

**$bodyCharSet = 'us-ascii';**

**}**

**//If lines are too long, and we're not already using an encoding that will shorten them,**

**//change to quoted-printable transfer encoding for the body part only**

**if ('base64' != $this->Encoding and self::hasLineLongerThanMax($this->Body)) {**

**$bodyEncoding = 'quoted-printable';**

**}**

**$altBodyEncoding = $this->Encoding;**

**$altBodyCharSet = $this->CharSet;**

**//Can we do a 7-bit downgrade?**

**if ($altBodyEncoding == '8bit' and !$this->has8bitChars($this->AltBody)) {**

**$altBodyEncoding = '7bit';**

**//All ISO 8859, Windows codepage and UTF-8 charsets are ascii compatible up to 7-bit**

**$altBodyCharSet = 'us-ascii';**

**}**

**//If lines are too long, and we're not already using an encoding that will shorten them,**

**//change to quoted-printable transfer encoding for the alt body part only**

**if ('base64' != $altBodyEncoding and self::hasLineLongerThanMax($this->AltBody)) {**

**$altBodyEncoding = 'quoted-printable';**

**}**

**//Use this as a preamble in all multipart message types**

**$mimepre = "This is a multi-part message in MIME format." . $this->LE . $this->LE;**

**switch ($this->message\_type) {**

**case 'inline':**

**$body .= $mimepre;**

**$body .= $this->getBoundary($this->boundary[1], $bodyCharSet, '', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->attachAll('inline', $this->boundary[1]);**

**break;**

**case 'attach':**

**$body .= $mimepre;**

**$body .= $this->getBoundary($this->boundary[1], $bodyCharSet, '', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->attachAll('attachment', $this->boundary[1]);**

**break;**

**case 'inline\_attach':**

**$body .= $mimepre;**

**$body .= $this->textLine('--' . $this->boundary[1]);**

**$body .= $this->headerLine('Content-Type', 'multipart/related;');**

**$body .= $this->textLine("\tboundary=\"" . $this->boundary[2] . '"');**

**$body .= $this->LE;**

**$body .= $this->getBoundary($this->boundary[2], $bodyCharSet, '', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->attachAll('inline', $this->boundary[2]);**

**$body .= $this->LE;**

**$body .= $this->attachAll('attachment', $this->boundary[1]);**

**break;**

**case 'alt':**

**$body .= $mimepre;**

**$body .= $this->getBoundary($this->boundary[1], $altBodyCharSet, 'text/plain', $altBodyEncoding);**

**$body .= $this->encodeString($this->AltBody, $altBodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->getBoundary($this->boundary[1], $bodyCharSet, 'text/html', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**if (!empty($this->Ical)) {**

**$body .= $this->getBoundary($this->boundary[1], '', 'text/calendar; method=REQUEST', '');**

**$body .= $this->encodeString($this->Ical, $this->Encoding);**

**$body .= $this->LE . $this->LE;**

**}**

**$body .= $this->endBoundary($this->boundary[1]);**

**break;**

**case 'alt\_inline':**

**$body .= $mimepre;**

**$body .= $this->getBoundary($this->boundary[1], $altBodyCharSet, 'text/plain', $altBodyEncoding);**

**$body .= $this->encodeString($this->AltBody, $altBodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->textLine('--' . $this->boundary[1]);**

**$body .= $this->headerLine('Content-Type', 'multipart/related;');**

**$body .= $this->textLine("\tboundary=\"" . $this->boundary[2] . '"');**

**$body .= $this->LE;**

**$body .= $this->getBoundary($this->boundary[2], $bodyCharSet, 'text/html', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->attachAll('inline', $this->boundary[2]);**

**$body .= $this->LE;**

**$body .= $this->endBoundary($this->boundary[1]);**

**break;**

**case 'alt\_attach':**

**$body .= $mimepre;**

**$body .= $this->textLine('--' . $this->boundary[1]);**

**$body .= $this->headerLine('Content-Type', 'multipart/alternative;');**

**$body .= $this->textLine("\tboundary=\"" . $this->boundary[2] . '"');**

**$body .= $this->LE;**

**$body .= $this->getBoundary($this->boundary[2], $altBodyCharSet, 'text/plain', $altBodyEncoding);**

**$body .= $this->encodeString($this->AltBody, $altBodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->getBoundary($this->boundary[2], $bodyCharSet, 'text/html', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->endBoundary($this->boundary[2]);**

**$body .= $this->LE;**

**$body .= $this->attachAll('attachment', $this->boundary[1]);**

**break;**

**case 'alt\_inline\_attach':**

**$body .= $mimepre;**

**$body .= $this->textLine('--' . $this->boundary[1]);**

**$body .= $this->headerLine('Content-Type', 'multipart/alternative;');**

**$body .= $this->textLine("\tboundary=\"" . $this->boundary[2] . '"');**

**$body .= $this->LE;**

**$body .= $this->getBoundary($this->boundary[2], $altBodyCharSet, 'text/plain', $altBodyEncoding);**

**$body .= $this->encodeString($this->AltBody, $altBodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->textLine('--' . $this->boundary[2]);**

