LAMPIRAN

Lampiran – 1 Listing Program

- **Listing Program**

```
<?php

 if (!empty($_GET['q'])) {
  switch ($_GET['q']) {
    case 'info':
     phpinfo();
     exit;
    break;
  }
 }
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Laragon</title>

    <link
href="https://fonts.googleapis.com/css?family=Karla:400
" rel="stylesheet" type="text/css">

    <style>
      html, body {
        height: 100%;
      }

      body {
        margin: 0;
        padding: 0;
        width: 100%;
        display: table;
        font-weight: 100;
        font-family: 'Karla';
      }

      .container {
        text-align: center;
        display: table-cell;
        vertical-align: middle;
      }

      .content {
        text-align: center;
        display: inline-block;
      }

      .title {
        font-size: 96px;
      }

      .opt {
        margin-top: 30px;
      }

      .opt a {
        text-decoration: none;
        font-size: 150%;
      }

      a:hover {
        color: red;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="content">
        <div class="title"
title="Laragon">Laragon</div>

          <div class="info"><br />
            <?php
print($_SERVER['SERVER_SOFTWARE']); ?><br />
            PHP version: <?php print phpversion(); ?>
<span><a title="phpinfo()"
href="/?q=info">info</a></span><br />
            Document Root: <?php print
($_SERVER['DOCUMENT_ROOT']); ?><br />

          </div>
          <div class="opt">
            <div><a title="Getting Started"
href="https://laragon.org/docs">Getting
Started</a></div>
          </div>
        </div>

    </div>
  </body>
</html>
<?php
/**
 * PHP memcached client class
 *
 * For build develop environment in windows using
memcached.
 *
 * @package     memcached-client
 * @copyright   Copyright 2013-2014, Fwolf
```

```
 * @license     http://opensource.org/licenses/mit-license
MIT
 * @version     1.2.0
 */
class Memcached
{
    // Predefined Constants
    // See:
http://php.net/manual/en/memcached.constants.php

    // Defined in php_memcached.c
    const OPT_COMPRESSION = -1001;
    const OPT_SERIALIZER = -1003;

    // enum memcached_serializer in php_memcached
    const SERIALIZER_PHP = 1;
    const SERIALIZER_IGBINARY = 2;
    const SERIALIZER_JSON = 3;

    // Defined in php_memcached.c
    const OPT_PREFIX_KEY = -1002;

    // enum memcached_behavior_t in libmemcached
    const OPT_HASH = 2;
//MEMCACHED_BEHAVIOR_HASH

    // enum memcached_hash_t in libmemcached
    const HASH_DEFAULT = 0;
    const HASH_MD5 = 1;
    const HASH_CRC = 2;
    const HASH_FNV1_64 = 3;
    const HASH_FNV1A_64 = 4;
    const HASH_FNV1_32 = 5;
    const HASH_FNV1A_32 = 6;
    const HASH_HSIEH = 7;
    const HASH_MURMUR = 8;

    // enum memcached_behavior_t in libmemcached
    const OPT_DISTRIBUTION = 9;    //
MEMCACHED_BEHAVIOR_DISTRIBUTION

    // enum memcached_server_distribution_t in
libmemcached
    const DISTRIBUTION_MODULA = 0;
    const DISTRIBUTION_CONSISTENT = 1;

    // enum memcached_behavior_t in libmemcached
    const OPT_LIBKETAMA_COMPATIBLE = 16;   //
MEMCACHED_BEHAVIOR_KETAMA_WEIGHTED
    const OPT_BUFFER_WRITES = 10;          //
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
    const OPT_BINARY_PROTOCOL = 18;        //
MEMCACHED_BEHAVIOR_BINARY_PROTOCOL
    const OPT_NO_BLOCK = 0;                //
MEMCACHED_BEHAVIOR_NO_BLOCK
    const OPT_TCP_NODELAY = 1;             //
MEMCACHED_BEHAVIOR_TCP_NODELAY
    const OPT_SOCKET_SEND_SIZE = 4;        //
MEMCACHED_BEHAVIOR_SOCKET_SEND_SIZE
    const OPT_SOCKET_RECV_SIZE = 5;        //
MEMCACHED_BEHAVIOR_SOCKET_RECV_SIZE
    const OPT_CONNECT_TIMEOUT = 14;        //
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT
    const OPT_RETRY_TIMEOUT = 15;          //
MEMCACHED_BEHAVIOR_RETRY_TIMEOUT
    const OPT_SEND_TIMEOUT = 19;           //
MEMCACHED_BEHAVIOR_SND_TIMEOUT
    const OPT_RECV_TIMEOUT = 20;           //
MEMCACHED_BEHAVIOR_RCV_TIMEOUT
    const OPT_POLL_TIMEOUT = 8;            //
MEMCACHED_BEHAVIOR_POLL_TIMEOUT
    const OPT_CACHE_LOOKUPS = 6;           //
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS
    const OPT_SERVER_FAILURE_LIMIT = 21;   //
MEMCACHED_BEHAVIOR_SERVER_FAILURE_LIMIT

    // In php_memcached config, define
HAVE_MEMCACHED_IGBINARY default 1,
    // then use ifdef define HAVE_IGBINARY to 1.
    const HAVE_IGBINARY = 1;
    // In php_memcached config, define
HAVE_JSON_API default 1,
    // then use ifdef define HAVE_JSON to 1.
    const HAVE_JSON = 1;

    // Defined in php_memcached.c, (1<<0)
    const GET_PRESERVE_ORDER = 1;

    // enum memcached_return_t in libmemcached
    const RES_SUCCESS = 0;                 //
MEMCACHED_SUCCESS
    const RES_FAILURE = 1;                 //
MEMCACHED_FAILURE
    const RES_HOST_LOOKUP_FAILURE = 2;     //
MEMCACHED_HOST_LOOKUP_FAILURE
    const RES_UNKNOWN_READ_FAILURE = 7;    //
MEMCACHED_UNKNOWN_READ_FAILURE
    const RES_PROTOCOL_ERROR = 8;          //
MEMCACHED_PROTOCOL_ERROR
