

## 3.7: Email

*maDisclaimer: Unless you are working on a server that already has email capability, this chapter may not work for you. WAMP 2.0 (or just Apache, MySQL, and PHP by themselves) do not contain the ability to act as an email server in and of themselves. If you have an email account that uses an exchange email server or another hosted solution, you can research how to configure that account into your server to use your address to send email. Since this process will differ depending on the service you use, comprehensive directions on how to do so are not possible here.*

In short, you will need account credentials such as a username and password, some connection settings that can be found in your account settings that may be stored in your email software, web settings, or phone settings, and you will need to edit your php.ini file's sendmail settings. Once you have made the necessary changes, do not forget to stop and start your server for them to take effect.

### Text Based

Regardless of what service you use to facilitate sending email, you will always use the same function in PHP to trigger it. The mail() function allows us to specify the “Who, What, and Where” settings. Through this function we can support multimedia (HTML based) emails and attachments as well.

The minimum information necessary to send an email is the recipient and message, assuming you have placed an email address in the “From” portion of the php.ini settings file:

1. mail("ourRecipient@nowhere.com", "That is a nice inbox you got there!");

The full list of options at our disposal includes the following:

1. mail([to], [subject], [message], [headers]);

The headers section allows us to pass further information like a different “from” address than our default, a reply email, CCs, BCCs, and information on other features like HTML content or attachments. Since this last section and our actual message can be quite long, it is helpful to declare all of these elements as variables first to keep our actual mail function call readable:

1. \$to = "ourRecipient@nowhere.com";
2. \$subject = "You Win a million dollars!";
3. \$message = "Not really. We just wanted you to read this.";
4. \$headers .= "From: Us <us@somewhereOutThere.com>" . "\r\n";
5. \$headers .= "Cc: someoneelse@nowhere.com" . "\r\n";
6. \$headers .= "Bcc: hiddenPerson@definatlyNotHere.com" . "\r\n";
7. mail(\$to, \$subject, \$message, \$headers);

You will notice that our headers string actually contains the labels of the email, and that line breaks are included at the end of each piece. This helps translate the string into separate pieces of information when we submit the variable to the mail function in order to create our actual email.

### HTML

To make our messages look better and incorporate things like color and images, we can add HTML to our message. We can do this by inserting two more lines to our header that specify this:

1. \$headers .= "MIME-Version: 1.0\r\n";
2. \$headers .= "Content-Type: text/html; charset=ISO-8859-1\r\n";

Declaring our MIME Type version and content type of text/html allows us to include HTML in our message. At this point, we can edit our message string to include HTML tags. Since it is still a PHP string, we can include variables and concatenate results from functions just as we can in other strings, allowing us to create messages that include content specific to the user or recipient:

1. \$message = "<table width='20%'><tr><td>First:</td><td>Jose</td></tr><tr><td>Last:</td><td>Jalapeno</td></tr></table>";

Now we have sent our user some information formatted into a table. While we can include a lot of HTML in a message, keep in mind your users will be viewing them on a number of different devices and through different programs. The more complex the

content, the more likely your user will not see it as you intend.

In fact, best practices for HTML email are to include the content formatted for email clients that only support text, or as a fall back when something else goes awry. To do this, we need to add a few more lines of code to specify which parts of our message belong to the HTML version, and which parts belong to the text version. Part of this involves creating an indicator that specifies where sections start and end. To do this, we need a unique string—unique enough that it would never be an intended part of our message. An easy way to do this is to use the md5 and time functions to generate our string. **This does not mean we have encrypted our email.** The use of md5 simply helps us generate a long, random string:

1. `$divider = md5((date('r', time())));`

We also need to edit our header line to announce that our message is multipart and not just HTML:

1. `$headers .= "MIME-Version: 1.0\r\n";`
2. `$headers .= "Content-Type: multipart/alternative; boundary=\"PHP-alt-\".$random_hash; charset=ISO-8859-1\r\n";`

Now we will add our \$divider where our message starts, where we fall back from HTML to text, and then at the end of our text section. Whenever we start a new section, we need to specify which MIME format we are using. This allows us to mix and match multiple things, even text, HTML, and attachments, all in one message. Since things are getting a bit more complex, we will introduce a new concept, output buffering, to keep things cleaner. Output buffering allows us to “stop” outputting anything to the screen or browser. By using the buffer, we can create larger sections of text to use elsewhere. When we are done, we will store the buffer contents into our message variable.

1. `<?php ob_start(); ?>`
2. `—PHP-alt-<?php echo $divider; ?>`
3. `Content-Type: text/html; charset="iso-8859-1"`
4. `Content-Transfer-Encoding: 7bit`
5. `<table width='20%'><tr><td>First:</td><td>Jose</td></tr>`
6. `<tr><td>Last:</td><td>Jalapeno</td></tr></table>`
7. `—PHP-alt-<?php echo $divider; ?>`
8. `Content-Type: text/plain; charset="iso-8859-1"`
9. `Content-Transfer-Encoding: 7bit`
10. `First: Jose`
11. `Last: Jalapeno`
12. `—PHP-alt-<?php echo $divider; —`
13. `$message = ob_get_clean(); ?>`

You will note that we closed our PHP tags after starting our output buffer. This means everything that follows would be treated as HTML to be rendered, so we still need to open and close PHP tags when we want to insert dynamic content into our message. Once we are done with our message(s), we use the `ob_get_clean()` function to dump the buffer’s content into our message variable. This action also closes, or clears, the buffer we were using. Now we have an email that supports an HTML and plain text version of our message. To add an attachment, we would add one more MIME type section that names the file format we want to attach, then include in our buffer content the file’s content. Since we cannot drop the actual file into our code, we need to encode it. We can take care of all of this by adding a couple extra lines at the start of our section for our attachment.

1. `—PHP-mixed-<?php echo $division; ?>`
2. `Content-Type: application/zip; name="ourFile.pdf"`
3. `Content-Transfer-Encoding: base64`
4. `Content-Disposition: attachment`
5. `<?php echo $attachment; ?>`
6. `—PHP-mixed-<?php echo $division; ?>—`

When you send attachments, you will want to keep in mind the size of the file(s) that you are attaching. Just because your server is able to process and send the file does not mean your recipient’s server will be able to accept it. Emails with attachments should be accompanied by a body as well. If there is no text in the body of the email, the recipient’s email client may elect to treat the attachment as the body of the message.

---

3.7: Email is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- 3.7: Email by Michael Mendez has no license indicated.