**$body .= $this->headerLine('Content-Type', 'multipart/related;');**

**$body .= $this->textLine("\tboundary=\"" . $this->boundary[3] . '"');**

**$body .= $this->LE;**

**$body .= $this->getBoundary($this->boundary[3], $bodyCharSet, 'text/html', $bodyEncoding);**

**$body .= $this->encodeString($this->Body, $bodyEncoding);**

**$body .= $this->LE . $this->LE;**

**$body .= $this->attachAll('inline', $this->boundary[3]);**

**$body .= $this->LE;**

**$body .= $this->endBoundary($this->boundary[2]);**

**$body .= $this->LE;**

**$body .= $this->attachAll('attachment', $this->boundary[1]);**

**break;**

**default:**

**// Catch case 'plain' and case '', applies to simple `text/plain` and `text/html` body content types**

**//Reset the `Encoding` property in case we changed it for line length reasons**

**$this->Encoding = $bodyEncoding;**

**$body .= $this->encodeString($this->Body, $this->Encoding);**

**break;**

**}**

**if ($this->isError()) {**

**$body = '';**

**} elseif ($this->sign\_key\_file) {**

**try {**

**if (!defined('PKCS7\_TEXT')) {**

**throw new phpmailerException($this->lang('extension\_missing') . 'openssl');**

**}**

**// @TODO would be nice to use php://temp streams here, but need to wrap for PHP < 5.1**

**$file = tempnam(sys\_get\_temp\_dir(), 'mail');**

**if (false === file\_put\_contents($file, $body)) {**

**throw new phpmailerException($this->lang('signing') . ' Could not write temp file');**

**}**

**$signed = tempnam(sys\_get\_temp\_dir(), 'signed');**

**//Workaround for PHP bug https://bugs.php.net/bug.php?id=69197**

**if (empty($this->sign\_extracerts\_file)) {**

**$sign = @openssl\_pkcs7\_sign(**

**$file,**

**$signed,**

**'file://' . realpath($this->sign\_cert\_file),**

**array('file://' . realpath($this->sign\_key\_file), $this->sign\_key\_pass),**

**null**

**);**

**} else {**

**$sign = @openssl\_pkcs7\_sign(**

**$file,**

**$signed,**

**'file://' . realpath($this->sign\_cert\_file),**

**array('file://' . realpath($this->sign\_key\_file), $this->sign\_key\_pass),**

**null,**

**PKCS7\_DETACHED,**

**$this->sign\_extracerts\_file**

**);**

**}**

**if ($sign) {**

**@unlink($file);**

**$body = file\_get\_contents($signed);**

**@unlink($signed);**

**//The message returned by openssl contains both headers and body, so need to split them up**

**$parts = explode("\n\n", $body, 2);**

**$this->MIMEHeader .= $parts[0] . $this->LE . $this->LE;**

**$body = $parts[1];**

**} else {**

**@unlink($file);**

**@unlink($signed);**

**throw new phpmailerException($this->lang('signing') . openssl\_error\_string());**

**}**

**} catch (phpmailerException $exc) {**

**$body = '';**

**if ($this->exceptions) {**

**throw $exc;**

**}**

**}**

**}**

**return $body;**

**}**

**/\*\***

**\* Return the start of a message boundary.**

**\* @access protected**

**\* @param string $boundary**

**\* @param string $charSet**

**\* @param string $contentType**

**\* @param string $encoding**

**\* @return string**

**\*/**

**protected function getBoundary($boundary, $charSet, $contentType, $encoding)**

**{**

**$result = '';**

**if ($charSet == '') {**

**$charSet = $this->CharSet;**

**}**

**if ($contentType == '') {**

**$contentType = $this->ContentType;**

**}**

**if ($encoding == '') {**

**$encoding = $this->Encoding;**

**}**

**$result .= $this->textLine('--' . $boundary);**

**$result .= sprintf('Content-Type: %s; charset=%s', $contentType, $charSet);**

**$result .= $this->LE;**

**// RFC1341 part 5 says 7bit is assumed if not specified**

**if ($encoding != '7bit') {**

**$result .= $this->headerLine('Content-Transfer-Encoding', $encoding);**

**}**

**$result .= $this->LE;**

**return $result;**

**}**

**/\*\***

**\* Return the end of a message boundary.**

**\* @access protected**

**\* @param string $boundary**

**\* @return string**

**\*/**

**protected function endBoundary($boundary)**

**{**

**return $this->LE . '--' . $boundary . '--' . $this->LE;**

**}**

**/\*\***

**\* Set the message type.**

**\* PHPMailer only supports some preset message types, not arbitrary MIME structures.**

**\* @access protected**

**\* @return void**

**\*/**

**protected function setMessageType()**

**{**

**$type = array();**

**if ($this->alternativeExists()) {**

**$type[] = 'alt';**

**}**

**if ($this->inlineImageExists()) {**

**$type[] = 'inline';**

**}**

**if ($this->attachmentExists()) {**

**$type[] = 'attach';**

**}**

**$this->message\_type = implode('\_', $type);**

**if ($this->message\_type == '') {**

**//The 'plain' message\_type refers to the message having a single body element, not that it is plain-text**

**$this->message\_type = 'plain';**

**}**

**}**

**/\*\***

**\* Format a header line.**

**\* @access public**

**\* @param string $name**

**\* @param string $value**

**\* @return string**

**\*/**

**public function headerLine($name, $value)**

**{**

**return $name . ': ' . $value . $this->LE;**

**}**

**/\*\***

**\* Return a formatted mail line.**

**\* @access public**

**\* @param string $value**

**\* @return string**

**\*/**

**public function textLine($value)**

**{**

**return $value . $this->LE;**

**}**

**/\*\***

**\* Add an attachment from a path on the filesystem.**

**\* Never use a user-supplied path to a file!**

**\* Returns false if the file could not be found or read.**

**\* Explicitly \*does not\* support passing URLs; PHPMailer is not an HTTP client.**

**\* If you need to do that, fetch the resource yourself and pass it in via a local file or string.**

**\* @param string $path Path to the attachment.**

**\* @param string $name Overrides the attachment name.**

**\* @param string $encoding File encoding (see $Encoding).**

**\* @param string $type File extension (MIME) type.**

**\* @param string $disposition Disposition to use**

**\* @throws phpmailerException**

**\* @return boolean**

**\*/**

**public function addAttachment($path, $name = '', $encoding = 'base64', $type = '', $disposition = 'attachment')**

**{**

**try {**

**if (!self::isPermittedPath($path) or !@is\_file($path)) {**

**throw new phpmailerException($this->lang('file\_access') . $path, self::STOP\_CONTINUE);**

**}**

**// If a MIME type is not specified, try to work it out from the file name**

**if ($type == '') {**

**$type = self::filenameToType($path);**

**}**

**$filename = basename($path);**

**if ($name == '') {**

**$name = $filename;**

**}**

**$this->attachment[] = array(**

**0 => $path,**

**1 => $filename,**

**2 => $name,**

**3 => $encoding,**

**4 => $type,**

**5 => false, // isStringAttachment**

**6 => $disposition,**

**7 => 0**

**);**

**} catch (phpmailerException $exc) {**

**$this->setError($exc->getMessage());**

**$this->edebug($exc->getMessage());**

**if ($this->exceptions) {**

**throw $exc;**

**}**

**return false;**

**}**

**return true;**

**}**

**/\*\***

**\* Return the array of attachments.**

**\* @return array**

**\*/**

**public function getAttachments()**

**{**

**return $this->attachment;**

**}**

**/\*\***

**\* Attach all file, string, and binary attachments to the message.**

**\* Returns an empty string on failure.**

**\* @access protected**

**\* @param string $disposition\_type**

**\* @param string $boundary**

**\* @return string**

**\*/**

**protected function attachAll($disposition\_type, $boundary)**

**{**

**// Return text of body**

**$mime = array();**

**$cidUniq = array();**

**$incl = array();**

**// Add all attachments**

**foreach ($this->attachment as $attachment) {**

**// Check if it is a valid disposition\_filter**

**if ($attachment[6] == $disposition\_type) {**

**// Check for string attachment**

**$string = '';**

**$path = '';**

**$bString = $attachment[5];**

**if ($bString) {**

**$string = $attachment[0];**

**} else {**

**$path = $attachment[0];**

**}**

**$inclhash = md5(serialize($attachment));**

**if (in\_array($inclhash, $incl)) {**

**continue;**

**}**

**$incl[] = $inclhash;**

**$name = $attachment[2];**

**$encoding = $attachment[3];**

**$type = $attachment[4];**

**$disposition = $attachment[6];**

**$cid = $attachment[7];**

**if ($disposition == 'inline' && array\_key\_exists($cid, $cidUniq)) {**

**continue;**

**}**

**$cidUniq[$cid] = true;**

**$mime[] = sprintf('--%s%s', $boundary, $this->LE);**

**//Only include a filename property if we have one**

**if (!empty($name)) {**

**$mime[] = sprintf(**

**'Content-Type: %s; name="%s"%s',**

**$type,**

**$this->encodeHeader($this->secureHeader($name)),**

**$this->LE**

**);**

**} else {**

**$mime[] = sprintf(**

**'Content-Type: %s%s',**

**$type,**

**$this->LE**

**);**

**}**

**// RFC1341 part 5 says 7bit is assumed if not specified**

**if ($encoding != '7bit') {**

**$mime[] = sprintf('Content-Transfer-Encoding: %s%s', $encoding, $this->LE);**

**}**

**if ($disposition == 'inline') {**

**$mime[] = sprintf('Content-ID: <%s>%s', $cid, $this->LE);**

**}**

**// If a filename contains any of these chars, it should be quoted,**

**// but not otherwise: RFC2183 & RFC2045 5.1**

**// Fixes a warning in IETF's msglint MIME checker**

**// Allow for bypassing the Content-Disposition header totally**

**if (!(empty($disposition))) {**

**$encoded\_name = $this->encodeHeader($this->secureHeader($name));**

**if (preg\_match('/[ \(\)<>@,;:\\"\/\[\]\?=]/', $encoded\_name)) {**

**$mime[] = sprintf(**

**'Content-Disposition: %s; filename="%s"%s',**

**$disposition,**

**$encoded\_name,**

**$this->LE . $this->LE**

**);**

**} else {**

**if (!empty($encoded\_name)) {**

**$mime[] = sprintf(**

**'Content-Disposition: %s; filename=%s%s',**

**$disposition,**

**$encoded\_name,**

**$this->LE . $this->LE**

**);**

**} else {**

**$mime[] = sprintf(**

**'Content-Disposition: %s%s',**

**$disposition,**

**$this->LE . $this->LE**

**);**

**}**

**}**

**} else {**

**$mime[] = $this->LE;**

**}**

**// Encode as string attachment**

**if ($bString) {**

**$mime[] = $this->encodeString($string, $encoding);**

**if ($this->isError()) {**

**return '';**

**}**

**$mime[] = $this->LE . $this->LE;**

**} else {**

**$mime[] = $this->encodeFile($path, $encoding);**

**if ($this->isError()) {**

**return '';**

**}**

**$mime[] = $this->LE . $this->LE;**

**}**

**}**

**}**

**$mime[] = sprintf('--%s--%s', $boundary, $this->LE);**

**return implode('', $mime);**

**}**

**/\*\***

**\* Encode a file attachment in requested format.**

**\* Returns an empty string on failure.**

**\* @param string $path The full path to the file**

**\* @param string $encoding The encoding to use; one of 'base64', '7bit', '8bit', 'binary', 'quoted-printable'**

**\* @throws phpmailerException**

**\* @access protected**

**\* @return string**

**\*/**

**protected function encodeFile($path, $encoding = 'base64')**

**{**

**try {**

**if (!self::isPermittedPath($path) or !file\_exists($path)) {**

**throw new phpmailerException($this->lang('file\_open') . $path, self::STOP\_CONTINUE);**

**}**

**$magic\_quotes = false;**

**if( version\_compare(PHP\_VERSION, '7.4.0', '<') ) {**

**$magic\_quotes = get\_magic\_quotes\_runtime();**

**}**

**if ($magic\_quotes) {**

**if (version\_compare(PHP\_VERSION, '5.3.0', '<')) {**

**set\_magic\_quotes\_runtime(false);**

**} else {**

**//Doesn't exist in PHP 5.4, but we don't need to check because**

**//get\_magic\_quotes\_runtime always returns false in 5.4+**

**//so it will never get here**

**ini\_set('magic\_quotes\_runtime', false);**

**}**

**}**

**$file\_buffer = file\_get\_contents($path);**

**$file\_buffer = $this->encodeString($file\_buffer, $encoding);**

**if ($magic\_quotes) {**

**if (version\_compare(PHP\_VERSION, '5.3.0', '<')) {**

**set\_magic\_quotes\_runtime($magic\_quotes);**

**} else {**

**ini\_set('magic\_quotes\_runtime', $magic\_quotes);**

**}**

**}**

**return $file\_buffer;**

**} catch (Exception $exc) {**

**$this->setError($exc->getMessage());**

**return '';**

**}**

**}**

**/\*\***

**\* Encode a string in requested format.**

**\* Returns an empty string on failure.**

**\* @param string $str The text to encode**

**\* @param string $encoding The encoding to use; one of 'base64', '7bit', '8bit', 'binary', 'quoted-printable'**

**\* @access public**

**\* @return string**

**\*/**

**public function encodeString($str, $encoding = 'base64')**

**{**

**$encoded = '';**

**switch (strtolower($encoding)) {**

**case 'base64':**

**$encoded = chunk\_split(base64\_encode($str), 76, $this->LE);**

**break;**

**case '7bit':**

**case '8bit':**

**$encoded = $this->fixEOL($str);**

**// Make sure it ends with a line break**

**if (substr($encoded, -(strlen($this->LE))) != $this->LE) {**

**$encoded .= $this->LE;**

**}**

**break;**

**case 'binary':**

**$encoded = $str;**

**break;**

**case 'quoted-printable':**

**$encoded = $this->encodeQP($str);**

**break;**

**default:**

**$this->setError($this->lang('encoding') . $encoding);**

**break;**

**}**

**return $encoded;**

**}**

**/\*\***

**\* Encode a header string optimally.**

**\* Picks shortest of Q, B, quoted-printable or none.**

**\* @access public**

**\* @param string $str**

**\* @param string $position**

**\* @return string**

**\*/**

**public function encodeHeader($str, $position = 'text')**

**{**

**$matchcount = 0;**

**switch (strtolower($position)) {**

**case 'phrase':**

**if (!preg\_match('/[\200-\377]/', $str)) {**

**// Can't use addslashes as we don't know the value of magic\_quotes\_sybase**

**$encoded = addcslashes($str, "\0..\37\177\\\"");**

**if (($str == $encoded) && !preg\_match('/[^A-Za-z0-9!#$%&\'\*+\/=?^\_`{|}~ -]/', $str)) {**

**return ($encoded);**

**} else {**

**return ("\"$encoded\"");**

**}**

**}**

**$matchcount = preg\_match\_all('/[^\040\041\043-\133\135-\176]/', $str, $matches);**

**break;**

**/\*\* @noinspection PhpMissingBreakStatementInspection \*/**

**case 'comment':**

**$matchcount = preg\_match\_all('/[()"]/', $str, $matches);**

**// Intentional fall-through**

**case 'text':**

**default:**

**$matchcount += preg\_match\_all('/[\000-\010\013\014\016-\037\177-\377]/', $str, $matches);**

**break;**

**}**

**//There are no chars that need encoding**

**if ($matchcount == 0) {**

**return ($str);**

**}**

**$maxlen = 75 - 7 - strlen($this->CharSet);**

**// Try to select the encoding which should produce the shortest output**

**if ($matchcount > strlen($str) / 3) {**

**// More than a third of the content will need encoding, so B encoding will be most efficient**

**$encoding = 'B';**

**if (function\_exists('mb\_strlen') && $this->hasMultiBytes($str)) {**

**// Use a custom function which correctly encodes and wraps long**

**// multibyte strings without breaking lines within a character**

**$encoded = $this->base64EncodeWrapMB($str, "\n");**

**} else {**

**$encoded = base64\_encode($str);**

**$maxlen -= $maxlen % 4;**

**$encoded = trim(chunk\_split($encoded, $maxlen, "\n"));**

**}**

**} else {**

**$encoding = 'Q';**

**$encoded = $this->encodeQ($str, $position);**

**$encoded = $this->wrapText($encoded, $maxlen, true);**

**$encoded = str\_replace('=' . self::CRLF, "\n", trim($encoded));**

**}**

**$encoded = preg\_replace('/^(.