    const RES_CLIENT_ERROR = 9;            //
MEMCACHED_CLIENT_ERROR
    const RES_SERVER_ERROR = 10;           //
MEMCACHED_SERVER_ERROR
    const RES_WRITE_FAILURE = 5;           //
MEMCACHED_WRITE_FAILURE
    const RES_DATA_EXISTS = 12;            //
MEMCACHED_DATA_EXISTS
    const RES_NOTSTORED = 14;              //
MEMCACHED_NOTSTORED
    const RES_NOTFOUND = 16;               //
MEMCACHED_NOTFOUND
    const RES_PARTIAL_READ = 18;           //
MEMCACHED_PARTIAL_READ
    const RES_SOME_ERRORS = 19;            //
MEMCACHED_SOME_ERRORS
    const RES_NO_SERVERS = 20;             //
MEMCACHED_NO_SERVERS
    const RES_END = 21;                    //
MEMCACHED_END
    const RES_ERRNO = 26;                  //
MEMCACHED_ERRNO
    const RES_BUFFERED = 32;               //
MEMCACHED_BUFFERED
```

```
    const RES_TIMEOUT = 31;                 //
MEMCACHED_TIMEOUT
    const RES_BAD_KEY_PROVIDED = 33;        //
MEMCACHED_BAD_KEY_PROVIDED
    const
RES_CONNECTION_SOCKET_CREATE_FAILURE =
11;    //
MEMCACHED_CONNECTION_SOCKET_CREATE_FA
ILURE

    // Defined in php_memcached.c
    const RES_PAYLOAD_FAILURE = -1001;


    /**
     * Dummy option array
     *
     * @var array
     */
    protected $option = array(
        Memcached::OPT_COMPRESSION  => true,
        Memcached::OPT_SERIALIZER   =>
Memcached::SERIALIZER_PHP,
        Memcached::OPT_PREFIX_KEY   => '',
        Memcached::OPT_HASH         =>
Memcached::HASH_DEFAULT,
        Memcached::OPT_DISTRIBUTION =>
Memcached::DISTRIBUTION_MODULA,
        Memcached::OPT_LIBKETAMA_COMPATIBLE =>
false,
        Memcached::OPT_BUFFER_WRITES    => false,
        Memcached::OPT_BINARY_PROTOCOL  =>
false,
        Memcached::OPT_NO_BLOCK     => false,
        Memcached::OPT_TCP_NODELAY  => false,

        // This two is a value by guess
        Memcached::OPT_SOCKET_SEND_SIZE =>
32767,
        Memcached::OPT_SOCKET_RECV_SIZE =>
65535,

        Memcached::OPT_CONNECT_TIMEOUT  =>
1000,
        Memcached::OPT_RETRY_TIMEOUT    => 0,
        Memcached::OPT_SEND_TIMEOUT     => 0,
        Memcached::OPT_RECV_TIMEOUT     => 0,
        Memcached::OPT_POLL_TIMEOUT     => 1000,
        Memcached::OPT_CACHE_LOOKUPS    => false,
        Memcached::OPT_SERVER_FAILURE_LIMIT =>
0,
    );


    /**
     * Last result code
     *
     * @var int
     */
    protected $resultCode = 0;


    /**
     * Last result message
     *
     * @var string
     */
    protected $resultMessage = '';


    /**
     * Server list array/pool
     *
     * I added array index.
     *
     * array (
     *  host:port:weight => array(
     *      host,
     *      port,
     *      weight,
     *  )
     * )
     *
     * @var array
     */
    protected $server = array();


    /**
     * Socket connect handle
     *
     * Point to last successful connect, ignore others
     * @var resource
     */
    protected $socket = null;

    public function getVersion() {
        return ['localhost:11211' =>  '1.4.5'];
    }


    //may check:
https://raw.githubusercontent.com/GoogleCloudPlatform
/python-compat-runtime/master/appengine-
compat/exported_appengine_sdk/php/sdk/google/appen
gine/runtime/Memcached.php
                public function setMulti() {
                        die('TODO');
                }
    public function getMulti() {
        //TODO
        die('TODO');
    }

    /**
     * Add a serer to the server pool
     *
     * @param   string  $host
     * @param   int     $port
     * @param   int     $weight
     * @return  boolean
     */
    public function addServer($host, $port = 11211,
$weight = 0)
```

```
    {
        $key = $this->getServerKey($host, $port, $weight);
        if (isset($this->server[$key])) {
            // Dup
            $this->resultCode =
Memcached::RES_FAILURE;
            $this->resultMessage = 'Server duplicate.';
            return false;

        } else {
            $this->server[$key] = array(
                'host'  => $host,
                'port'  => $port,
                'weight'    => $weight,
            );

            $this->connect();
            return true;
        }
    }


    /**
     * Add multiple servers to the server pool
     *
     * @param   array   $servers
     * @return  boolean
     */
    public function addServers($servers)
    {
        foreach ((array)$servers as $svr) {
            $host = array_shift($svr);

            $port = array_shift($svr);
            if (is_null($port)) {
                $port = 11211;
            }

            $weight = array_shift($svr);
            if (is_null($weight)) {
                $weight = 0;
            }

            $this->addServer($host, $port, $weight);
        }

        return true;
    }


    /**
     * Connect to memcached server
     *
     * @return  boolean
     */
    protected function connect()
    {
        $rs = false;

        foreach ((array)$this->server as $svr) {
            $error = 0;
            $errstr = '';

            $rs = @fsockopen($svr['host'], $svr['port'],
$error, $errstr);

            if ($rs) {
                $this->socket = $rs;

            } else {
                $key = $this->getServerKey(
                    $svr['host'],
                    $svr['port'],
                    $svr['weight']
                );
                $s = "Connect to $key error:" . PHP_EOL .
                    "    [$error] $errstr";
                error_log($s);
            }
        }

        if (is_null($this->socket)) {
            $this->resultCode =
Memcached::RES_FAILURE;
            $this->resultMessage = 'No server avaliable.';
            return false;

        } else {
            $this->resultCode =
Memcached::RES_SUCCESS;
            $this->resultMessage = '';
            return true;
        }
    }


    /**
     * Delete an item
     *
     * @param   string  $key
     * @param   int     $time       Ignored
     * @return  boolean
     */
    public function delete($key, $time = 0)
    {
        $keyString = $this->getKey($key);
        $this->writeSocket("delete $keyString");

        $s = $this->readSocket();
        if ('DELETED' == $s) {
            $this->resultCode =
Memcached::RES_SUCCESS;
            $this->resultMessage = '';
            return true;

        } else {
            $this->resultCode =
Memcached::RES_NOTFOUND;
            $this->resultMessage = 'Delete fail, key not
exists.';
            return false;
        }
    }
```

```
    /**
     * Retrieve an item
     *
     * @param   string  $key
     * @param   callable    $cache_cb       Ignored
     * @param   float   $cas_token          Ignored
     * @return  mixed
     */
    public function get($key, $cache_cb = null,
$cas_token = null)
    {
        $keyString = $this->getKey($key);
        $this->writeSocket("get $keyString");

        $s = $this->readSocket();

        if (is_null($s) || 'VALUE' != substr($s, 0, 5)) {
            $this->resultCode =
Memcached::RES_FAILURE;
            $this->resultMessage = 'Get fail.';
            return false;

        } else {
            $s_result = '';
            $s = $this->readSocket();
            while ('END' != $s) {
                $s_result .= $s;
                $s = $this->readSocket();
            }
            $this->resultCode =
Memcached::RES_SUCCESS;
            $this->resultMessage = '';

            return unserialize($s_result);
        }
    }


    /**
     * Get item key
     *
     * @param   string  $key
     * @return  string
     */
    public function getKey($key)
    {
        return addslashes($this-
>option[Memcached::OPT_PREFIX_KEY]) . $key;
    }


    /**
     * Get a memcached option value
     *
     * @param   int     $option
     * @return  mixed
     */
    public function getOption($option)
    {
        if (isset($this->option[$option])) {
            $this->resultCode =
Memcached::RES_SUCCESS;
            $this->resultMessage = '';
            return $this->option[$option];

        } else {
            $this->resultCode =
Memcached::RES_FAILURE;
            $this->resultMessage = 'Option not seted.';
            return false;
        }
    }


    /**
     * Return the result code of the last operation
     *
     * @return  int
     */
    public function getResultCode()
    {
        return $this->resultCode;
    }


    /**
     * Return the message describing the result of the last
opteration
     *
     * @return  string
     */
    public function getResultMessage()
    {
        return $this->resultMessage;
    }


    /**
     * Get key of server array
     *
     * @param   string  $host
     * @param   int     $port
     * @param   int     $weight
     * @return  string
     */
    protected function getServerKey($host, $port =
11211, $weight = 0)
    {
        return "$host:$port:$weight";
    }


    /**
     * Get list array of servers
     *
     * @see     $server
     * @return  array
     */
    public function getServerList()
    {
        return $this->server;
    }
```

```
    /**
     * Read from socket
     *
     * @return  string|null
     */
    protected function readSocket()
    {
        if (is_null($this->socket)) {
            return null;
        }

        return trim(fgets($this->socket));
    }



    /**
     * Store an item
     *
     * @param   string  $key
     * @param   mixed   $val
     * @param   int     $expt
     * @return  boolean
     */
    public function set($key, $val, $expt = 0)
    {
        $valueString = serialize($val);
        $keyString = $this->getKey($key);

        $this->writeSocket(
            "set $keyString 0 $expt " . strlen($valueString)
        );
        $s = $this->writeSocket($valueString, true);

        if ('STORED' == $s) {
            $this->resultCode =
Memcached::RES_SUCCESS;
            $this->resultMessage = '';
            return true;

        } else {
            $this->resultCode =
Memcached::RES_FAILURE;