\*)$/m', ' =?' . $this->CharSet . "?$encoding?\\1?=", $encoded);**

**$encoded = trim(str\_replace("\n", $this->LE, $encoded));**

**return $encoded;**

**}**

**/\*\***

**\* Check if a string contains multi-byte characters.**

**\* @access public**

**\* @param string $str multi-byte text to wrap encode**

**\* @return boolean**

**\*/**

**public function hasMultiBytes($str)**

**{**

**if (function\_exists('mb\_strlen')) {**

**return (strlen($str) > mb\_strlen($str, $this->CharSet));**

**} else { // Assume no multibytes (we can't handle without mbstring functions anyway)**

**return false;**

**}**

**}**

**/\*\***

**\* Does a string contain any 8-bit chars (in any charset)?**

**\* @param string $text**

**\* @return boolean**

**\*/**

**public function has8bitChars($text)**

**{**

**return (boolean)preg\_match('/[\x80-\xFF]/', $text);**

**}**

**/\*\***

**\* Encode and wrap long multibyte strings for mail headers**

**\* without breaking lines within a character.**

**\* Adapted from a function by paravoid**

**\* @link http://www.php.net/manual/en/function.mb-encode-mimeheader.php#60283**

**\* @access public**

**\* @param string $str multi-byte text to wrap encode**

**\* @param string $linebreak string to use as linefeed/end-of-line**

**\* @return string**

**\*/**

**public function base64EncodeWrapMB($str, $linebreak = null)**

**{**

**$start = '=?' . $this->CharSet . '?B?';**

**$end = '?=';**

**$encoded = '';**

**if ($linebreak === null) {**

**$linebreak = $this->LE;**

**}**

**$mb\_length = mb\_strlen($str, $this->CharSet);**

**// Each line must have length <= 75, including $start and $end**

**$length = 75 - strlen($start) - strlen($end);**

**// Average multi-byte ratio**

**$ratio = $mb\_length / strlen($str);**

**// Base64 has a 4:3 ratio**

**$avgLength = floor($length \* $ratio \* .75);**

**for ($i = 0; $i < $mb\_length; $i += $offset) {**

**$lookBack = 0;**

**do {**

**$offset = $avgLength - $lookBack;**

**$chunk = mb\_substr($str, $i, $offset, $this->CharSet);**

**$chunk = base64\_encode($chunk);**

**$lookBack++;**

**} while (strlen($chunk) > $length);**

**$encoded .= $chunk . $linebreak;**

**}**

**// Chomp the last linefeed**

**$encoded = substr($encoded, 0, -strlen($linebreak));**

**return $encoded;**

**}**

**/\*\***

**\* Encode a string in quoted-printable format.**

**\* According to RFC2045 section 6.7.**

**\* @access public**

**\* @param string $string The text to encode**

**\* @param integer $line\_max Number of chars allowed on a line before wrapping**

**\* @return string**

**\* @link http://www.php.net/manual/en/function.quoted-printable-decode.php#89417 Adapted from this comment**

**\*/**

**public function encodeQP($string, $line\_max = 76)**

**{**

**// Use native function if it's available (>= PHP5.3)**

**if (function\_exists('quoted\_printable\_encode')) {**

**return quoted\_printable\_encode($string);**

**}**

**// Fall back to a pure PHP implementation**

**$string = str\_replace(**

**array('%20', '%0D%0A.', '%0D%0A', '%'),**

**array(' ', "\r\n=2E", "\r\n", '='),**

**rawurlencode($string)**

**);**

**return preg\_replace('/[^\r\n]{' . ($line\_max - 3) . '}[^=\r\n]{2}/', "$0=\r\n", $string);**

**}**

**/\*\***

**\* Backward compatibility wrapper for an old QP encoding function that was removed.**

**\* @see PHPMailer::encodeQP()**

**\* @access public**

**\* @param string $string**

**\* @param integer $line\_max**

**\* @param boolean $space\_conv**

**\* @return string**

**\* @deprecated Use encodeQP instead.**

**\*/**

**public function encodeQPphp(**

**$string,**

**$line\_max = 76,**

**/\*\* @noinspection PhpUnusedParameterInspection \*/ $space\_conv = false**

**) {**

**return $this->encodeQP($string, $line\_max);**

**}**

**/\*\***

**\* Encode a string using Q encoding.**

**\* @link http://tools.ietf.org/html/rfc2047**

**\* @param string $str the text to encode**

**\* @param string $position Where the text is going to be used, see the RFC for what that means**

**\* @access public**

**\* @return string**

**\*/**

**public function encodeQ($str, $position = 'text')**

**{**

**// There should not be any EOL in the string**

**$pattern = '';**

**$encoded = str\_replace(array("\r", "\n"), '', $str);**

**switch (strtolower($position)) {**

**case 'phrase':**

**// RFC 2047 section 5.3**

**$pattern = '^A-Za-z0-9!\*+\/ -';**

**break;**

**/\*\* @noinspection PhpMissingBreakStatementInspection \*/**

**case 'comment':**

**// RFC 2047 section 5.2**

**$pattern = '\(\)"';**

**// intentional fall-through**

**// for this reason we build the $pattern without including delimiters and []**

**case 'text':**

**default:**

**// RFC 2047 section 5.1**

**// Replace every high ascii, control, =, ? and \_ characters**

**$pattern = '\000-\011\013\014\016-\037\075\077\137\177-\377' . $pattern;**

**break;**

**}**

**$matches = array();**

**if (preg\_match\_all("/[{$pattern}]/", $encoded, $matches)) {**

**// If the string contains an '=', make sure it's the first thing we replace**

**// so as to avoid double-encoding**

**$eqkey = array\_search('=', $matches[0]);**

**if (false !== $eqkey) {**

**unset($matches[0][$eqkey]);**

**array\_unshift($matches[0], '=');**

**}**

**foreach (array\_unique($matches[0]) as $char) {**

**$encoded = str\_replace($char, '=' . sprintf('%02X', ord($char)), $encoded);**

**}**

**}**

**// Replace every spaces to \_ (more readable than =20)**

**return str\_replace(' ', '\_', $encoded);**

**}**

**/\*\***

**\* Add a string or binary attachment (non-filesystem).**

**\* This method can be used to attach ascii or binary data,**

**\* such as a BLOB record from a database.**

**\* @param string $string String attachment data.**

**\* @param string $filename Name of the attachment.**

**\* @param string $encoding File encoding (see $Encoding).**

**\* @param string $type File extension (MIME) type.**

**\* @param string $disposition Disposition to use**

**\* @return void**

**\*/**

**public function addStringAttachment(**

**$string,**

**$filename,**

**$encoding = 'base64',**

**$type = '',**

**$disposition = 'attachment'**

**) {**

**// If a MIME type is not specified, try to work it out from the file name**

**if ($type == '') {**

**$type = self::filenameToType($filename);**

**}**

**// Append to $attachment array**

**$this->attachment[] = array(**

**0 => $string,**

**1 => $filename,**

**2 => basename($filename),**

**3 => $encoding,**

**4 => $type,**

**5 => true, // isStringAttachment**

**6 => $disposition,**

**7 => 0**

**);**

**}**

**/\*\***

**\* Add an embedded (inline) attachment from a file.**

**\* This can include images, sounds, and just about any other document type.**

**\* These differ from 'regular' attachments in that they are intended to be**

**\* displayed inline with the message, not just attached for download.**

**\* This is used in HTML messages that embed the images**

**\* the HTML refers to using the $cid value.**

**\* Never use a user-supplied path to a file!**

**\* @param string $path Path to the attachment.**

**\* @param string $cid Content ID of the attachment; Use this to reference**

**\* the content when using an embedded image in HTML.**

**\* @param string $name Overrides the attachment name.**

**\* @param string $encoding File encoding (see $Encoding).**

**\* @param string $type File MIME type.**

**\* @param string $disposition Disposition to use**

**\* @return boolean True on successfully adding an attachment**

**\*/**

**public function addEmbeddedImage($path, $cid, $name = '', $encoding = 'base64', $type = '', $disposition = 'inline')**

**{**

**if (!self::isPermittedPath($path) or !@is\_file($path)) {**

**$this->setError($this->lang('file\_access') . $path);**

**return false;**

**}**

**// If a MIME type is not specified, try to work it out from the file name**

**if ($type == '') {**

**$type = self::filenameToType($path);**

**}**

**$filename = basename($path);**

**if ($name == '') {**

**$name = $filename;**

**}**

**// Append to $attachment array**

**$this->attachment[] = array(**

**0 => $path,**

**1 => $filename,**

**2 => $name,**

**3 => $encoding,**

**4 => $type,**

**5 => false, // isStringAttachment**

**6 => $disposition,**

**7 => $cid**

**);**

**return true;**

**}**

**/\*\***

**\* Add an embedded stringified attachment.**

**\* This can include images, sounds, and just about any other document type.**

**\* Be sure to set the $type to an image type for images:**

**\* JPEG images use 'image/jpeg', GIF uses 'image/gif', PNG uses 'image/png'.**

**\* @param string $string The attachment binary data.**

**\* @param string $cid Content ID of the attachment; Use this to reference**

**\* the content when using an embedded image in HTML.**

**\* @param string $name**

**\* @param string $encoding File encoding (see $Encoding).**

**\* @param string $type MIME type.**

**\* @param string $disposition Disposition to use**

**\* @return boolean True on successfully adding an attachment**

**\*/**

**public function addStringEmbeddedImage(**

**$string,**

**$cid,**

**$name = '',**

**$encoding = 'base64',**

**$type = '',**

**$disposition = 'inline'**

**) {**

**// If a MIME type is not specified, try to work it out from the name**

**if ($type == '' and !empty($name)) {**

**$type = self::filenameToType($name);**

**}**

**// Append to $attachment array**

**$this->attachment[] = array(**

**0 => $string,**

**1 => $name,**

**2 => $name,**

**3 => $encoding,**

**4 => $type,**

**5 => true, // isStringAttachment**

**6 => $disposition,**

**7 => $cid**

**);**

**return true;**

**}**

**/\*\***

**\* Check if an inline attachment is present.**

**\* @access public**

**\* @return boolean**

**\*/**

**public function inlineImageExists()**

**{**

**foreach ($this->attachment as $attachment) {**

**if ($attachment[6] == 'inline') {**

**return true;**

**}**

**}**

**return false;**

**}**

**/\*\***

**\* Check if an attachment (non-inline) is present.**

**\* @return boolean**

**\*/**

**public function attachmentExists()**

**{**

**foreach ($this->attachment as $attachment) {**

**if ($attachment[6] == 'attachment') {**

**return true;**

**}**

**}**

**return false;**

**}**

**/\*\***

**\* Check if this message has an alternative body set.**

**\* @return boolean**

**\*/**

**public function alternativeExists()**

**{**

**return !empty($this->AltBody);**

**}**

**/\*\***

**\* Clear queued addresses of given kind.**

**\* @access protected**

**\* @param string $kind 'to', 'cc', or 'bcc'**

**\* @return void**

**\*/**

**public function clearQueuedAddresses($kind)**

**{**

**$RecipientsQueue = $this->RecipientsQueue;**

**foreach ($RecipientsQueue as $address => $params) {**

**if ($params[0] == $kind) {**

**unset($this->RecipientsQueue[$address]);**

**}**

**}**

**}**

**/\*\***

**\* Clear all To recipients.**

**\* @return void**

**\*/**

**public function clearAddresses()**

**{**

**foreach ($this->to as $to) {**

**unset($this->all\_recipients[strtolower($to[0])]);**

**}**

**$this->to = array();**

**$this->clearQueuedAddresses('to');**

**}**

**/\*\***

**\* Clear all CC recipients.**

**\* @return void**

**\*/**

**public function clearCCs()**

**{**

**foreach ($this->cc as $cc) {**

**unset($this->all\_recipients[strtolower($cc[0])]);**

**}**

**$this->cc = array();**

**$this->clearQueuedAddresses('cc');**

**}**

**/\*\***

**\* Clear all BCC recipients.**

**\* @return void**

**\*/**

**public function clearBCCs()**

**{**

**foreach ($this->bcc as $bcc) {**

**unset($this->all\_recipients[strtolower($bcc[0])]);**

**}**

**$this->bcc = array();**

**$this->clearQueuedAddresses('bcc');**

**}**

**/\*\***

**\* Clear all ReplyTo recipients.**

**\* @return void**

**\*/**

**public function clearReplyTos()**

**{**

**$this->ReplyTo = array();**

**$this->ReplyToQueue = array();**

**}**

**/\*\***

**\* Clear all recipient types.**

**\* @return void**

**\*/**

**public function clearAllRecipients()**

**{**

**$this->to = array();**

**$this->cc = array();**

**$this->bcc = array();**

**$this->all\_recipients = array();**

**$this->RecipientsQueue = array();**

**}**

**/\*\***

**\* Clear all filesystem, string, and binary attachments.**

**\* @return void**

**\*/**

**public function clearAttachments()**

**{**

**$this->attachment = array();**

**}**

**/\*\***

**\* Clear all custom headers.**

**\* @return void**

**\*/**

**public function clearCustomHeaders()**

**{**

**$this->CustomHeader = array();**

**}**

**/\*\***

**\* Add an error message to the error container.**

**\* @access protected**

**\* @param string $msg**

**\* @return void**

**\*/**

**protected function setError($msg)**

**{**

**$this->error\_count++;**

**if ($this->Mailer == 'smtp' and !is\_null($this->smtp)) {**

**$lasterror = $this->smtp->getError();**

**if (!empty($lasterror['error'])) {**

**$msg .= $this->lang('smtp\_error') . $lasterror['error'];**

**if (!empty($lasterror['detail'])) {**

**$msg .= ' Detail: '. $lasterror['detail'];**

**}**

**if (!empty($lasterror['smtp\_code'])) {**

**$msg .= ' SMTP code: ' . $lasterror['smtp\_code'];**

**}**

**if (!empty($lasterror['smtp\_code\_ex'])) {**

**$msg .= ' Additional SMTP info: ' . $lasterror['smtp\_code\_ex'];**

**}**

**}**

**}**

**$this->ErrorInfo = $msg;**

**}**

**/\*\***

**\* Return an RFC 822 formatted date.**

**\* @access public**

**\* @return string**

**\* @static**

**\*/**

**public static function rfcDate()**

**{**

**// Set the time zone to whatever the default is to avoid 500 errors**

**// Will default to UTC if it's not set properly in php.ini**

**date\_default\_timezone\_set(@date\_default\_timezone\_get());**

**return date('D, j M Y H:i:s O');**

**}**

**/\*\***

**\* Get the server hostname.**

**\* Returns 'localhost.localdomain' if unknown.**

**\* @access protected**

**\* @return string**

**\*/**

**protected function serverHostname()**

**{**

**$result = 'localhost.localdomain';**

**if (!empty($this->Hostname)) {**

**$result = $this->Hostname;**

**} elseif (isset($\_SERVER) and array\_key\_exists('SERVER\_NAME', $\_SERVER) and !empty($\_SERVER['SERVER\_NAME'])) {**

**$result = $\_SERVER['SERVER\_NAME'];**

**} elseif (function\_exists('gethostname') && gethostname() !== false) {**

**$result = gethostname();**

**} elseif (php\_uname('n') !== false) {**

**$result = php\_uname('n');**

**}**

**return $result;**

**}**

**/\*\***

**\* Get an error message in the current language.**

**\* @access protected**

**\* @param string $key**

**\* @return string**

**\*/**

**protected function lang($key)**

**{**

**if (count($this->language) < 1) {**

**$this->setLanguage('en'); // set the default language**

**}**

**if (array\_key\_exists($key, $this->language)) {**

**if ($key == 'smtp\_connect\_failed') {**

**//Include a link to troubleshooting docs on SMTP connection failure**

**//this is by far the biggest cause of support questions**

**//but it's usually not PHPMailer's fault.**

**return $this->language[$key] . ' https://github.com/PHPMailer/PHPMailer/wiki/Troubleshooting';**

**}**

**return $this->language[$key];**

**} else {**

**//Return the key as a fallback**

**return $key;**

**}**

**}**

**/\*\***

**\* Check if an error occurred.**

**\* @access public**

**\* @return boolean True if an error did occur.**

**\*/**

**public function isError()**

**{**

**return ($this->error\_count > 0);**

**}**

**/\*\***

**\* Ensure consistent line endings in a string.**

**\* Changes every end of line from CRLF, CR or LF to $this->LE.**

**\* @access public**

**\* @param string $str String to fixEOL**

**\* @return string**

**\*/**

**public function fixEOL($str)**

**{**

**// Normalise to \n**

**$nstr = str\_replace(array("\r\n", "\r"), "\n", $str);**

**// Now convert LE as needed**

**if ($this->LE !== "\n") {**

**$nstr = str\_replace("\n", $this->LE, $nstr);**

**}**

**return $nstr;**

**}**

**/\*\***

**\* Add a custom header.**

**\* $name value can be overloaded to contain**

**\* both header name and value (name:value)**

**\* @access public**

**\* @param string $name Custom header name**

**\* @param string $value Header value**

**\* @return void**

**\*/**

**public function addCustomHeader($name, $value = null)**

**{**

**if ($value === null) {**

**// Value passed in as name:value**

**$this->CustomHeader[] = explode(':', $name, 2);**

**} else {**

**$this->CustomHeader[] = array($name, $value);**

**}**

**}**

**/\*\***

**\* Returns all custom headers.**

**\* @return array**

**\*/**

**public function getCustomHeaders()**

**{**

**return $this->CustomHeader;**

**}**

**/\*\***

**\* Create a message body from an HTML string.**

**\* Automatically inlines images and creates a plain-text version by converting the HTML,**

**\* overwriting any existing values in Body and AltBody.**

**\* Do not source $message content from user input!**

**\* $basedir is prepended when handling relative URLs, e.g. <img src="/images/a.png"> and must not be empty**

**\* will look for an image file in $basedir/images/a.png and convert it to inline.**

**\* If you don't provide a $basedir, relative paths will be left untouched (and thus probably break in email)**

**\* If you don't want to apply these transformations to your HTML, just set Body and AltBody directly.**

**\* @access public**

**\* @param string $message HTML message string**

**\* @param string $basedir Absolute path to a base directory to prepend to relative paths to images**

**\* @param boolean|callable $advanced Whether to use the internal HTML to text converter**

**\* or your own custom converter @see PHPMailer::html2text()**

**\* @return string $message The transformed message Body**

**\*/**

**public function msgHTML($message, $basedir = '', $advanced = false)**

**{**

**preg\_match\_all('/(src|background)=["\'](.\*)["\']/Ui', $message, $images);**

**if (array\_key\_exists(2, $images)) {**

**if (strlen($basedir) > 1 && substr($basedir, -1) != '/') {**

**// Ensure $basedir has a trailing /**

**$basedir .= '/';**

**}**

**foreach ($images[2] as $imgindex => $url) {**

**// Convert data URIs into embedded images**

**if (preg\_match('#^data:(image[^;,]\*)(;base64)?,#', $url, $match)) {**

**$data = substr($url, strpos($url, ','));**

**if ($match[2]) {**

**$data = base64\_decode($data);**

**} else {**

**$data = rawurldecode($data);**

**}**

**$cid = md5($url) . '@phpmailer.0'; // RFC2392 S 2**

**if ($this->addStringEmbeddedImage($data, $cid, 'embed' . $imgindex, 'base64', $match[1])) {**

**$message = str\_replace(**

**$images[0][$imgindex],**

**$images[1][$imgindex] . '="cid:' . $cid . '"',**

**$message**

**);**

**}**

**continue;**

**}**

**if (**

**// Only process relative URLs if a basedir is provided (i.e. no absolute local paths)**

**!empty($basedir)**

**// Ignore URLs containing parent dir traversal (..)**

**&& (strpos($url, '..') === false)**

**// Do not change urls that are already inline images**

**&& substr($url, 0, 4) !== 'cid:'**

**// Do not change absolute URLs, including anonymous protocol**

**&& !preg\_match('#^[a-z][a-z0-9+.-]\*:?//#i', $url)**

**) {**

**$filename = basename($url);**

**$directory = dirname($url);**

**if ($directory == '.') {**

**$directory = '';**

**}**

**$cid = md5($url) . '@phpmailer.0'; // RFC2392 S 2**

**if (strlen($directory) > 1 && substr($directory, -1) != '/') {**

**$directory .= '/';**

**}**

**if ($this->addEmbeddedImage(**

**$basedir . $directory . $filename,**

**$cid,**

**$filename,**

**'base64',**

**self::\_mime\_types((string)self::mb\_pathinfo($filename, PATHINFO\_EXTENSION))**

**)**

**) {**

**$message = preg\_replace(**

**'/' . $images[1][$imgindex] . '=["\']' . preg\_quote($url, '/') . '["\']/Ui',**

**$images[1][$imgindex] . '="cid:' . $cid . '"',**

**$message**

**);**

**}**

**}**

**}**

**}**

**$this->isHTML(true);**

**// Convert all message body line breaks to CRLF, makes quoted-printable encoding work much better**

**$this->Body = $this->normalizeBreaks($message);**

**$this->AltBody = $this->normalizeBreaks($this->html2text($message, $advanced));**

**if (!$this->alternativeExists()) {**

**$this->AltBody = 'To view this email message, open it in a program that understands HTML!' .**

**self::CRLF . self::CRLF;**

**}**

**return $this->Body;**

**}**

**/\*\***

**\* Convert an HTML string into plain text.**

**\* This is used by msgHTML().**

**\* Note - older versions of this function used a bundled advanced converter**

**\* which was been removed for license reasons in #232.**

**\* Example usage:**

**\* <code>**

**\* // Use default conversion**

**\* $plain = $mail->html2text($html);**

**\* // Use your own custom converter**

**\* $plain = $mail->html2text($html, function($html) {**

**\* $converter = new MyHtml2text($html);**

**\* return $converter->get\_text();**

**\* });**

**\* </code>**

**\* @param string $html The HTML text to convert**

**\* @param boolean|callable $advanced Any boolean value to use the internal converter,**

**\* or provide your own callable for custom conversion.**

**\* @return string**

**\*/**

**public function html2text($html, $advanced = false)**

**{**

**if (is\_callable($advanced)) {**

**return call\_user\_func($advanced, $html);**

**}**

**return html\_entity\_decode(**

**trim(strip\_tags(preg\_replace('/<(head|title|style|script)[^>]\*>.\*?<\/\\1>/si', '', $html))),**

**ENT\_QUOTES,**

**$this->CharSet**

**);**

**}**

**/\*\***

**\* Get the MIME type for a file extension.**

**\* @param string $ext File extension**

**\* @access public**

**\* @return string MIME type of file.**

**\* @static**

**\*/**

**public static function \_mime\_types($ext = '')**

**{**

**$mimes = array(**

**'xl' => 'application/excel',**

**'js' => 'application/javascript',**

**'hqx' => 'application/mac-binhex40',**

**'cpt' => 'application/mac-compactpro',**

**'bin' => 'application/macbinary',**

**'doc' => 'application/msword',**

**'word' => 'application/msword',**

**'xlsx' => 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',**

**'xltx' => 'application/vnd.openxmlformats-officedocument.spreadsheetml.template',**

**'potx' => 'application/vnd.openxmlformats-officedocument.presentationml.template',**

**'ppsx' => 'application/vnd.openxmlformats-officedocument.presentationml.slideshow',**

**'pptx' => 'application/vnd.openxmlformats-officedocument.presentationml.presentation',**

**'sldx' => 'application/vnd.openxmlformats-officedocument.presentationml.slide',**

**'docx' => 'application/vnd.openxmlformats-officedocument.wordprocessingml.document',**

**'dotx' => 'application/vnd.openxmlformats-officedocument.wordprocessingml.template',**

**'xlam' => 'application/vnd.ms-excel.addin.macroEnabled.12',**

**'xlsb' => 'application/vnd.ms-excel.sheet.binary.macroEnabled.12',**

**'class' => 'application/octet-stream',**

**'dll' => 'application/octet-stream',**

**'dms' => 'application/octet-stream',**

**'exe' => 'application/octet-stream',**

**'lha' => 'application/octet-stream',**

**'lzh' => 'application/octet-stream',**

**'psd' => 'application/octet-stream',**

**'sea' => 'application/octet-stream',**

**'so' => 'application/octet-stream',**

**'oda' => 'application/oda',**

**'pdf' => 'application/pdf',**

**'ai' => 'application/postscript',**

**'eps' => 'application/postscript',**

**'ps' => 'application/postscript',**

**'smi' => 'application/smil',**

**'smil' => 'application/smil',**

**'mif' => 'application/vnd.mif',**

**'xls' => 'application/vnd.ms-excel',**

**'ppt' => 'application/vnd.ms-powerpoint',**

**'wbxml' => 'application/vnd.wap.wbxml',**

**'wmlc' => 'application/vnd.wap.wmlc',**

**'dcr' => 'application/x-director',**

**'dir' => 'application/x-director',**

**'dxr' => 'application/x-director',**

**'dvi' => 'application/x-dvi',**

**'gtar' => 'application/x-gtar',**

**'php3' => 'application/x-httpd-php',**

**'php4' => 'application/x-httpd-php',**

**'php' => 'application/x-httpd-php',**

**'phtml' => 'application/x-httpd-php',**

**'phps' => 'application/x-httpd-php-source',**

**'swf' => 'application/x-shockwave-flash',**

**'sit' => 'application/x-stuffit',**

**'tar' => 'application/x-tar',**

**'tgz' => 'application/x-tar',**

**'xht' => 'application/xhtml+xml',**

**'xhtml' => 'application/xhtml+xml',**

**'zip' => 'application/zip',**

**'mid' => 'audio/midi',**

**'midi' => 'audio/midi',**

**'mp2' => 'audio/mpeg',**

**'mp3' => 'audio/mpeg',**

**'mpga' => 'audio/mpeg',**

**'aif' => 'audio/x-aiff',**

**'aifc' => 'audio/x-aiff',**

**'aiff' => 'audio/x-aiff',**

**'ram' => 'audio/x-pn-realaudio',**

**'rm' => 'audio/x-pn-realaudio',**

**'rpm' => 'audio/x-pn-realaudio-plugin',**

**'ra' => 'audio/x-realaudio',**

**'wav' => 'audio/x-wav',**

**'bmp' => 'image/bmp',**

**'gif' => 'image/gif',**

**'jpeg' => 'image/jpeg',**

**'jpe' => 'image/jpeg',**

**'jpg' => 'image/jpeg',**

**'png' => 'image/png',**

**'tiff' => 'image/tiff',**

**'tif' => 'image/tiff',**

**'eml' => 'message/rfc822',**

**'css' => 'text/css',**

**'html' => 'text/html',**

**'htm' => 'text/html',**

**'shtml' => 'text/html',**

**'log' => 'text/plain',**

**'text' => 'text/plain',**

**'txt' => 'text/plain',**

**'rtx' => 'text/richtext',**

**'rtf' => 'text/rtf',**

**'vcf' => 'text/vcard',**

**'vcard' => 'text/vcard',**

**'xml' => 'text/xml',**

**'xsl' => 'text/xml',**

**'mpeg' => 'video/mpeg',**

**'mpe' => 'video/mpeg',**

**'mpg' => 'video/mpeg',**

**'mov' => 'video/quicktime',**

**'qt' => 'video/quicktime',**

**'rv' => 'video/vnd.rn-realvideo',**

**'avi' => 'video/x-msvideo',**

**'movie' => 'video/x-sgi-movie'**

**);**

**if (array\_key\_exists(strtolower($ext), $mimes)) {**

**return $mimes[strtolower($ext)];**

**}**

**return 'application/octet-stream';**

**}**

**/\*\***

**\* Map a file name to a MIME type.**

**\* Defaults to 'application/octet-stream', i.e.. arbitrary binary data.**

**\* @param string $filename A file name or full path, does not need to exist as a file**

**\* @return string**

**\* @static**

**\*/**

**public static function filenameToType($filename)**

**{**

**// In case the path is a URL, strip any query string before getting extension**

**$qpos = strpos($filename, '?');**

**if (false !== $qpos) {**

**$filename = substr($filename, 0, $qpos);**

**}**

**$pathinfo = self::mb\_pathinfo($filename);**

**return self::\_mime\_types($pathinfo['extension']);**

**}**

**/\*\***

**\* Multi-byte-safe pathinfo replacement.**

**\* Drop-in replacement for pathinfo(), but multibyte-safe, cross-platform-safe, old-version-safe.**

**\* Works similarly to the one in PHP >= 5.2.0**

**\* @link http://www.php.net/manual/en/function.pathinfo.php#107461**

**\* @param string $path A filename or path, does not need to exist as a file**

**\* @param integer|string $options Either a PATHINFO\_\* constant,**

**\* or a string name to return only the specified piece, allows 'filename' to work on PHP < 5.2**

**\* @return string|array**

**\* @static**

**\*/**

**public static function mb\_pathinfo($path, $options = null)**

**{**

**$ret = array('dirname' => '', 'basename' => '', 'extension' => '', 'filename' => '');**

**$pathinfo = array();**

**if (preg\_match('%^(.\*?)[\\\\/]\*(([^/\\\\]\*?)(\.([^\.\\\\/]+?)|))[\\\\/\.]\*$%im', $path, $pathinfo)) {**

**if (array\_key\_exists(1, $pathinfo)) {**

**$ret['dirname'] = $pathinfo[1];**

**}**

**if (array\_key\_exists(2, $pathinfo)) {**

**$ret['basename'] = $pathinfo[2];**

**}**

**if (array\_key\_exists(5, $pathinfo)) {**

**$ret['extension'] = $pathinfo[5];**

**}**

**if (array\_key\_exists(3, $pathinfo)) {**

**$ret['filename'] = $pathinfo[3];**

**}**

**}**

**switch ($options) {**

**case PATHINFO\_DIRNAME:**

**case 'dirname':**

**return $ret['dirname'];**

**case PATHINFO\_BASENAME:**

**case 'basename':**

**return $ret['basename'];**

**case PATHINFO\_EXTENSION:**

**case 'extension':**

**return $ret['extension'];**

**case PATHINFO\_FILENAME:**

**case 'filename':**

**return $ret['filename'];**

**default:**

**return $ret;**

**}**

**}**

**/\*\***

**\* Set or reset instance properties.**

**\* You should avoid this function - it's more verbose, less efficient, more error-prone and**

**\* harder to debug than setting properties directly.**

**\* Usage Example:**

**\* `$mail->set('SMTPSecure', 'tls');`**

**\* is the same as:**

**\* `$mail->SMTPSecure = 'tls';`**

**\* @access public**

**\* @param string $name The property name to set**

**\* @param mixed $value The value to set the property to**

**\* @return boolean**

**\* @TODO Should this not be using the \_\_set() magic function?**

**\*/**

**public function set($name, $value = '')**

**{**

**if (property\_exists($this, $name)) {**

**$this->$name = $value;**

**return true;**

**} else {**

**$this->setError($this->lang('variable\_set') . $name);**

**return false;**

**}**

**}**

**/\*\***

**\* Strip newlines to prevent header injection.**

**\* @access public**

**\* @param string $str**

**\* @return string**

**\*/**

**public function secureHeader($str)**

**{**

**return trim(str\_replace(array("\r", "\n"), '', $str));**

**}**

**/\*\***

**\* Normalize line breaks in a string.**

**\* Converts UNIX LF, Mac CR and Windows CRLF line breaks into a single line break format.**

**\* Defaults to CRLF (for message bodies) and preserves consecutive breaks.**

**\* @param string $text**

**\* @param string $breaktype What kind of line break to use, defaults to CRLF**

**\* @return string**

**\* @access public**

**\* @static**

**\*/**

**public static function normalizeBreaks($text, $breaktype = "\r\n")**

**{**

**return preg\_replace('/(\r\n|\r|\n)/ms', $breaktype, $text);**

**}**

**/\*\***

**\* Set the public and private key files and password for S/MIME signing.**

**\* @access public**

**\* @param string $cert\_filename**

**\* @param string $key\_filename**

**\* @param string $key\_pass Password for private key**

**\* @param string $extracerts\_filename Optional path to chain certificate**

**\*/**

**public function sign($cert\_filename, $key\_filename, $key\_pass, $extracerts\_filename = '')**

**{**

**$this->sign\_cert\_file = $cert\_filename;**

**$this->sign\_key\_file = $key\_filename;**

**$this->sign\_key\_pass = $key\_pass;**

**$this->sign\_extracerts\_file = $extracerts\_filename;**

**}**

**/\*\***

**\* Quoted-Printable-encode a DKIM header.**

**\* @access public**

**\* @param string $txt**

**\* @return string**

**\*/**

**public function DKIM\_QP($txt)**

**{**

**$line = '';**

**for ($i = 0; $i < strlen($txt); $i++) {**

**$ord = ord($txt[$i]);**

**if (((0x21 <= $ord) && ($ord <= 0x3A)) || $ord == 0x3C || ((0x3E <= $ord) && ($ord <= 0x7E))) {**

**$line .= $txt[$i];**

**} else {**

**$line .= '=' . sprintf('%02X', $ord);**

**}**

**}**

**return $line;**

**}**

**/\*\***

**\* Generate a DKIM signature.**

**\* @access public**

**\* @param string $signHeader**

**\* @throws phpmailerException**

**\* @return string The DKIM signature value**

**\*/**

**public function DKIM\_Sign($signHeader)**

**{**

**if (!defined('PKCS7\_TEXT')) {**

**if ($this->exceptions) {**

**throw new phpmailerException($this->lang('extension\_missing') . 'openssl');**

**}**

**return '';**

**}**

**$privKeyStr = !empty($this->DKIM\_private\_string) ? $this->DKIM\_private\_string : file\_get\_contents($this->DKIM\_private);**