            $this->resultMessage = 'Set fail.';
            return false;
        }
    }


    /**
     * Set a memcached option
     *
     * @param   int     $option
     * @param   mixed   $value
     * @return  boolean
     */
    public function setOption($option, $value)
    {
        $this->option[$option] = $value;
        return true;
    }


    /**
     * Set memcached options
     *
     * @param   array   $options
     * @return  bollean
     */
    public function setOptions($options)
    {
        $this->option = array_merge($this->option,
$options);
        return true;
    }

    /**
     * Increment numeric item's value
     *
     * @param string $key           The key of the item to
increment.
     * @param int    $offset        The amount by which to
increment the item's value.
     * @param int    $initial_value The value to set the
item to if it doesn't currently exist.
     * @param int    $expiry        The expiry time to set on
the item.
     *
     * @return mixed                Returns new item's value
on success or FALSE on failure.
     */
    public function increment($key, $offset = 1,
$initial_value = 0, $expiry = 0)
    {
        if (($prevVal = $this->get($key))) {
            if (!is_numeric($prevVal)) {
                return false;
            }

            $newVal = $prevVal + $offset;
        } else {
            $newVal = $initial_value;
        }

        $this->set($key, $newVal, $expiry);

        return $newVal;
    }

    /**
     * Write data to socket
     *
     * @param   string  $cmd
     * @param   boolean $result     Need result/response
     * @return  mixed
     */
    protected function writeSocket($cmd, $result = false)
    {
        if (is_null($this->socket)) {
            return false;
        }

        fwrite($this->socket, $cmd . "\r\n");
```

```
        if (true == $result) {
            return $this->readSocket();
        }

        return true;
    }
}
<?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 = '5.2.14';

    /**
     * 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.
     * @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 LOGIN (default), PLAIN, NTLM,
CRAM-MD5
     * @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.
     * @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 = '';

    /**
     * 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
     *   string  $to            email address of the recipient
```

```
     *   string  $cc           cc email addresses
     *   string  $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 = '';

    /**
     * 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 = false)
    {
        $this->exceptions = (boolean)$exceptions;
    }

    /**
     * Destructor.
     */
    public function __destruct()
    {
        //Close any open SMTP connection nicely
        if ($this->Mailer == 'smtp') {
            $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($subject));
        }
        if (ini_get('safe_mode') || !($this-
>UseSendmailOptions)) {
            $result = @mail($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|\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 $patternselect A selector for the
validation pattern to use :
   * * `auto` Pick best pattern automatically;
   * * `pcre8` Use the squiloople.com pattern, requires
PCRE > 8.0, PHP >= 5.3.2, 5.2.14;
   * * `pcre` Use old PCRE implementation;
   * * `php` Use PHP built-in
FILTER_VALIDATE_EMAIL;
   * * `html5` Use the pattern given by the HTML5 spec
for 'email' type form input elements.
   * * `noregex` Don't use a regex: super fast, really
dumb.
   * @return boolean
   * @static
   * @access public
   */
  public static function validateAddress($address,
$patternselect = 'auto')
  {
    //Reject line breaks in addresses; it's valid
RFC5322, but not RFC5321
    if (strpos($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($this->DKIM_domain)
                && !empty($this->DKIM_private)
                && !empty($this->DKIM_selector)
                && file_exists($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, $body)
    {
        if ($this->Sender != '') {
            if ($this->Mailer == 'qmail') {
                $sendmail = sprintf('%s -f%s',
escapeshellcmd($this->Sendmail),
escapeshellarg($this->Sender));
            } else {
                $sendmail = sprintf('%s -oi -f%s -t',
escapeshellcmd($this->Sendmail),
escapeshellarg($this->Sender));
            }
        } else {
            if ($this->Mailer == 'qmail') {
                $sendmail = sprintf('%s',
escapeshellcmd($this->Sendmail));
            } else {
```

```
            $sendmail = sprintf('%s -oi -t',
escapeshellcmd($this->Sendmail));
          }
        }
        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;
      }

      /**
       * 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);

        if (empty($this->Sender)) {
          $params = ' ';
        } else {
          $params = sprintf('-f%s', $this->Sender);
        }
        if ($this->Sender != '' and !ini_get('safe_mode')) {
          $old_from = ini_get('sendmail_from');
          ini_set('sendmail_from', $this->Sender);
        }
        $result = false;
        if ($this->SingleTo && 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 ('' == $this->Sender) {
         $smtp_from = $this->From;
      } else {
         $smtp_from = $this->Sender;
      }
      if (!$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 = array())
   {
      if (is_null($this->smtp)) {
         $this->smtp = $this->getSMTPInstance();
      }

      // 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) {
         $hostinfo = array();
         if (!preg_match('/^((ssl|tls):\/\/)*([a-zA-Z0-9\.-
]*):?([0-9]*)$/', trim($hostentry), $hostinfo)) {
            // Not a valid host entry
            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 HELO 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 ($this->smtp !== null) {
      if ($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 = '')
  {
    // 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;
        }
        $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 (!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 = '';

    if ($this->MessageDate == '') {
        $this->MessageDate = self::rfcDate();
    }
    $result .= $this->headerLine('Date', $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)));
        }

        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;
    }

    /**
     * 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 = md5(uniqid(time()));
        $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';
            $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
        if ('base64' != $this->Encoding and
self::hasLineLongerThanMax($this->Body)) {
            $this->Encoding = 'quoted-printable';
            $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';
            $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
        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 ''
            $body .= $this->encodeString($this->Body,
$bodyEncoding);
            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 == '') {
      $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.
     * Returns false if the file could not be found or read.
     * @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 = '', $disposition =
'attachment')
    {
        try {
            if (!@is_file($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, $encoding =
'base64')
  {
    try {
      if (!is_readable($path)) {
        throw new phpmailerException($this-
>lang('file_open') . $path, self::STOP_CONTINUE);
      }
      $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.
     * @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 = '', $disposition
= 'inline')
    {
        if (!@is_file($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_g
et());
        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/Troublesh
ooting';
            }
            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 from an HTML string.
     * Automatically makes modifications for inline images
and backgrounds
     * and creates a plain-text version by converting the
HTML.
     * Overwrites any existing values in $this->Body and
$this->AltBody
     * @access public
     * @param string $message HTML message string
     * @param string $basedir baseline directory for path
     * @param boolean|callable $advanced Whether to
use the internal HTML to text converter
     *    or your own custom converter @see
PHPMailer::html2text()
     * @return string $message
     */
    public function msgHTML($message, $basedir = '',
$advanced = false)
    {
        preg_match_all('/(src|background)=["\'](.*)["\']/Ui',
$message, $images);
        if (array_key_exists(2, $images)) {
            foreach ($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 .
'"',
                            $message
                        );
                    }
                } elseif (substr($url, 0, 4) !== 'cid:' &&
!preg_match('#^[A-z]+://#', $url)) {
                    // Do not change urls for absolute images
(thanks to corvuscorax)
                    // Do not change urls that are already inline
images
                    $filename = basename($url);
                    $directory = dirname($url);
                    if ($directory == '.') {
                        $directory = '';
                    }
                    $cid = md5($url) . '@phpmailer.0'; //
RFC2392 S 2
                    if (strlen($basedir) > 1 && substr($basedir,
-1) != '/') {
                        $basedir .= '/';
                    }
                    if (strlen($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#10
7461
     * @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) {
```