**if ('' != $this->DKIM\_passphrase) {**

**$privKey = openssl\_pkey\_get\_private($privKeyStr, $this->DKIM\_passphrase);**

**} else {**

**$privKey = openssl\_pkey\_get\_private($privKeyStr);**

**}**

**//Workaround for missing digest algorithms in old PHP & OpenSSL versions**

**//@link http://stackoverflow.com/a/11117338/333340**

**if (version\_compare(PHP\_VERSION, '5.3.0') >= 0 and**

**in\_array('sha256WithRSAEncryption', openssl\_get\_md\_methods(true))) {**

**if (openssl\_sign($signHeader, $signature, $privKey, 'sha256WithRSAEncryption')) {**

**openssl\_pkey\_free($privKey);**

**return base64\_encode($signature);**

**}**

**} else {**

**$pinfo = openssl\_pkey\_get\_details($privKey);**

**$hash = hash('sha256', $signHeader);**

**//'Magic' constant for SHA256 from RFC3447**

**//@link https://tools.ietf.org/html/rfc3447#page-43**

**$t = '3031300d060960864801650304020105000420' . $hash;**

**$pslen = $pinfo['bits'] / 8 - (strlen($t) / 2 + 3);**

**$eb = pack('H\*', '0001' . str\_repeat('FF', $pslen) . '00' . $t);**

**if (openssl\_private\_encrypt($eb, $signature, $privKey, OPENSSL\_NO\_PADDING)) {**

**openssl\_pkey\_free($privKey);**

**return base64\_encode($signature);**

**}**

**}**

**openssl\_pkey\_free($privKey);**

**return '';**

**}**

**/\*\***

**\* Generate a DKIM canonicalization header.**

**\* @access public**

**\* @param string $signHeader Header**

**\* @return string**

**\*/**

**public function DKIM\_HeaderC($signHeader)**

**{**

**$signHeader = preg\_replace('/\r\n\s+/', ' ', $signHeader);**

**$lines = explode("\r\n", $signHeader);**

**foreach ($lines as $key => $line) {**

**list($heading, $value) = explode(':', $line, 2);**

**$heading = strtolower($heading);**

**$value = preg\_replace('/\s{2,}/', ' ', $value); // Compress useless spaces**

**$lines[$key] = $heading . ':' . trim($value); // Don't forget to remove WSP around the value**

**}**

**$signHeader = implode("\r\n", $lines);**

**return $signHeader;**

**}**

**/\*\***

**\* Generate a DKIM canonicalization body.**

**\* @access public**

**\* @param string $body Message Body**

**\* @return string**

**\*/**

**public function DKIM\_BodyC($body)**

**{**

**if ($body == '') {**

**return "\r\n";**

**}**

**// stabilize line endings**

**$body = str\_replace("\r\n", "\n", $body);**

**$body = str\_replace("\n", "\r\n", $body);**

**// END stabilize line endings**

**while (substr($body, strlen($body) - 4, 4) == "\r\n\r\n") {**

**$body = substr($body, 0, strlen($body) - 2);**

**}**

**return $body;**

**}**

**/\*\***

**\* Create the DKIM header and body in a new message header.**

**\* @access public**

**\* @param string $headers\_line Header lines**

**\* @param string $subject Subject**

**\* @param string $body Body**

**\* @return string**

**\*/**

**public function DKIM\_Add($headers\_line, $subject, $body)**

**{**

**$DKIMsignatureType = 'rsa-sha256'; // Signature & hash algorithms**

**$DKIMcanonicalization = 'relaxed/simple'; // Canonicalization of header/body**

**$DKIMquery = 'dns/txt'; // Query method**

**$DKIMtime = time(); // Signature Timestamp = seconds since 00:00:00 - Jan 1, 1970 (UTC time zone)**

**$subject\_header = "Subject: $subject";**

**$headers = explode($this->LE, $headers\_line);**

**$from\_header = '';**

**$to\_header = '';**

**$date\_header = '';**

**$current = '';**

**foreach ($headers as $header) {**

**if (strpos($header, 'From:') === 0) {**

**$from\_header = $header;**

**$current = 'from\_header';**

**} elseif (strpos($header, 'To:') === 0) {**

**$to\_header = $header;**

**$current = 'to\_header';**

**} elseif (strpos($header, 'Date:') === 0) {**

**$date\_header = $header;**

**$current = 'date\_header';**

**} else {**

**if (!empty($$current) && strpos($header, ' =?') === 0) {**

**$$current .= $header;**

**} else {**

**$current = '';**

**}**

**}**

**}**

**$from = str\_replace('|', '=7C', $this->DKIM\_QP($from\_header));**

**$to = str\_replace('|', '=7C', $this->DKIM\_QP($to\_header));**

**$date = str\_replace('|', '=7C', $this->DKIM\_QP($date\_header));**

**$subject = str\_replace(**

**'|',**

**'=7C',**

**$this->DKIM\_QP($subject\_header)**

**); // Copied header fields (dkim-quoted-printable)**

**$body = $this->DKIM\_BodyC($body);**

**$DKIMlen = strlen($body); // Length of body**

**$DKIMb64 = base64\_encode(pack('H\*', hash('sha256', $body))); // Base64 of packed binary SHA-256 hash of body**

**if ('' == $this->DKIM\_identity) {**

**$ident = '';**

**} else {**

**$ident = ' i=' . $this->DKIM\_identity . ';';**

**}**

**$dkimhdrs = 'DKIM-Signature: v=1; a=' .**

**$DKIMsignatureType . '; q=' .**

**$DKIMquery . '; l=' .**

**$DKIMlen . '; s=' .**

**$this->DKIM\_selector .**

**";\r\n" .**

**"\tt=" . $DKIMtime . '; c=' . $DKIMcanonicalization . ";\r\n" .**

**"\th=From:To:Date:Subject;\r\n" .**

**"\td=" . $this->DKIM\_domain . ';' . $ident . "\r\n" .**

**"\tz=$from\r\n" .**

**"\t|$to\r\n" .**

**"\t|$date\r\n" .**

**"\t|$subject;\r\n" .**

**"\tbh=" . $DKIMb64 . ";\r\n" .**

**"\tb=";**

**$toSign = $this->DKIM\_HeaderC(**

**$from\_header . "\r\n" .**

**$to\_header . "\r\n" .**

**$date\_header . "\r\n" .**

**$subject\_header . "\r\n" .**

**$dkimhdrs**

**);**

**$signed = $this->DKIM\_Sign($toSign);**

**return $dkimhdrs . $signed . "\r\n";**

**}**

**/\*\***

**\* Detect if a string contains a line longer than the maximum line length allowed.**

**\* @param string $str**

**\* @return boolean**

**\* @static**

**\*/**

**public static function hasLineLongerThanMax($str)**

**{**

**//+2 to include CRLF line break for a 1000 total**

**return (boolean)preg\_match('/^(.{'.(self::MAX\_LINE\_LENGTH + 2).',})/m', $str);**

**}**

**/\*\***

**\* Allows for public read access to 'to' property.**

**\* @note: Before the send() call, queued addresses (i.e. with IDN) are not yet included.**

**\* @access public**

**\* @return array**

**\*/**

**public function getToAddresses()**

**{**

**return $this->to;**

**}**

**/\*\***

**\* Allows for public read access to 'cc' property.**

**\* @note: Before the send() call, queued addresses (i.e. with IDN) are not yet included.**

**\* @access public**

**\* @return array**

**\*/**

**public function getCcAddresses()**

**{**

**return $this->cc;**

**}**

**/\*\***

**\* Allows for public read access to 'bcc' property.**

**\* @note: Before the send() call, queued addresses (i.e. with IDN) are not yet included.**

**\* @access public**

**\* @return array**

**\*/**

**public function getBccAddresses()**

**{**

**return $this->bcc;**

**}**

**/\*\***

**\* Allows for public read access to 'ReplyTo' property.**

**\* @note: Before the send() call, queued addresses (i.e. with IDN) are not yet included.**

**\* @access public**

**\* @return array**

**\*/**

**public function getReplyToAddresses()**

**{**

**return $this->ReplyTo;**

**}**

**/\*\***

**\* Allows for public read access to 'all\_recipients' property.**

**\* @note: Before the send() call, queued addresses (i.e. with IDN) are not yet included.**

**\* @access public**

**\* @return array**

**\*/**

**public function getAllRecipientAddresses()**

**{**

**return $this->all\_recipients;**

**}**

**/\*\***

**\* Perform a callback.**

**\* @param boolean $isSent**

**\* @param array $to**

**\* @param array $cc**

**\* @param array $bcc**

**\* @param string $subject**

**\* @param string $body**

**\* @param string $from**

**\*/**

**protected function doCallback($isSent, $to, $cc, $bcc, $subject, $body, $from)**

**{**

**if (!empty($this->action\_function) && is\_callable($this->action\_function)) {**

**$params = array($isSent, $to, $cc, $bcc, $subject, $body, $from);**

**call\_user\_func\_array($this->action\_function, $params);**

**}**

**}**

**}**

**/\*\***

**\* PHPMailer exception handler**

**\* @package PHPMailer**

**\*/**

**class phpmailerException extends Exception**

**{**

**/\*\***

**\* Prettify error message output**

**\* @return string**

**\*/**

**public function errorMessage()**

**{**

**$errorMsg = '<strong>' . htmlspecialchars($this->getMessage()) . "</strong><br />\n";**

**return $errorMsg;**

**}**

**}**

**code of class.smtp.php**

**<?php**

**/\*\***

**\* PHPMailer RFC821 SMTP email transport class.**

**\* PHP Version 5**

**\* @package PHPMailer**

**\* @link https://github.com/PHPMailer/PHPMailer/ The PHPMailer GitHub project**

**\* @author Marcus Bointon (Synchro/coolbru) <phpmailer@synchromedia.co.uk>**

**\* @author Jim Jagielski (jimjag) <jimjag@gmail.com>**

**\* @author Andy Prevost (codeworxtech) <codeworxtech@users.sourceforge.net>**

**\* @author Brent R. Matzelle (original founder)**

**\* @copyright 2014 Marcus Bointon**

**\* @copyright 2010 - 2012 Jim Jagielski**

**\* @copyright 2004 - 2009 Andy Prevost**

**\* @license http://www.gnu.org/copyleft/lesser.html GNU Lesser General Public License**

**\* @note This program is distributed in the hope that it will be useful - WITHOUT**

**\* ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or**

**\* FITNESS FOR A PARTICULAR PURPOSE.**

**\*/**

**/\*\***

**\* PHPMailer RFC821 SMTP email transport class.**

**\* Implements RFC 821 SMTP commands and provides some utility methods for sending mail to an SMTP server.**

**\* @package PHPMailer**

**\* @author Chris Ryan**

**\* @author Marcus Bointon <phpmailer@synchromedia.co.uk>**

**\*/**

**class SMTP**

**{**

**/\*\***

**\* The PHPMailer SMTP version number.**

**\* @var string**

**\*/**

**const VERSION = '5.2.28';**

**/\*\***

**\* SMTP line break constant.**

**\* @var string**

**\*/**

**const CRLF = "\r\n";**

**/\*\***

**\* The SMTP port to use if one is not specified.**

**\* @var integer**

**\*/**

**const DEFAULT\_SMTP\_PORT = 25;**

**/\*\***

**\* The maximum line length allowed by RFC 2822 section 2.1.1**

**\* @var integer**

**\*/**

**const MAX\_LINE\_LENGTH = 998;**

**/\*\***

**\* Debug level for no output**

**\*/**

**const DEBUG\_OFF = 0;**

**/\*\***

**\* Debug level to show client -> server messages**

**\*/**

**const DEBUG\_CLIENT = 1;**

**/\*\***

**\* Debug level to show client -> server and server -> client messages**

**\*/**

**const DEBUG\_SERVER = 2;**

**/\*\***

**\* Debug level to show connection status, client -> server and server -> client messages**

**\*/**

**const DEBUG\_CONNECTION = 3;**

**/\*\***

**\* Debug level to show all messages**

**\*/**

**const DEBUG\_LOWLEVEL = 4;**

**/\*\***

**\* The PHPMailer SMTP Version number.**

**\* @var string**

**\* @deprecated Use the `VERSION` constant instead**

**\* @see SMTP::VERSION**

**\*/**

**public $Version = '5.2.28';**

**/\*\***

**\* SMTP server port number.**

**\* @var integer**

**\* @deprecated This is only ever used as a default value, so use the `DEFAULT\_SMTP\_PORT` constant instead**

**\* @see SMTP::DEFAULT\_SMTP\_PORT**

**\*/**

**public $SMTP\_PORT = 25;**

**/\*\***

**\* SMTP reply line ending.**

**\* @var string**

**\* @deprecated Use the `CRLF` constant instead**

**\* @see SMTP::CRLF**

**\*/**

**public $CRLF = "\r\n";**

**/\*\***

**\* Debug output level.**

**\* Options:**

**\* \* self::DEBUG\_OFF (`0`) No debug output, default**

**\* \* self::DEBUG\_CLIENT (`1`) Client commands**

**\* \* self::DEBUG\_SERVER (`2`) Client commands and server responses**

**\* \* self::DEBUG\_CONNECTION (`3`) As DEBUG\_SERVER plus connection status**

**\* \* self::DEBUG\_LOWLEVEL (`4`) Low-level data output, all messages**

**\* @var integer**

**\*/**

**public $do\_debug = self::DEBUG\_OFF;**

**/\*\***

**\* How to handle debug output.**

**\* Options:**

**\* \* `echo` Output plain-text as-is, appropriate for CLI**

**\* \* `html` Output escaped, line breaks converted to `<br>`, appropriate for browser output**

**\* \* `error\_log` Output to error log as configured in php.ini**

**\***

**\* Alternatively, you can provide a callable expecting two params: a message string and the debug level:**

**\* <code>**

**\* $smtp->Debugoutput = function($str, $level) {echo "debug level $level; message: $str";};**

**\* </code>**

**\* @var string|callable**

**\*/**

**public $Debugoutput = 'echo';**

**/\*\***

**\* Whether to use VERP.**

**\* @link http://en.wikipedia.org/wiki/Variable\_envelope\_return\_path**

**\* @link http://www.postfix.org/VERP\_README.html Info on VERP**

**\* @var boolean**

**\*/**

**public $do\_verp = false;**

**/\*\***

**\* The timeout value for connection, in seconds.**

**\* Default of 5 minutes (300sec) is from RFC2821 section 4.5.3.2**

**\* This needs to be quite high to function correctly with hosts using greetdelay as an anti-spam measure.**

**\* @link http://tools.ietf.org/html/rfc2821#section-4.5.3.2**

**\* @var integer**

**\*/**

**public $Timeout = 300;**

**/\*\***

**\* How long to wait for commands to complete, in seconds.**

**\* Default of 5 minutes (300sec) is from RFC2821 section 4.5.3.2**

**\* @var integer**

**\*/**

**public $Timelimit = 300;**

**/\*\***

**\* @var array Patterns to extract an SMTP transaction id from reply to a DATA command.**

**\* The first capture group in each regex will be used as the ID.**

**\*/**

**protected $smtp\_transaction\_id\_patterns = array(**

**'exim' => '/[0-9]{3} OK id=(.\*)/',**

**'sendmail' => '/[0-9]{3} 2.0.0 (.\*) Message/',**

**'postfix' => '/[0-9]{3} 2.0.0 Ok: queued as (.\*)/'**

**);**

**/\*\***

**\* @var string The last transaction ID issued in response to a DATA command,**

**\* if one was detected**

**\*/**

**protected $last\_smtp\_transaction\_id;**

**/\*\***

**\* The socket for the server connection.**

**\* @var resource**

**\*/**

**protected $smtp\_conn;**

**/\*\***

**\* Error information, if any, for the last SMTP command.**

**\* @var array**

**\*/**

**protected $error = array(**

**'error' => '',**

**'detail' => '',**

**'smtp\_code' => '',**

**'smtp\_code\_ex' => ''**

**);**

**/\*\***

**\* The reply the server sent to us for HELO.**

**\* If null, no HELO string has yet been received.**

**\* @var string|null**

**\*/**

**protected $helo\_rply = null;**

**/\*\***

**\* The set of SMTP extensions sent in reply to EHLO command.**

**\* Indexes of the array are extension names.**

**\* Value at index 'HELO' or 'EHLO' (according to command that was sent)**

**\* represents the server name. In case of HELO it is the only element of the array.**

**\* Other values can be boolean TRUE or an array containing extension options.**

**\* If null, no HELO/EHLO string has yet been received.**

**\* @var array|null**

**\*/**

**protected $server\_caps = null;**

**/\*\***

**\* The most recent reply received from the server.**

**\* @var string**

**\*/**

**protected $last\_reply = '';**

**/\*\***

**\* Output debugging info via a user-selected method.**

**\* @see SMTP::$Debugoutput**

**\* @see SMTP::$do\_debug**

**\* @param string $str Debug string to output**

**\* @param integer $level The debug level of this message; see DEBUG\_\* constants**

**\* @return void**

**\*/**

**protected function edebug($str, $level = 0)**

**{**

**if ($level > $this->do\_debug) {**

**return;**

**}**

**//Avoid clash with built-in function names**

**if (!in\_array($this->Debugoutput, array('error\_log', 'html', 'echo')) and is\_callable($this->Debugoutput)) {**

**call\_user\_func($this->Debugoutput, $str, $level);**

**return;**

**}**

**switch ($this->Debugoutput) {**

**case 'error\_log':**

**//Don't output, just log**

**error\_log($str);**

**break;**

**case 'html':**

**//Cleans up output a bit for a better looking, HTML-safe output**

**echo gmdate('Y-m-d H:i:s') . ' ' . htmlentities(**

**preg\_replace('/[\r\n]+/', '', $str),**

**ENT\_QUOTES,**

**'UTF-8'**

**) . "<br>\n";**

**break;**

**case 'echo':**

**default:**

**//Normalize line breaks**

**$str = preg\_replace('/(\r\n|\r|\n)/ms', "\n", $str);**

**echo gmdate('Y-m-d H:i:s') . "\t" . str\_replace(**

**"\n",**

**"\n \t ",**

**trim($str)**

**) . "\n";**

**}**

**}**

**/\*\***

**\* Connect to an SMTP server.**

**\* @param string $host SMTP server IP or host name**

**\* @param integer $port The port number to connect to**

**\* @param integer $timeout How long to wait for the connection to open**

**\* @param array $options An array of options for stream\_context\_create()**

**\* @access public**

**\* @return boolean**

**\*/**

**public function connect($host, $port = null, $timeout = 30, $options = array())**

**{**

**static $streamok;**

**//This is enabled by default since 5.0.0 but some providers disable it**

**//Check this once and cache the result**

**if (is\_null($streamok)) {**

**$streamok = function\_exists('stream\_socket\_client');**

**}**

**// Clear errors to avoid confusion**

**$this->setError('');**

**// Make sure we are \_\_not\_\_ connected**

**if ($this->connected()) {**

**// Already connected, generate error**

**$this->setError('Already connected to a server');**

**return false;**

**}**

**if (empty($port)) {**

**$port = self::DEFAULT\_SMTP\_PORT;**

**}**

**// Connect to the SMTP server**

**$this->edebug(**

**"Connection: opening to $host:$port, timeout=$timeout, options=" .**

**var\_export($options, true),**

**self::DEBUG\_CONNECTION**

**);**

**$errno = 0;**

**$errstr = '';**

**if ($streamok) {**

**$socket\_context = stream\_context\_create($options);**

**set\_error\_handler(array($this, 'errorHandler'));**

**$this->smtp\_conn = stream\_socket\_client(**

**$host . ":" . $port,**

**$errno,**

**$errstr,**

**$timeout,**

**STREAM\_CLIENT\_CONNECT,**

**$socket\_context**

**);**

**restore\_error\_handler();**

**} else {**

**//Fall back to fsockopen which should work in more places, but is missing some features**

**$this->edebug(**

**"Connection: stream\_socket\_client not available, falling back to fsockopen",**

**self::DEBUG\_CONNECTION**

**);**

**set\_error\_handler(array($this, 'errorHandler'));**

**$this->smtp\_conn = fsockopen(**

**$host,**

**$port,**

**$errno,**

**$errstr,**

**$timeout**

**);**

**restore\_error\_handler();**

**}**

**// Verify we connected properly**

**if (!is\_resource($this->smtp\_conn)) {**

**$this->setError(**

**'Failed to connect to server',**

**$errno,**

**$errstr**

**);**

**$this->edebug(**

**'SMTP ERROR: ' . $this->error['error']**

**. ": $errstr ($errno)",**

**self::DEBUG\_CLIENT**

**);**

**return false;**

**}**

**$this->edebug('Connection: opened', self::DEBUG\_CONNECTION);**

**// SMTP server can take longer to respond, give longer timeout for first read**

**// Windows does not have support for this timeout function**

**if (substr(PHP\_OS, 0, 3) != 'WIN') {**

**$max = ini\_get('max\_execution\_time');**

**// Don't bother if unlimited**

**if ($max != 0 && $timeout > $max) {**

**@set\_time\_limit($timeout);**

**}**

**stream\_set\_timeout($this->smtp\_conn, $timeout, 0);**

**}**

**// Get any announcement**

**$announce = $this->get\_lines();**

**$this->edebug('SERVER -> CLIENT: ' . $announce, self::DEBUG\_SERVER);**

**return true;**

**}**

**/\*\***

**\* Initiate a TLS (encrypted) session.**

**\* @access public**

**\* @return boolean**

**\*/**

**public function startTLS()**

**{**

**if (!$this->sendCommand('STARTTLS', 'STARTTLS', 220)) {**

**return false;**

**}**

**//Allow the best TLS version(s) we can**

**$crypto\_method = STREAM\_CRYPTO\_METHOD\_TLS\_CLIENT;**

**//PHP 5.6.7 dropped inclusion of TLS 1.1 and 1.2 in STREAM\_CRYPTO\_METHOD\_TLS\_CLIENT**

**//so add them back in manually if we can**

**if (defined('STREAM\_CRYPTO\_METHOD\_TLSv1\_2\_CLIENT')) {**

**$crypto\_method |= STREAM\_CRYPTO\_METHOD\_TLSv1\_2\_CLIENT;**

**$crypto\_method |= STREAM\_CRYPTO\_METHOD\_TLSv1\_1\_CLIENT;**

**}**

**// Begin encrypted connection**

**set\_error\_handler(array($this, 'errorHandler'));**

**$crypto\_ok = stream\_socket\_enable\_crypto(**

**$this->smtp\_conn,**

**true,**

**$crypto\_method**

**);**

**restore\_error\_handler();**

**return $crypto\_ok;**

**}**

**/\*\***

**\* Perform SMTP authentication.**

**\* Must be run after hello().**

**\* @see hello()**

**\* @param string $username The user name**

**\* @param string $password The password**

**\* @param string $authtype The auth type (PLAIN, LOGIN, NTLM, CRAM-MD5, XOAUTH2)**

**\* @param string $realm The auth realm for NTLM**

**\* @param string $workstation The auth workstation for NTLM**

**\* @param null|OAuth $OAuth An optional OAuth instance (@see PHPMailerOAuth)**

**\* @return bool True if successfully authenticated.\* @access public**

**\*/**

**public function authenticate(**

**$username,**

**$password,**

**$authtype = null,**

**$realm = '',**

**$workstation = '',**

**$OAuth = null**

**) {**

**if (!$this->server\_caps) {**

**$this->setError('Authentication is not allowed before HELO/EHLO');**

**return false;**

**}**

**if (array\_key\_exists('EHLO', $this->server\_caps)) {**

**// SMTP extensions are available; try to find a proper authentication method**

**if (!array\_key\_exists('AUTH', $this->server\_caps)) {**

**$this->setError('Authentication is not allowed at this stage');**

**// 'at this stage' means that auth may be allowed after the stage changes**

**// e.g. after STARTTLS**

**return false;**

**}**

**self::edebug('Auth method requested: ' . ($authtype ? $authtype : 'UNKNOWN'), self::DEBUG\_LOWLEVEL);**

**self::edebug(**

**'Auth methods available on the server: ' . implode(',', $this->server\_caps['AUTH']),**

**self::DEBUG\_LOWLEVEL**

**);**

**if (empty($authtype)) {**

**foreach (array('CRAM-MD5', 'LOGIN', 'PLAIN', 'NTLM', 'XOAUTH2') as $method) {**

**if (in\_array($method, $this->server\_caps['AUTH'])) {**

**$authtype = $method;**

**break;**

**}**

**}**

**if (empty($authtype)) {**

**$this->setError('No supported authentication methods found');**

**return false;**

**}**

**self::edebug('Auth method selected: ' . $authtype, self::DEBUG\_LOWLEVEL);**

**}**

**if (!in\_array($authtype, $this->server\_caps['AUTH'])) {**

**$this->setError("The requested authentication method \"$authtype\" is not supported by the server");**

**return false;**

**}**

**} elseif (empty($authtype)) {**

**$authtype = 'LOGIN';**

**}**

**switch ($authtype) {**

**case 'PLAIN':**

**// Start authentication**

**if (!$this->sendCommand('AUTH', 'AUTH PLAIN', 334)) {**

**return false;**

**}**

**// Send encoded username and password**

**if (!$this->sendCommand(**

**'User & Password',**

**base64\_encode("\0" . $username . "\0" . $password),**

**235**

**)**

**) {**

**return false;**

**}**

**break;**

**case 'LOGIN':**

**// Start authentication**

**if (!$this->sendCommand('AUTH', 'AUTH LOGIN', 334)) {**

**return false;**

**}**

**if (!$this->sendCommand("Username", base64\_encode($username), 334)) {**

**return false;**

**}**

**if (!$this->sendCommand("Password", base64\_encode($password), 235)) {**

**return false;**

**}**

**break;**

**case 'XOAUTH2':**

**//If the OAuth Instance is not set. Can be a case when PHPMailer is used**

**//instead of PHPMailerOAuth**

**if (is\_null($OAuth)) {**

**return false;**

**}**

**$oauth = $OAuth->getOauth64();**

**// Start authentication**

**if (!$this->sendCommand('AUTH', 'AUTH XOAUTH2 ' . $oauth, 235)) {**

**return false;**

**}**

**break;**

**case 'NTLM':**

**/\***

**\* ntlm\_sasl\_client.php**

**\* Bundled with Permission**

**\***

**\* How to telnet in windows:**

**\* http://technet.microsoft.com/en-us/library/aa995718%28EXCHG.65%29.aspx**

**\* PROTOCOL Docs http://curl.haxx.se/rfc/ntlm.html#ntlmSmtpAuthentication**

**\*/**

**require\_once 'extras/ntlm\_sasl\_client.php';**

**$temp = new stdClass;**

**$ntlm\_client = new ntlm\_sasl\_client\_class;**

**//Check that functions are available**

**if (!$ntlm\_client->initialize($temp)) {**

**$this->setError($temp->error);**

**$this->edebug(**

**'You need to enable some modules in your php.ini file: '**

**. $this->error['error'],**

**self::DEBUG\_CLIENT**

**);**

**return false;**

**}**

**//msg1**

**$msg1 = $ntlm\_client->typeMsg1($realm, $workstation); //msg1**

**if (!$this->sendCommand(**

**'AUTH NTLM',**

**'AUTH NTLM ' . base64\_encode($msg1),**

**334**

**)**

**) {**

**return false;**

**}**

**//Though 0 based, there is a white space after the 3 digit number**

**//msg2**

**$challenge = substr($this->last\_reply, 3);**

**$challenge = base64\_decode($challenge);**

**$ntlm\_res = $ntlm\_client->NTLMResponse(**

**substr($challenge, 24, 8),**

**$password**

**);**

**//msg3**

**$msg3 = $ntlm\_client->typeMsg3(**

**$ntlm\_res,**

**$username,**

**$realm,**

**$workstation**

**);**

**// send encoded username**

**return $this->sendCommand('Username', base64\_encode($msg3), 235);**

**case 'CRAM-MD5':**

**// Start authentication**

**if (!$this->sendCommand('AUTH CRAM-MD5', 'AUTH CRAM-MD5', 334)) {**

**return false;**

**}**

**// Get the challenge**

**$challenge = base64\_decode(substr($this->last\_reply, 4));**

**// Build the response**

**$response = $username . ' ' . $this->hmac($challenge, $password);**

**// send encoded credentials**

**return $this->sendCommand('Username', base64\_encode($response), 235);**

**default:**

**$this->setError("Authentication method \"$authtype\" is not supported");**

**return false;**

**}**

**return true;**

**}**

**/\*\***

**\* Calculate an MD5 HMAC hash.**

**\* Works like hash\_hmac('md5', $data, $key)**

**\* in case that function is not available**

**\* @param string $data The data to hash**

**\* @param string $key The key to hash with**

**\* @access protected**

**\* @return string**

**\*/**

**protected function hmac($data, $key)**

**{**

**if (function\_exists('hash\_hmac')) {**

**return hash\_hmac('md5', $data, $key);**

**}**

**// The following borrowed from**

**// http://php.net/manual/en/function.mhash.php#27225**

**// RFC 2104 HMAC implementation for php.**

**// Creates an md5 HMAC.**

**// Eliminates the need to install mhash to compute a HMAC**

**// by Lance Rushing**

**$bytelen = 64; // byte length for md5**

**if (strlen($key) > $bytelen) {**

**$key = pack('H\*', md5($key));**

**}**

**$key = str\_pad($key, $bytelen, chr(0x00));**

**$ipad = str\_pad('', $bytelen, chr(0x36));**

**$opad = str\_pad('', $bytelen, chr(0x5c));**

**$k\_ipad = $key ^ $ipad;**

**$k\_opad = $key ^ $opad;**

**return md5($k\_opad . pack('H\*', md5($k\_ipad . $data)));**

**}**

**/\*\***

**\* Check connection state.**

**\* @access public**

**\* @return boolean True if connected.**

**\*/**

**public function connected()**

**{**

**if (is\_resource($this->smtp\_conn)) {**

**$sock\_status = stream\_get\_meta\_data($this->smtp\_conn);**

**if ($sock\_status['eof']) {**

**// The socket is valid but we are not connected**

**$this->edebug(**

**'SMTP NOTICE: EOF caught while checking if connected',**

**self::DEBUG\_CLIENT**

**);**

**$this->close();**

**return false;**

**}**

**return true; // everything looks good**

**}**

**return false;**

**}**

**/\*\***

**\* Close the socket and clean up the state of the class.**

**\* Don't use this function without first trying to use QUIT.**

**\* @see quit()**

**\* @access public**

**\* @return void**

**\*/**

**public function close()**

**{**

**$this->setError('');**

**$this->server\_caps = null;**

**$this->helo\_rply = null;**

**if (is\_resource($this->smtp\_conn)) {**

**// close the connection and cleanup**

**fclose($this->smtp\_conn);**

**$this->smtp\_conn = null; //Makes for cleaner serialization**

**$this->edebug('Connection: closed', self::DEBUG\_CONNECTION);**

**}**

**}**

**/\*\***

**\* Send an SMTP DATA command.**

**\* Issues a data command and sends the msg\_data to the server,**

**\* finializing the mail transaction. $msg\_data is the message**

**\* that is to be send with the headers. Each header needs to be**

**\* on a single line followed by a <CRLF> with the message headers**

**\* and the message body being separated by and additional <CRLF>.**

**\* Implements rfc 821: DATA <CRLF>**

**\* @param string $msg\_data Message data to send**

**\* @access public**

**\* @return boolean**

**\*/**

**public function data($msg\_data)**

**{**

**//This will use the standard timelimit**

**if (!$this->sendCommand('DATA', 'DATA', 354)) {**

**return false;**

**}**

**/\* The server is ready to accept data!**

**\* According to rfc821 we should not send more than 1000 characters on a single line (including the CRLF)**

**\* so we will break the data up into lines by \r and/or \n then if needed we will break each of those into**

**\* smaller lines to fit within the limit.**

**\* We will also look for lines that start with a '.' and prepend an additional '.'.**

**\* NOTE: this does not count towards line-length limit.**

**\*/**

**// Normalize line breaks before exploding**

**$lines = explode("\n", str\_replace(array("\r\n", "\r"), "\n", $msg\_data));**

**/\* To distinguish between a complete RFC822 message and a plain message body, we check if the first field**

**\* of the first line (':' separated) does not contain a space then it \_should\_ be a header and we will**

**\* process all lines before a blank line as headers.**

**\*/**

**$field = substr($lines[0], 0, strpos($lines[0], ':'));**

**$in\_headers = false;**

**if (!empty($field) && strpos($field, ' ') === false) {**

**$in\_headers = true;**

**}**

**foreach ($lines as $line) {**

**$lines\_out = array();**

**if ($in\_headers and $line == '') {**

**$in\_headers = false;**

**}**

**//Break this line up into several smaller lines if it's too long**

**//Micro-optimisation: isset($str[$len]) is faster than (strlen($str) > $len),**

**while (isset($line[self::MAX\_LINE\_LENGTH])) {**

**//Working backwards, try to find a space within the last MAX\_LINE\_LENGTH chars of the line to break on**

**//so as to avoid breaking in the middle of a word**

**$pos = strrpos(substr($line, 0, self::MAX\_LINE\_LENGTH), ' ');**

**//Deliberately matches both false and 0**

**if (!$pos) {**

**//No nice break found, add a hard break**

**$pos = self::MAX\_LINE\_LENGTH - 1;**

**$lines\_out[] = substr($line, 0, $pos);**

**$line = substr($line, $pos);**

**} else {**

**//Break at the found point**

**$lines\_out[] = substr($line, 0, $pos);**

**//Move along by the amount we dealt with**

**$line = substr($line, $pos + 1);**

**}**

**//If processing headers add a LWSP-char to the front of new line RFC822 section 3.1.1**

**if ($in\_headers) {**

**$line = "\t" . $line;**

**}**

**}**

**$lines\_out[] = $line;**

**//Send the lines to the server**

**foreach ($lines\_out as $line\_out) {**

**//RFC2821 section 4.5.2**

**if (!empty($line\_out) and $line\_out[0] == '.') {**

**$line\_out = '.' . $line\_out;**

**}**

**$this->client\_send($line\_out . self::CRLF);**

**}**

**}**

**//Message data has been sent, complete the command**

**//Increase timelimit for end of DATA command**

**$savetimelimit = $this->Timelimit;**

**$this->Timelimit = $this->Timelimit \* 2;**

**$result = $this->sendCommand('DATA END', '.', 250);**

**$this->recordLastTransactionID();**

**//Restore timelimit**

**$this->Timelimit = $savetimelimit;**

**return $result;**

**}**

**/\*\***

**\* Send an SMTP HELO or EHLO command.**

**\* Used to identify the sending server to the receiving server.**

**\* This makes sure that client and server are in a known state.**

**\* Implements RFC 821: HELO <SP> <domain> <CRLF>**

**\* and RFC 2821 EHLO.**

**\* @param string $host The host name or IP to connect to**

**\* @access public**

**\* @return boolean**

**\*/**

**public function hello($host = '')**

**{**

**//Try extended hello first (RFC 2821)**

**return (boolean)($this->sendHello('EHLO', $host) or $this->sendHello('HELO', $host));**

**}**

**/\*\***

**\* Send an SMTP HELO or EHLO command.**

**\* Low-level implementation used by hello()**

**\* @see hello()**

**\* @param string $hello The HELO string**

**\* @param string $host The hostname to say we are**

**\* @access protected**

**\* @return boolean**

**\*/**

**protected function sendHello($hello, $host)**

**{**

**$noerror = $this->sendCommand($hello, $hello . ' ' . $host, 250);**

**$this->helo\_rply = $this->last\_reply;**

**if ($noerror) {**

**$this->parseHelloFields($hello);**

**} else {**

**$this->server\_caps = null;**

**}**

**return $noerror;**

**}**

**/\*\***

**\* Parse a reply to HELO/EHLO command to discover server extensions.**

**\* In case of HELO, the only parameter that can be discovered is a server name.**

**\* @access protected**

**\* @param string $type - 'HELO' or 'EHLO'**

**\*/**

**protected function parseHelloFields($type)**

**{**

**$this->server\_caps = array();**

**$lines = explode("\n", $this->helo\_rply);**

**foreach ($lines as $n => $s) {**

**//First 4 chars contain response code followed by - or space**

**$s = trim(substr($s, 4));**

**if (empty($s)) {**

**continue;**

**}**

**$fields = explode(' ', $s);**

**if (!empty($fields)) {**

**if (!$n) {**

**$name = $type;**

**$fields = $fields[0];**

**} else {**

**$name = array\_shift($fields);**

**switch ($name) {**

**case 'SIZE':**

**$fields = ($fields ? $fields[0] : 0);**

**break;**

**case 'AUTH':**

**if (!is\_array($fields)) {**

**$fields = array();**

**}**

**break;**

**default:**

**$fields = true;**

**}**

**}**

**$this->server\_caps[$name] = $fields;**

**}**

**}**

**}**

**/\*\***

**\* Send an SMTP MAIL command.**

**\* Starts a mail transaction from the email address specified in**

**\* $from. Returns true if successful or false otherwise. If True**

**\* the mail transaction is started and then one or more recipient**

**\* commands may be called followed by a data command.**

**\* Implements rfc 821: MAIL <SP> FROM:<reverse-path> <CRLF>**

**\* @param string $from Source address of this message**

**\* @access public**

**\* @return boolean**

**\*/**

**public function mail($from)**

**{**

**$useVerp = ($this->do\_verp ? ' XVERP' : '');**

**return $this->sendCommand(**

**'MAIL FROM',**

**'MAIL FROM:<' . $from . '>' . $useVerp,**

**250**

**);**

**}**

**/\*\***

**\* Send an SMTP QUIT command.**

**\* Closes the socket if there is no error or the $close\_on\_error argument is true.**

**\* Implements from rfc 821: QUIT <CRLF>**

**\* @param boolean $close\_on\_error Should the connection close if an error occurs?**

**\* @access public**

**\* @return boolean**

**\*/**

**public function quit($close\_on\_error = true)**

**{**

**$noerror = $this->sendCommand('QUIT', 'QUIT', 221);**

**$err = $this->error; //Save any error**

**if ($noerror or $close\_on\_error) {**

**$this->close();**

**$this->error = $err; //Restore any error from the quit command**

**}**

**return $noerror;**

**}**

**/\*\***

**\* Send an SMTP RCPT command.**

**\* Sets the TO argument to $toaddr.**

**\* Returns true if the recipient was accepted false if it was rejected.**

**\* Implements from rfc 821: RCPT <SP> TO:<forward-path> <CRLF>**

**\* @param string $address The address the message is being sent to**

**\* @access public**

**\* @return boolean**

**\*/**

**public function recipient($address)**

**{**

**return $this->sendCommand(**

**'RCPT TO',**

**'RCPT TO:<' . $address . '>',**

**array(250, 251)**

**);**

**}**

**/\*\***

**\* Send an SMTP RSET command.**

**\* Abort any transaction that is currently in progress.**

**\* Implements rfc 821: RSET <CRLF>**

**\* @access public**

**\* @return boolean True on success.**

**\*/**

**public function reset()**

**{**

**return $this->sendCommand('RSET', 'RSET', 250);**

**}**

**/\*\***

**\* Send a command to an SMTP server and check its return code.**

**\* @param string $command The command name - not sent to the server**

**\* @param string $commandstring The actual command to send**

**\* @param integer|array $expect One or more expected integer success codes**

**\* @access protected**

**\* @return boolean True on success.**

**\*/**

**protected function sendCommand($command, $commandstring, $expect)**

**{**

**if (!$this->connected()) {**

**$this->setError("Called $command without being connected");**

**return false;**

**}**

**//Reject line breaks in all commands**

**if (strpos($commandstring, "\n") !== false or strpos($commandstring, "\r") !== false) {**

**$this->setError("Command '$command' contained line breaks");**

**return false;**

**}**

**$this->client\_send($commandstring . self::CRLF);**

**$this->last\_reply = $this->get\_lines();**

**// Fetch SMTP code and possible error code explanation**

**$matches = array();**

**if (preg\_match("/^([0-9]{3})[ -](?:([0-9]\\.[0-9]\\.[0-9]) )?/", $this->last\_reply, $matches)) {**

**$code = $matches[1];**

**$code\_ex = (count($matches) > 2 ? $matches[2] : null);**

**// Cut off error code from each response line**

**$detail = preg\_replace(**

**"/{$code}[ -]" .**

**($code\_ex ? str\_replace('.', '\\.', $code\_ex) . ' ' : '') . "/m",**

**'',**

**$this->last\_reply**

**);**

**} else {**

**// Fall back to simple parsing if regex fails**

**$code = substr($this->last\_reply, 0, 3);**

**$code\_ex = null;**

**$detail = substr($this->last\_reply, 4);**

**}**

**$this->edebug('SERVER -> CLIENT: ' . $this->last\_reply, self::DEBUG\_SERVER);**

**if (!in\_array($code, (array)$expect)) {**

**$this->setError(**

**"$command command failed",**

**$detail,**

**$code,**

**$code\_ex**

**);**

**$this->edebug(**

**'SMTP ERROR: ' . $this->error['error'] . ': ' . $this->last\_reply,**

**self::DEBUG\_CLIENT**

**);**

**return false;**

**}**

**$this->setError('');**

**return true;**

**}**

**/\*\***

**\* Send an SMTP SAML command.**

**\* Starts a mail transaction from the email address specified in $from.