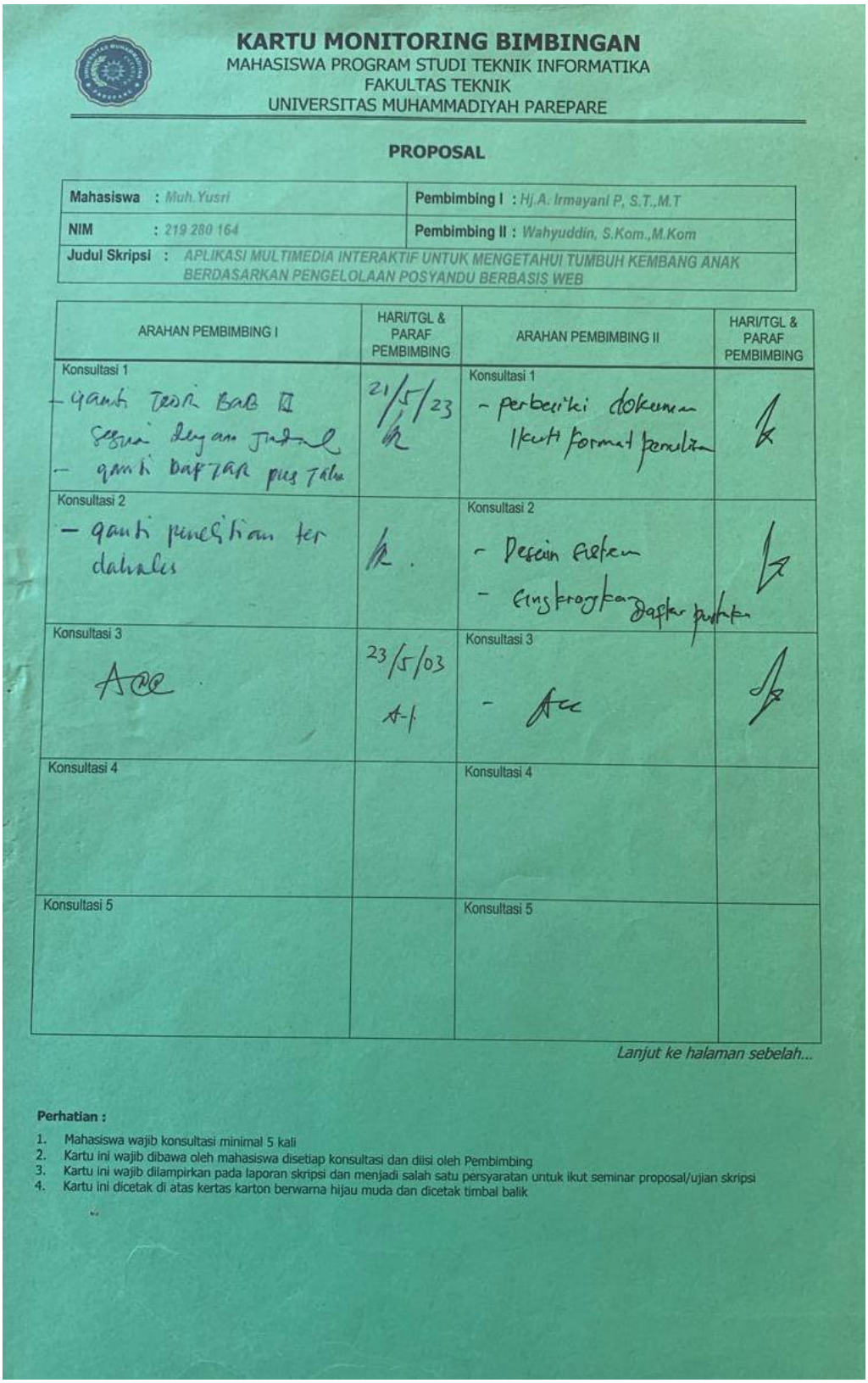
```
            case PATHINFO_DIRNAME:
            case 'dirname':
                return $ret['dirname'];
            case PATHINFO_BASENAME:
            case 'basename':
                return $ret['basename'];
            case PATHINFO_EXTENSION:
            case 'extension':
                return $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
```

```
     */
    public function DKIM_Sign($signHeader)
    {
        if (!defined('PKCS7_TEXT')) {
            if ($this->exceptions) {
                throw new phpmailerException($this->lang('extension_missing') . 'openssl');
            }
            return '';
        }
        $privKeyStr = file_get_contents($this->DKIM_private);
        if ($this->DKIM_passphrase != '') {
            $privKey = openssl_pkey_get_private($privKeyStr, $this->DKIM_passphrase);
        } else {
            $privKey = openssl_pkey_get_private($privKeyStr);
        }
        if (openssl_sign($signHeader, $signature, $privKey, 'sha256WithRSAEncryption')) {
//sha1WithRSAEncryption
            openssl_pkey_free($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+/', ' ', $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>' . $this->getMessage() .
"</strong><br />\n";
        return $errorMsg;
    }
}
```

Lampiran–2 Kartu Monitoring Proposal



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : Muh.Yusri	**Pembimbing I** : Hj.A. Irmayani P, S.T.,M.T
NIM : 219 280 164	**Pembimbing II** : Wahyuddin, S.Kom.,M.Kom
Judul Skripsi : APLIKASI MULTIMEDIA INTERAKTIF UNTUK MENGETAHUI TUMBUH KEMBANG ANAK BERDASARKAN PENGELOLAAN POSYANDU BERBASIS WEB	

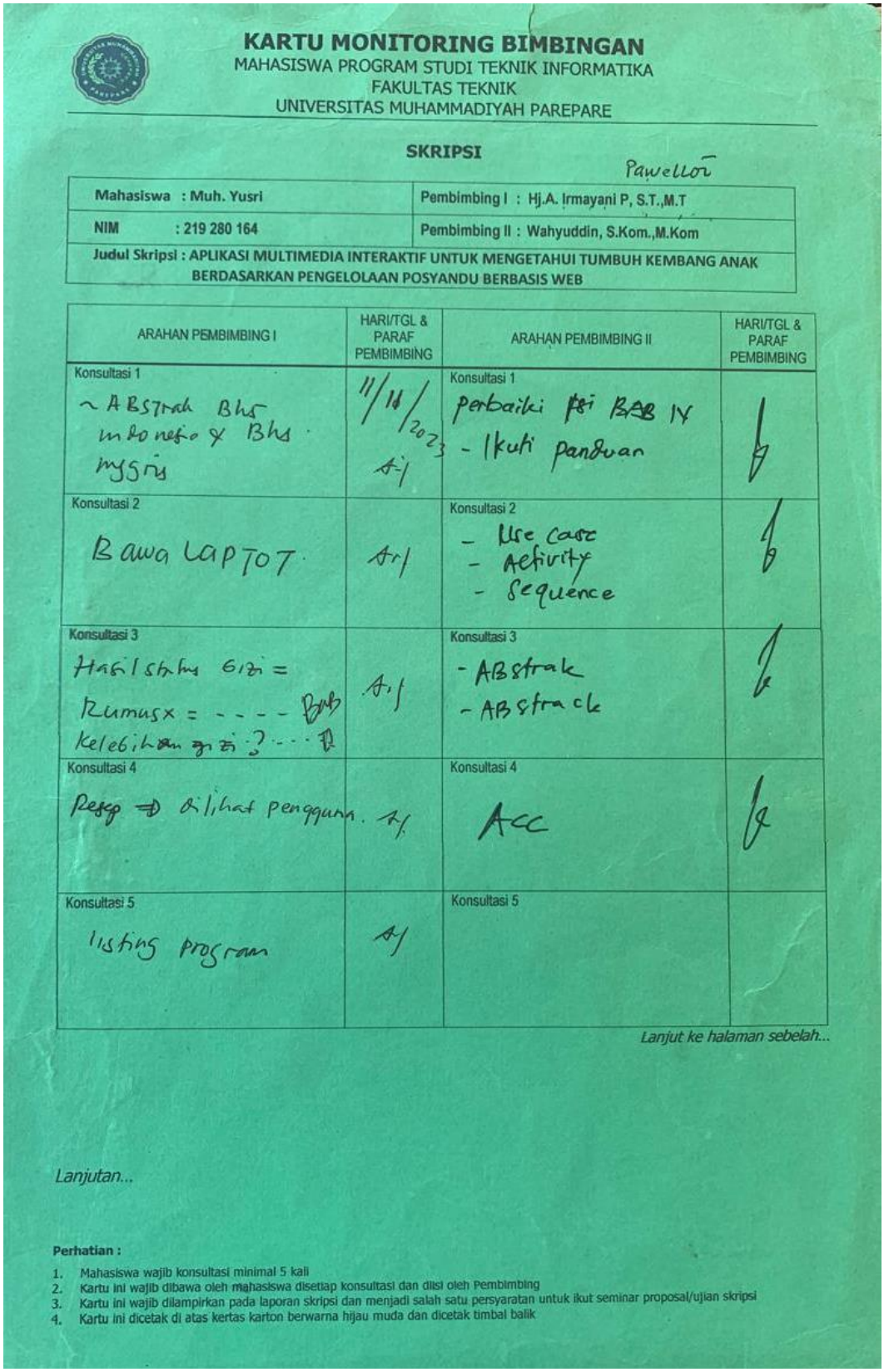
ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 - ganti Teori BAB II sesuai dengan Judul - ganti DAFTAR pus Tabel	21/5/23	Konsultasi 1 - perbaiki dokumen ikuti format penulisan	
Konsultasi 2 - ganti penelitian ter dahulu		Konsultasi 2 - Desain sistem - Lengkapkan daftar pustaka	
Konsultasi 3 Acc	23/5/03	Konsultasi 3 - Acc	
Konsultasi 4		Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

Lampiran – 3 Kartu Monitoring Skripsi



KARTU MONITORING BIMBINGAN
MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

Pembimbing I

Mahasiswa : Muh. Yusri	Pembimbing I : Hj.A. Irmayani P, S.T.,M.T
NIM : 219 280 164	Pembimbing II : Wahyuddin, S.Kom.,M.Kom