**

**\* Returns true if successful or false otherwise. If True**

**\* the mail transaction is started and then one or more recipient**

**\* commands may be called followed by a data command. This command**

**\* will send the message to the users terminal if they are logged**

**\* in and send them an email.**

**\* Implements rfc 821: SAML <SP> FROM:<reverse-path> <CRLF>**

**\* @param string $from The address the message is from**

**\* @access public**

**\* @return boolean**

**\*/**

**public function sendAndMail($from)**

**{**

**return $this->sendCommand('SAML', "SAML FROM:$from", 250);**

**}**

**/\*\***

**\* Send an SMTP VRFY command.**

**\* @param string $name The name to verify**

**\* @access public**

**\* @return boolean**

**\*/**

**public function verify($name)**

**{**

**return $this->sendCommand('VRFY', "VRFY $name", array(250, 251));**

**}**

**/\*\***

**\* Send an SMTP NOOP command.**

**\* Used to keep keep-alives alive, doesn't actually do anything**

**\* @access public**

**\* @return boolean**

**\*/**

**public function noop()**

**{**

**return $this->sendCommand('NOOP', 'NOOP', 250);**

**}**

**/\*\***

**\* Send an SMTP TURN command.**

**\* This is an optional command for SMTP that this class does not support.**

**\* This method is here to make the RFC821 Definition complete for this class**

**\* and \_may\_ be implemented in future**

**\* Implements from rfc 821: TURN <CRLF>**

**\* @access public**

**\* @return boolean**

**\*/**

**public function turn()**

**{**

**$this->setError('The SMTP TURN command is not implemented');**

**$this->edebug('SMTP NOTICE: ' . $this->error['error'], self::DEBUG\_CLIENT);**

**return false;**

**}**

**/\*\***

**\* Send raw data to the server.**

**\* @param string $data The data to send**

**\* @access public**

**\* @return integer|boolean The number of bytes sent to the server or false on error**

**\*/**

**public function client\_send($data)**

**{**

**$this->edebug("CLIENT -> SERVER: $data", self::DEBUG\_CLIENT);**

**set\_error\_handler(array($this, 'errorHandler'));**

**$result = fwrite($this->smtp\_conn, $data);**

**restore\_error\_handler();**

**return $result;**

**}**

**/\*\***

**\* Get the latest error.**

**\* @access public**

**\* @return array**

**\*/**

**public function getError()**

**{**

**return $this->error;**

**}**

**/\*\***

**\* Get SMTP extensions available on the server**

**\* @access public**

**\* @return array|null**

**\*/**

**public function getServerExtList()**

**{**

**return $this->server\_caps;**

**}**

**/\*\***

**\* A multipurpose method**

**\* The method works in three ways, dependent on argument value and current state**

**\* 1. HELO/EHLO was not sent - returns null and set up $this->error**

**\* 2. HELO was sent**

**\* $name = 'HELO': returns server name**

**\* $name = 'EHLO': returns boolean false**

**\* $name = any string: returns null and set up $this->error**

**\* 3. EHLO was sent**

**\* $name = 'HELO'|'EHLO': returns server name**

**\* $name = any string: if extension $name exists, returns boolean True**

**\* or its options. Otherwise returns boolean False**

**\* In other words, one can use this method to detect 3 conditions:**

**\* - null returned: handshake was not or we don't know about ext (refer to $this->error)**

**\* - false returned: the requested feature exactly not exists**

**\* - positive value returned: the requested feature exists**

**\* @param string $name Name of SMTP extension or 'HELO'|'EHLO'**

**\* @return mixed**

**\*/**

**public function getServerExt($name)**

**{**

**if (!$this->server\_caps) {**

**$this->setError('No HELO/EHLO was sent');**

**return null;**

**}**

**// the tight logic knot ;)**

**if (!array\_key\_exists($name, $this->server\_caps)) {**

**if ($name == 'HELO') {**

**return $this->server\_caps['EHLO'];**

**}**

**if ($name == 'EHLO' || array\_key\_exists('EHLO', $this->server\_caps)) {**

**return false;**

**}**

**$this->setError('HELO handshake was used. Client knows nothing about server extensions');**

**return null;**

**}**

**return $this->server\_caps[$name];**

**}**

**/\*\***

**\* Get the last reply from the server.**

**\* @access public**

**\* @return string**

**\*/**

**public function getLastReply()**

**{**

**return $this->last\_reply;**

**}**

**/\*\***

**\* Read the SMTP server's response.**

**\* Either before eof or socket timeout occurs on the operation.**

**\* With SMTP we can tell if we have more lines to read if the**

**\* 4th character is '-' symbol. If it is a space then we don't**

**\* need to read anything else.**

**\* @access protected**

**\* @return string**

**\*/**

**protected function get\_lines()**

**{**

**// If the connection is bad, give up straight away**

**if (!is\_resource($this->smtp\_conn)) {**

**return '';**

**}**

**$data = '';**

**$endtime = 0;**

**stream\_set\_timeout($this->smtp\_conn, $this->Timeout);**

**if ($this->Timelimit > 0) {**

**$endtime = time() + $this->Timelimit;**

**}**

**while (is\_resource($this->smtp\_conn) && !feof($this->smtp\_conn)) {**

**$str = @fgets($this->smtp\_conn, 515);**

**$this->edebug("SMTP -> get\_lines(): \$data is \"$data\"", self::DEBUG\_LOWLEVEL);**

**$this->edebug("SMTP -> get\_lines(): \$str is \"$str\"", self::DEBUG\_LOWLEVEL);**

**$data .= $str;**

**// If response is only 3 chars (not valid, but RFC5321 S4.2 says it must be handled),**

**// or 4th character is a space, we are done reading, break the loop,**

**// string array access is a micro-optimisation over strlen**

**if (!isset($str[3]) or (isset($str[3]) and $str[3] == ' ')) {**

**break;**

**}**

**// Timed-out? Log and break**

**$info = stream\_get\_meta\_data($this->smtp\_conn);**

**if ($info['timed\_out']) {**

**$this->edebug(**

**'SMTP -> get\_lines(): timed-out (' . $this->Timeout . ' sec)',**

**self::DEBUG\_LOWLEVEL**

**);**

**break;**

**}**

**// Now check if reads took too long**

**if ($endtime and time() > $endtime) {**

**$this->edebug(**

**'SMTP -> get\_lines(): timelimit reached (' .**

**$this->Timelimit . ' sec)',**

**self::DEBUG\_LOWLEVEL**

**);**

**break;**

**}**

**}**

**return $data;**

**}**

**/\*\***

**\* Enable or disable VERP address generation.**

**\* @param boolean $enabled**

**\*/**

**public function setVerp($enabled = false)**

**{**

**$this->do\_verp = $enabled;**

**}**

**/\*\***

**\* Get VERP address generation mode.**

**\* @return boolean**

**\*/**

**public function getVerp()**

**{**

**return $this->do\_verp;**

**}**

**/\*\***

**\* Set error messages and codes.**

**\* @param string $message The error message**

**\* @param string $detail Further detail on the error**

**\* @param string $smtp\_code An associated SMTP error code**

**\* @param string $smtp\_code\_ex Extended SMTP code**

**\*/**

**protected function setError($message, $detail = '', $smtp\_code = '', $smtp\_code\_ex = '')**

**{**

**$this->error = array(**

**'error' => $message,**

**'detail' => $detail,**

**'smtp\_code' => $smtp\_code,**

**'smtp\_code\_ex' => $smtp\_code\_ex**

**);**

**}**

**/\*\***

**\* Set debug output method.**

**\* @param string|callable $method The name of the mechanism to use for debugging output, or a callable to handle it.**

**\*/**

**public function setDebugOutput($method = 'echo')**

**{**

**$this->Debugoutput = $method;**

**}**

**/\*\***

**\* Get debug output method.**

**\* @return string**

**\*/**

**public function getDebugOutput()**

**{**

**return $this->Debugoutput;**

**}**

**/\*\***

**\* Set debug output level.**

**\* @param integer $level**

**\*/**

**public function setDebugLevel($level = 0)**

**{**

**$this->do\_debug = $level;**

**}**

**/\*\***

**\* Get debug output level.**

**\* @return integer**

**\*/**

**public function getDebugLevel()**

**{**

**return $this->do\_debug;**

**}**

**/\*\***

**\* Set SMTP timeout.**

**\* @param integer $timeout**

**\*/**

**public function setTimeout($timeout = 0)**

**{**

**$this->Timeout = $timeout;**

**}**

**/\*\***

**\* Get SMTP timeout.**

**\* @return integer**

**\*/**

**public function getTimeout()**

**{**

**return $this->Timeout;**

**}**

**/\*\***

**\* Reports an error number and string.**

**\* @param integer $errno The error number returned by PHP.**

**\* @param string $errmsg The error message returned by PHP.**

**\* @param string $errfile The file the error occurred in**

**\* @param integer $errline The line number the error occurred on**

**\*/**

**protected function errorHandler($errno, $errmsg, $errfile = '', $errline = 0)**

**{**

**$notice = 'Connection failed.';**

**$this->setError(**

**$notice,**

**$errno,**

**$errmsg**

**);**

**$this->edebug(**

**$notice . ' Error #' . $errno . ': ' . $errmsg . " [$errfile line $errline]",**

**self::DEBUG\_CONNECTION**

**);**

**}**

**/\*\***

**\* Extract and return the ID of the last SMTP transaction based on**

**\* a list of patterns provided in SMTP::$smtp\_transaction\_id\_patterns.**

**\* Relies on the host providing the ID in response to a DATA command.**

**\* If no reply has been received yet, it will return null.**

**\* If no pattern was matched, it will return false.**

**\* @return bool|null|string**

**\*/**

**protected function recordLastTransactionID()**

**{**

**$reply = $this->getLastReply();**

**if (empty($reply)) {**

**$this->last\_smtp\_transaction\_id = null;**

**} else {**

**$this->last\_smtp\_transaction\_id = false;**

**foreach ($this->smtp\_transaction\_id\_patterns as $smtp\_transaction\_id\_pattern) {**

**if (preg\_match($smtp\_transaction\_id\_pattern, $reply, $matches)) {**

**$this->last\_smtp\_transaction\_id = $matches[1];**

**}**

**}**

**}**

**return $this->last\_smtp\_transaction\_id;**

**}**

**/\*\***

**\* Get the queue/transaction ID of the last SMTP transaction**

**\* If no reply has been received yet, it will return null.**

**\* If no pattern was matched, it will return false.**

**\* @return bool|null|string**

**\* @see recordLastTransactionID()**

**\*/**

**public function getLastTransactionID()**

**{**

**return $this->last\_smtp\_transaction\_id;**

**}**

**}**

**code of PHPMailerAutoload.php**

**<?php**

**/\*\***

**\* PHPMailer SPL autoloader.**

**\* PHP Version 5**

**\* @package PHPMailer**

**\* @link https://github.com/PHPMailer/PHPMailer/ The PHPMailer GitHub project**

**\* @author Marcus Bointon (Synchro/coolbru) <phpmailer@synchromedia.co.uk>**

**\* @author Jim Jagielski (jimjag) <jimjag@gmail.com>**

**\* @author Andy Prevost (codeworxtech) <codeworxtech@users.sourceforge.net>**

**\* @author Brent R. Matzelle (original founder)**

**\* @copyright 2012 - 2014 Marcus Bointon**

**\* @copyright 2010 - 2012 Jim Jagielski**

**\* @copyright 2004 - 2009 Andy Prevost**

**\* @license http://www.gnu.org/copyleft/lesser.html GNU Lesser General Public License**

**\* @note This program is distributed in the hope that it will be useful - WITHOUT**

**\* ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or**

**\* FITNESS FOR A PARTICULAR PURPOSE.**

**\*/**

**/\*\***

**\* PHPMailer SPL autoloader.**

**\* @param string $classname The name of the class to load**

**\*/**

**function PHPMailerAutoload($classname)**

**{**

**//Can't use \_\_DIR\_\_ as it's only in PHP 5.3+**

**$filename = dirname(\_\_FILE\_\_).DIRECTORY\_SEPARATOR.'class.'.strtolower($classname).'.php';**

**if (is\_readable($filename)) {**

**require $filename;**

**}**

**}**

**if (version\_compare(PHP\_VERSION, '5.1.2', '>=')) {**

**//SPL autoloading was introduced in PHP 5.1.2**

**if (version\_compare(PHP\_VERSION, '5.3.0', '>=')) {**

**spl\_autoload\_register('PHPMailerAutoload', true, true);**

**} else {**

**spl\_autoload\_register('PHPMailerAutoload');**

**}**

**} else {**

**/\*\***

**\* Fall back to traditional autoload for old PHP versions**

**\* @param string $classname The name of the class to load**

**\*/**

**function \_\_autoload($classname)**

**{**

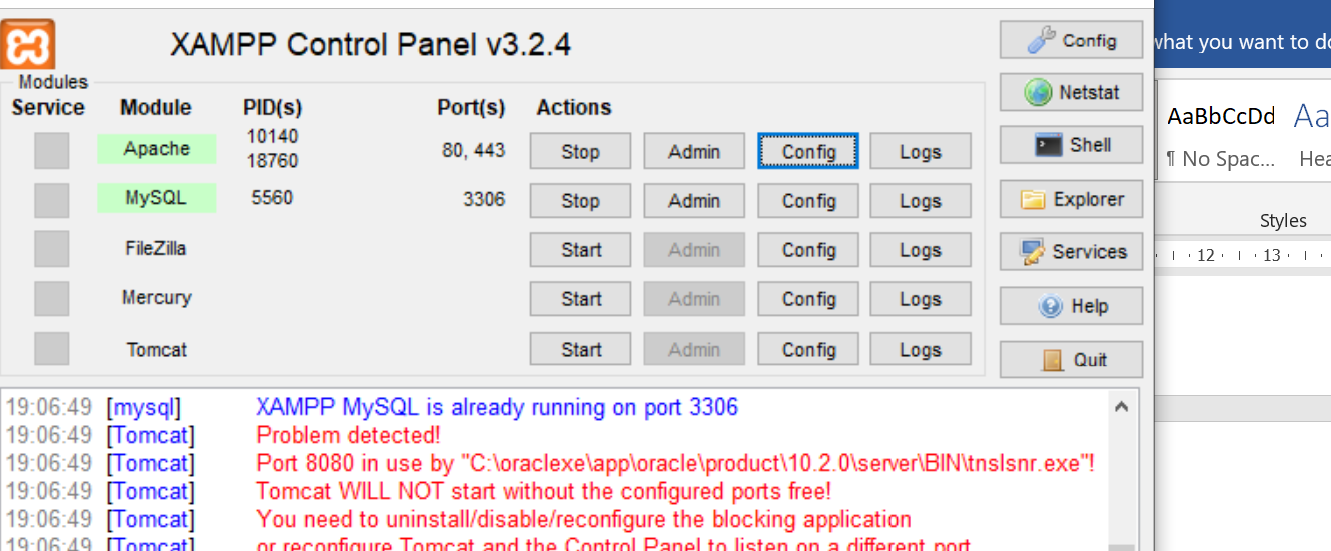
**PHPMailerAutoload($classname);**

**}**

**}**

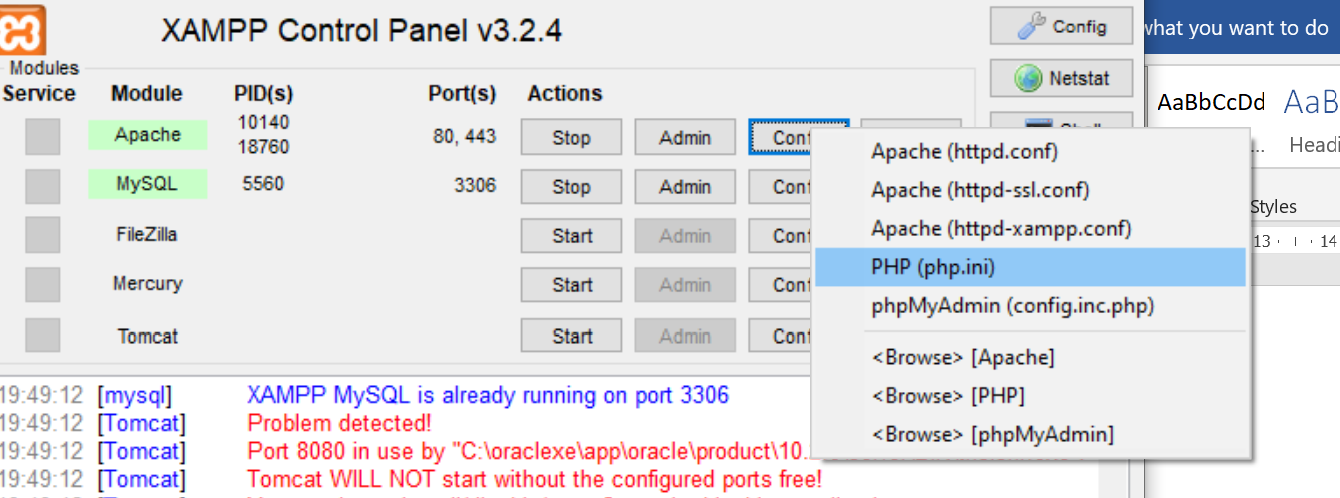
**6:Now the open xampp server application**

**and click right button Config pannel**

****

**and after his**

**click php(php.ini)**

****

**And last is Configuration inside the PHP( php.ini)**

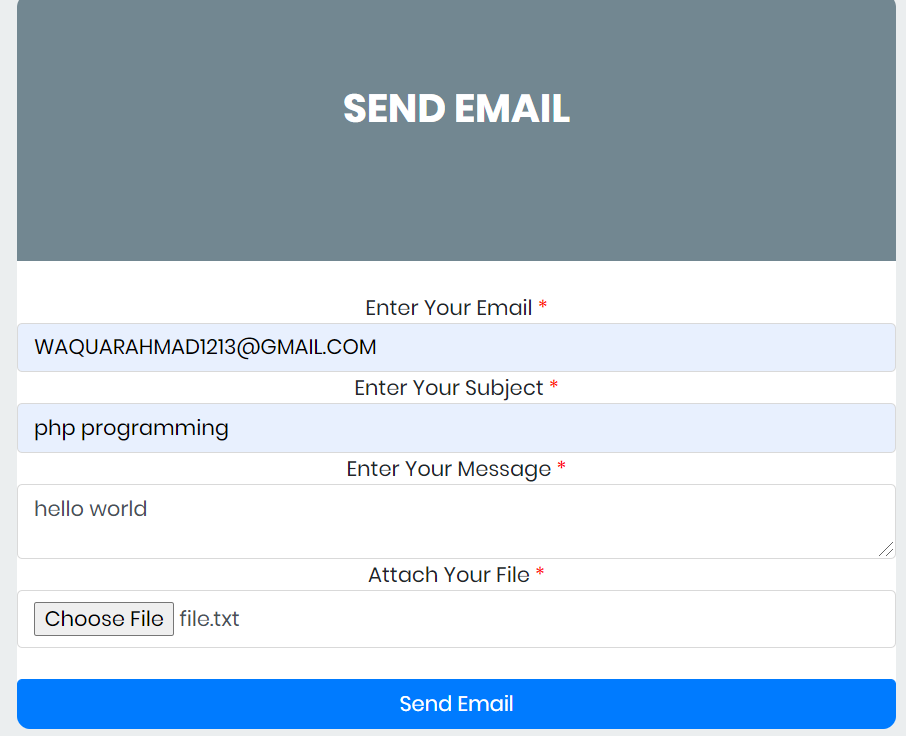
**;SMTP=smtp.gmail.com**

**;smtp\_port=587**

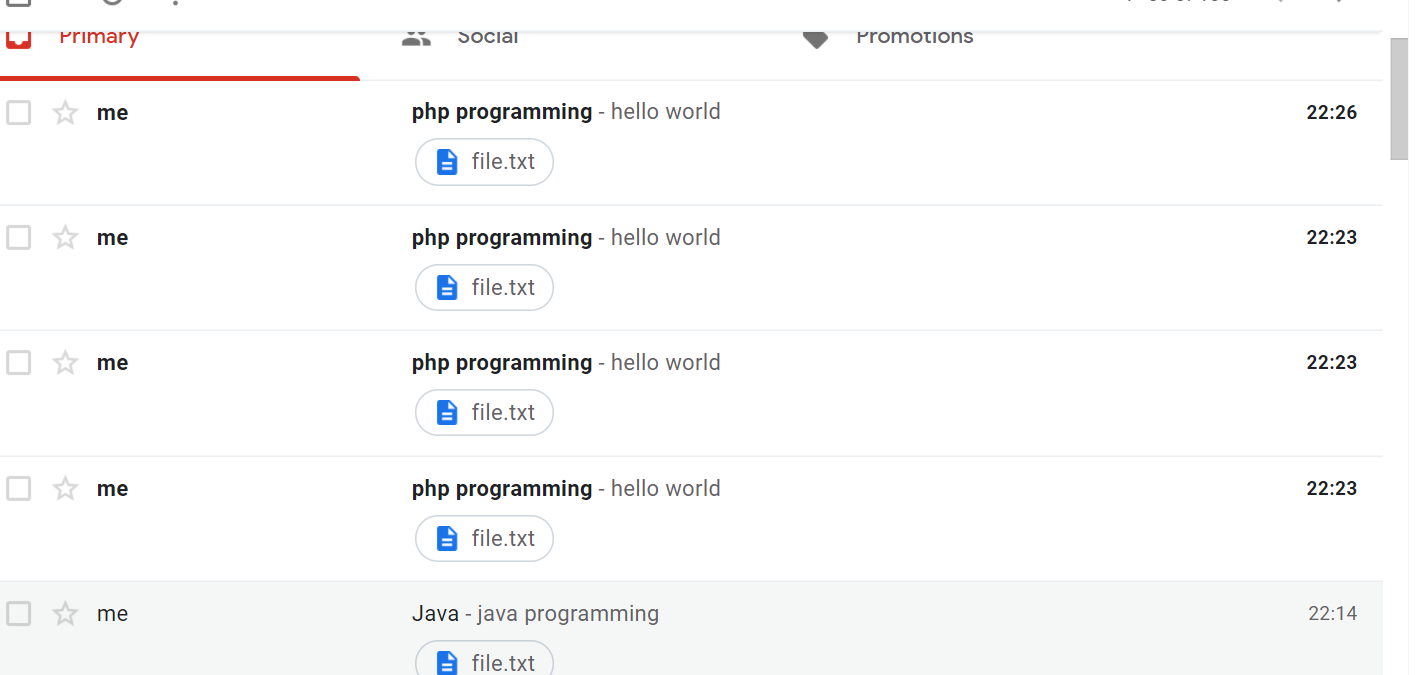
**;sendmail\_from** [**=youremail@gmail.com**](mailto:=youremail@gmail.com)

**;sendmail\_path ="\"C:\xampp\sendmail\sendmail.exe\" -t"**

**input/**

****

**output/**

****