Judul Skripsi : APLIKASI MULTIMEDIA INTERAKTIF UNTUK MENGETAHUI TUMBUH KEMBANG ANAK BERDASARKAN PENGELOLAAN POSYANDU BERBASIS WEB	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 ~ ABStrak Bhs Indonesia & Bhs Inggris	11/10/2023	Konsultasi 1 perbaiki lagi BAB IV - Ikuti panduan	
Konsultasi 2 Bawa LAPTOT.		Konsultasi 2 - Use Case - Aefivity - Sequence	
Konsultasi 3 Hasil status Gizi = Rumus x = - - - - Bab Kelebihan gizi ? - - -		Konsultasi 3 - ABStrak - ABStrack	
Konsultasi 4 Resep ⇒ dilihat pengguna.		Konsultasi 4 ACC	
Konsultasi 5 listing program		Konsultasi 5	

Lanjut ke halaman sebelah...

Lanjutan...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 6 - Tamgah batz pengujian cisig 12 Bulan 14 Bulan	4/12 2023	Konsultasi 6	
Konsultasi 7 Acc.	24/12/2023 A.n.	Konsultasi 7	
Konsultasi 8 Acc u/ skripsi	19/3/2024. A.n.	Konsultasi 8	
Konsultasi 9		Konsultasi 9	
Konsultasi 10		Konsultasi 10	

Parepare,

Mengetahui
Ketua Program Studi

Marlina, S.Kom.,M.Kom.
NBM. 1162 680

Mahasiswa

Muh. Yusri
NIM. 219 280 164

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik