Essential PHP Tools: Modules, Extensions, and Accelerators

DAVID SKLAR

Apress™

Essential PHP Tools: Modules, Extensions, and Accelerators
Copyright ©2004 by David Sklar

ISBN (pbk): 1-59059-280-8

Printed and bound in the United States of America 12345678910

Technical Reviewer: Adam Trachtenberg

Editorial Board: Steve Anglin, Dan Appleman, Gary Cornell, James Cox, Tony Davis, John Franklin, Chris Mills, Steve Rycroft, Dominic Shakeshaft, Julian Skinner, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Kylie Johnston

Copy Editor: Kim Wimpsett

Production Manager: Kari Brooks

Production Editor: Noemi Hollander

Proofreader: Thistle Hill Publishing Services, LLC

Compositor: Kinetic Publishing Services, LLC

Indexer: Valerie Perry

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, email `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 9

Sending Mail

WEB SITES DON'T JUST display Web pages. Often, they also send e-mail messages. For example, sending a confirmation message to a user after he creates an account, changes a password, or places an order is a common practice. Many Web sites also send periodic e-mail newsletters containing new headlines or information about recent site updates. The PEAR Mail and Mail_mime modules provide a flexible, robust way to send e-mail messages from your PHP programs.

This chapter discusses PEAR Mail 1.1.2 and Mail_mime 1.2.1. With the PEAR Mail module, you have more control over how your e-mail messages are sent than if you just use PHP's built-in `mail()` function. The Mail_mime module supplies a way to send messages that are more complicated than plain-text ones—messages with attachments, HTML, and embedded images.

Sending Plain-Text Mail Messages with PEAR Mail

The PEAR Mail module sends e-mail messages and lets you switch easily between using a local or remote method to send your message and piggybacking on PHP's built-in `mail()` function. PEAR Mail also automatically takes care of platform dependencies such as line endings.

To send a message, create a new `Mail` object using the `Mail::factory()` method and then call the `send()` method on that object, passing it the recipient's address, message body, and headers. Listing 9-1 demonstrates sending a message with the Mail module.

Listing 9-1. Sending an E-mail Message

```
require 'Mail.php';
$mailer =& Mail::factory('mail');
$to = 'destination@example.com';
$headers = array('Subject' => 'Hungry?',
                  'Cc: other-destination@example.com');
$body=<<<_MSG_
Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster
```

```
_MSG_;
$res = $mailer->send($to, $headers, $body);
// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

The first argument to `Mail::factory()` is which driver to use. The driver controls how the `Mail` object sends the message. Your driver choices are `mail`, `smtp`, and `sendmail`. The `mail` driver uses PHP's built-in `mail()` function, and the `smtp` driver uses an SMTP server. If you have Sendmail or a similar mailer program installed, you can use the `sendmail` driver as well.

Because the `mail` driver sends messages with PHP's built-in `mail()` function, it relies on the mail-related PHP configuration settings: `sendmail_path` and, on Windows, `sendmail_from`. The `sendmail_path` configuration setting should be set to the path of an external mailer program and any arguments you want to pass to the program. If you're using Sendmail on Unix, its path is probably `/usr/sbin/sendmail` or `/usr/lib/sendmail`. When you build and install PHP, the `configure` script looks for Sendmail and sets the configuration setting appropriately.

If you are using a mailer program other than Sendmail, set `sendmail_path` to the location of a Sendmail-compatible mail program that may have been supplied with your mailer software. For example, use the `sendmail` wrapper program that comes with qmail or the regular `exim` binary for Exim.

PHP 5 supports another mail-related configuration setting: `mail_force_extra_parameters`. The value of this setting is tacked onto the end of the command line when the `mail()` function runs Sendmail. You can use `mail_force_extra_parameters` to pass arguments to Sendmail.

Windows doesn't come with a built-in mailer program, but you can instruct the `mail()` function to send mail via an SMTP server with the `SMTP` and `smtp_port` configuration settings. Set `SMTP` to the name of the SMTP server you want to use, and, if the SMTP server uses a port other than 25 (the default SMTP port), set `smtp_port` to that alternative port number. With `SMTP` set and `sendmail_path` not set, the `mail()` function connects to the server defined by `SMTP` and relays the message through that server.

On any platform, you can bypass the `mail()` function and the intricacies of its configuration directives with the `smtp` driver. However, this driver requires that the Net_SMTP and Net_Socket PEAR modules be installed. Listing 9-2 shows how to send a message with the `smtp` driver.

Listing 9-2. Sending a Message with the `smtp` *Driver*

```
require 'Mail.php';
$params = array('host' => 'smtp-server.example.com',
                'port' => '25');
```

```
$mailer =& Mail::factory('smtp',$params);
$to = 'destination@example.com';
$headers = array('From' => 'icecream@example.com',
                 'Subject' => 'Hungry?',
                 'Cc' => 'other-destination@example.com');
$body=<<<_MSG_
Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster
_MSG_;

$res = $mailer->send($to, $headers, $body);

// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

Two differences exist between the code that uses the `smtp` driver and the code that uses the `mail` driver. First, when the `smtp` driver is specified, the `Mail::factory()` method can take a second argument, an array of parameters. Second, the `smtp` driver requires that a `From` address is specified in the message headers. This can be with a `From` header, as shown in Listing 9-2, or with a `Return-Path` header. Table 9-1 shows the complete list of the parameters that the `smtp` driver accepts.

Table 9-1. Options for the `smtp` *Driver*

Option	**Description**	**Default**
`host`	Hostname of an SMTP server to connect to	`localhost`
`port`	Port on the SMTP server to connect to	`25`
`auth`	Whether to use SMTP authentication	`false`
`username`	Username for authentication	
`password`	Password for authentication	
`localhost`	How the connecting host identifies itself to the remote SMTP server	`localhost`

The Net_SMTP module, used by the Mail module, can handle four SMTP authentication methods: DIGEST-MD5, CRAM-MD5, LOGIN, and PLAIN.[1] If you choose to use SMTP authentication, Net_SMTP asks the SMTP server what kind of authentication it knows about and then chooses a method that both it and the server support, preferring DIGEST-MD5 to CRAM-MD5, CRAM-MD5 to LOGIN, and LOGIN to PLAIN.

In addition to the `mail` and `smtp` drivers, Mail also has a `sendmail` driver. This driver runs a local Sendmail program (or a similarly behaving program from another mailer) to send a message. If you want to use a local mailer program but can't alter the `sendmail_path` configuration setting because of how your PHP installation is set up, this driver is a good choice. Listing 9-3 demonstrates how to send a message with the `sendmail` driver.

Listing 9-3. Sending a Message with the `sendmail` *Driver*

```
require 'Mail.php';
$params = array('sendmail_path' =>'/usr/lib/sendmail',
                 'sendmail_args' => '-i');
$mailer =& Mail::factory('sendmail',$params);
$to = 'destination@example.com';
$headers = array('From' => 'icecream@example.com',
                  'Subject' => 'Hungry?',
                  'Cc' => 'other-destination@example.com');
$body=<<<_MSG_
Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster
_MSG_;

$res = $mailer->send($to, $headers, $body);

// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

1. To use the DIGEST-MD5 or CRAM-MD5 methods, you must have the PEAR Auth_SASL module installed.

The sendmail driver requires you to set a From address in the message headers. When it runs the Sendmail program, specified in the sendmail_path parameter, it adds a -f argument including the From address to the command line. Any other arguments specified in the sendmail_args parameter are also added to the Sendmail command line. The -i argument is useful because without it Sendmail considers a period by itself on a line as the end of the message. Be careful not to include any external input that could contain shell metacharacters in the sendmail_args parameter—this data ends up being passed to the shell for execution. The sendmail_path and sendmail_args parameters are the only two parameters that the sendmail driver understands.

Sending MIME Mail Messages with Mail_mime

When SMTP servers exchange messages, they expect those messages to consist of plain ASCII text. No binary data or control characters are allowed. MIME is a way to get around that limitation. The MIME standard defines two things that are useful when sending e-mail messages: how to represent documents such as images or PDFs as plain text and how to package multiple documents in one message.

Every kind of document has a MIME media type, which is divided into two parts. The first part, the *type*, tells what kind of general category a document falls into (for example, image, video, or text). The second part of a media type, the *subtype*, describes the kind of document more specifically (for example, jpeg, mpeg, or html). A MIME media type is represented with the type and subtype joined by a slash: image/jpeg, video/mpeg, or text/html. Table 9-2 lists some common MIME media types. A directory of official MIME media types is available at http://www.iana.org/assignments/media-types/index.html.

Table 9-2. Some Common MIME Media Types

Media Type	Description
text/plain	ASCII text without markup or special formatting
text/html	HTML
text/csv	Tabular comma-separated value data
image/jpeg	JPEG image
image/gif	GIF image
image/png	PNG image
audio/wav	WAV sound
audio/mpeg	MP3 sound file

Table 9-2. Some Common MIME Media Types (continued)

Media Type	Description
video/mpeg	MPEG video
video/quicktime	QuickTime video
application/msword	Microsoft Word document

In addition to the MIME media types that describe individual documents, there are a set of media types called *composite types*, such as `multipart/alternative` and `multipart/mixed`. A document with a composite type holds inside it one or more other documents.

For example, an e-mail message with attachments has the media type `multipart/mixed`. Inside the `multipart/mixed` document are sections for the message body and each attachment. Each of these sections has its own media type.

Attachments

Although constructing documents with the right media type and the correct encoding of binary data can be a hassle, the PEAR Mail_mime module makes it a snap. It provides methods that produce the proper message headers and body, which can be passed to the Mail module's `send()` method for delivery. Listing 9-4 shows how to use Mail_mime to send a message with an attachment.

Listing 9-4. Sending an E-mail Message with an Attachment

```
require 'Mail.php';
require 'Mail/mime.php';
$to = 'destination@example.com';
$headers = array('From' => 'icecream@example.com',
                 'Subject' => 'Hungry?',
                 'Cc' => 'other-destination@example.com');
$body=<<<_MSG_
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon? The attached document lists
the numerous health benefits of this delicious flavor.

Sincerely,
Your local ice cream booster
_MSG_;
```

```
$mime = new Mail_mime();
$mime->setTXTBody($body);
$mime->addAttachment('/home/icecream/guava-health.doc','application/msword');
$body = $mime->get();
$headers = $mime->headers($headers);
$mailer =& Mail::factory('mail');
$mailer->send($to,$headers,$body);
```

The `setTXTBody()` method sets the body of the message to the text in `$body`. You can also give `setTXTBody()` a filename to use the contents of a file as the body of the message. Pass the filename as the first argument and `true` as the second argument:

```
// Uses the contents of /home/icecream/healthy.txt as the message body
$mime->setTXTBody('/home/icecream/healthy.txt', true);
```

The `setTXTBody()` method also supports creating the body incrementally. When you pass `true` as a third argument, `setTXTBody()` appends the new content to the message body. You can combine data in a variable and the contents of a file with multiple calls to `setTXTBody()`:

```
// Start off with the contents of a file
$mime->setTXTBody('/home/icecream/header.txt', true);
// Append a string
$mime->setTXTBody('The date today is ' . strftime('%c') . "\n\n", false, true);
// Append a file
$mime->setTXTBody('/home/icecream/healthy.txt', true, true);
// Append another file
$mime->setTXTBody('/home/icecream/footer.txt', true, true);
```

The `addAttachment()` method adds an attachment to the message. To add a file as an attachment, you need to pass `addAttachment()` two arguments: the filename and the MIME media type of the file. In Listing 9-4, the media type is `application/msword` because `guava-health.doc` is a Microsoft Word document.

The `get()` method of the `Mail_mime` object returns the body of the entire message, attachments and all, properly encoded as plain text. The `headers()` method returns an array of headers. You pass `headers()` an array containing any headers you want to add, and it returns a new array that contains the passed-in headers along with the additional headers required for the MIME encoding of the message. You must call `get()` before `headers()` because `get()` computes information about the content of the message that `headers()` needs to function properly. The return values from get() and headers() are formatted such that you can pass them directly to the Mail object's send() method.

Instead of passing a filename to addAttachment(), you can provide it with a variable that contains data you want to use as an attachment. You must supply a filename for the attachment to be known as in the third argument. Passing false as the fourth argument tells addAttachment() that its first argument holds the content of the attachment, not a filename. Listing 9-5 demonstrates sending a CSV file as a dynamic attachment.

Listing 9-5. Sending a Dynamic Attachment

```
require 'Mail.php';
require 'Mail/mime.php';
require 'DB.php';
$dbh = DB::connect('mysql://phpgems:phpgems1@localhost/phpgems');
$flavors = $dbh->getAll('SELECT * FROM ice_cream');
// The tocsv() function, defined below, turns an array of arrays
// into a string holding CSV data
$flavors_csv = tocsv($flavors);
$headers = array('From' => 'flavordept@example.com',
                 'Subject' => 'Flavor Matrix');
$to = 'flavor-summit-list@example.com';
$body=<<<_MSG_
The attached spreadsheet contains the current flavor matrix.
Please review it in detail before the flavor summit meeting tomorrow.

Sincerely,
The flavor department
_MSG_;

$mime = new Mail_mime();
$mime->setTXTBody($body);
// Since $flavors_csv holds the attachment data, we must supply a filename
// for the e-mail client to use and tell addAttachment() that the first
// argument holds data, not a filename
$mime->addAttachment($flavors_csv,'text/csv', 'flavors.csv',false);
$body = $mime->get();
$headers = $mime->headers($headers);
$mailer =& Mail::factory('mail');
$mailer->send($to,$headers,$body);
// The utility function to CSV-format an array of arrays
```

```
function tocsv($ar) {
    $s = '';
    foreach ($ar as $row) {
        $out = array();
        foreach ($row as $cell) {
            $quote = false;
            // double any double quotes, and quote the cell
            if (false !== strpos($cell,'"')) {
                $cell = str_replace('"','""',$cell);
                $quote = true;
            }
            // also quote the cell if there is an embedded
            // newline, comma, or the cell begins or ends
            // with whitespace
            elseif ((false !== strpos($cell,"\n")) ||
                    (false !== strpos($cell,"\r")) ||
                    (false !== strpos($cell,",")) ||
                    preg_match('/^\s/',$cell) ||
                    preg_match('/\s$/',$cell)) {
                $quote = true;
            }
            // add the cell, with optional quoting, to $out
            $out[] = $quote ? '"'.$cell.'"' : $cell;
        }
        // add the whole line, joined by commas, to $s
        $s .= join(',',$out)."\n";
    }
    return $s;
}
```

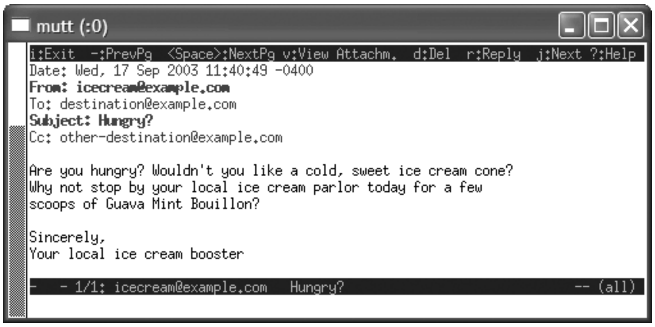
HTML Message Bodies

A message with attachments has a one main body and then one or more attachments. It is also possible to send a message with two main bodies—one that is plain text and one that is HTML. This is called a `multipart/alternative` message. The e-mail client decides which message body to display based on its capabilities. A text-based mail reader such as Pine or Mutt displays the plain-text message body, and a graphical mail reader such as Outlook or Eudora displays the HTML body. Sending a message with both plain-text and HTML bodies lets you offer a more fancily formatted version to users with capable mail readers but also accommodates people who can't (or don't like to) have their e-mail jazzed up by HTML. To include text and HTML alternatives, call both `setTXTBody()` and `setHTMLBody()` on the Mail_mime object. Listing 9-6 shows how send a message with both text and HTML bodies.

Listing 9-6. Sending a Message with Text and HTML Bodies

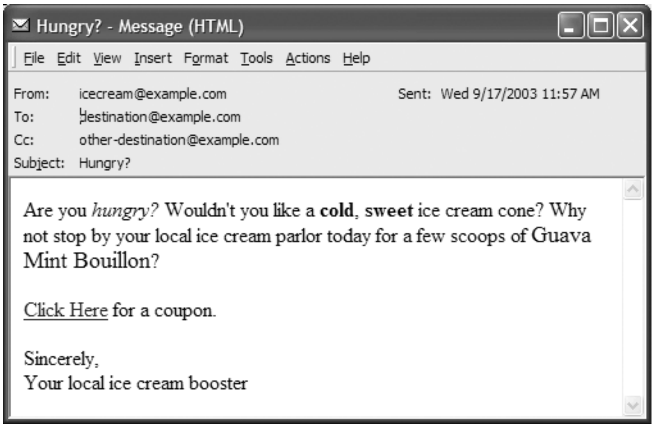
```
require 'Mail.php';
require 'Mail/mime.php';
$to = 'destination@example.com';
$headers = array('From' => 'icecream@example.com',
                 'Subject' => 'Hungry?',
                 'Cc' => 'other-destination@example.com');
$text_body=<<<_MSG_
Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster
_MSG_;

$html_body=<<<_HTML_
<p>Are you <i>hungry?</i> Wouldn't you like a <b>cold</b>, <b>sweet</b>
ice cream cone? Why not stop by your local ice cream parlor today for
a few scoops of <font size="+1">Guava Mint Bouillon</font>?</p>

<p><a href="http://coupons.example.com/">Click Here</a> for a coupon.</p>

<p>Sincerely, <br/>
Your local ice cream booster</p>
_HTML_;

$mime = new Mail_mime();
$mime->setTXTBody($text_body);
$mime->setHTMLBody($html_body);
$body = $mime->get();
$headers = $mime->headers($headers);
$mailer =& Mail::factory('mail');
$res = $mailer->send($to, $headers, $body);

// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

Figure 9-1 shows what the message from Listing 9-6 looks like in Mutt, which displays the text body. Figure 9-2 shows how Outlook displays the HTML body.



```
i:Exit  -:PrevPg  <Space>:NextPg v:View Attachm.  d:Del  r:Reply  j:Next ?:Help
Date: Wed, 17 Sep 2003 11:40:49 -0400
From: icecream@example.com
To: destination@example.com
Subject: Hungry?
Cc: other-destination@example.com

Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster

-   - 1/1: icecream@example.com   Hungry?                          -- (all)
```

Figure 9-1. Mutt displays the text part of a `multipart/alternative` *message.*



From: icecream@example.com Sent: Wed 9/17/2003 11:57 AM
To: destination@example.com
Cc: other-destination@example.com
Subject: Hungry?

Are you *hungry?* Wouldn't you like a **cold**, **sweet** ice cream cone? Why not stop by your local ice cream parlor today for a few scoops of Guava Mint Bouillon?

Click Here for a coupon.

Sincerely,
Your local ice cream booster

Figure 9-2. Outlook displays the HTML part of a `multipart/alternative` *message.*

The `setHTMLBody()` method supports the same syntax as `setTXTBody()` to specify content in a file instead of in a variable:

```
// Use the contents of /home/icecream/healthy.html as the HTML body
// of the message
$mime->setHTMLBody('/home/icecream/healthy.html', true);
```

Unlike `setTXTBody()`, however, you cannot append content with multiple calls to `setHTMLBody()`. If you call `setHTMLBody()` twice, the content from the second call replaces the content from the first.

Embedded Images

A message with an HTML body can include embedded images. To embed an image in an e-mail message, link to the image with a standard `<img>` tag in the HTML portion of the message and then add the image to the message with the `addHTMLImage()` method. Listing 9-7 sends a message whose HTML portion references two images.

Listing 9-7. Sending a Message with Embedded Images

```
require 'Mail.php';
require 'Mail/mime.php';
$to = 'destination@example.com';
$headers = array('Subject' => 'Hungry?', 'From' => 'icecream@example.com',
                 'Cc' => 'other-destination@example.com');
$text_body=<<<_MSG_
Are you hungry? Wouldn't you like a cold, sweet ice cream cone?
Why not stop by your local ice cream parlor today for a few
scoops of Guava Mint Bouillon?

Sincerely,
Your local ice cream booster
_MSG_;

$html_body=<<<_HTML_
<img src="header.gif" width="250" height="20"/>
<br clear="all"/>
<p>Are you <i>hungry?</i> Wouldn't you like a <b>cold</b>, <b>sweet</b>
ice cream cone? Why not stop by your local ice cream parlor today for
a few scoops of <font size="+1">Guava Mint Bouillon</font>?</p>

<p>Print out this coupon for 50% off a large cone:</p>
<img src="coupon.jpg"/>

<p>Sincerely, <br/>
Your local ice cream booster</p>
_HTML_;

$mime = new Mail_mime();
$mime->setTXTBody($text_body);
$mime->setHTMLBody($html_body);
$mime->addHTMLImage('/usr/local/images/header.gif','image/gif');
$mime->addHTMLImage('/usr/local/images/coupon.jpg','image/jpeg');
$body = $mime->get();
```

```
$headers = $mime->headers($headers);
$mailer =& Mail::factory('mail');
$res = $mailer->send($to, $headers, $body);

// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

The two required arguments to `addHTMLImage()` are the filename of the image to add to the message and the MIME media type of the image. For each image added to the message, Mail_mime does two things. First, it adds a new section containing the image to the message. This section has appropriate headers that identify the image's MIME media type and filename. The image section also gets a `Content-ID` header, which is a unique string to identify the section. The second thing that happens is that any instances of the image's filename in the HTML message body are replaced with the `Content-ID` value for the image. For example, `<img src="header.gif" width="250" height="20"/>` becomes `<img src="cid:5e74ee95a04dc3d3c740af4f677f0dc7" width="250" height="20"/>`. This new `src` attribute for the `<img>` tag tells a mail reader such as Outlook where in the MIME-encoded message it can find the appropriate image to display. Image filenames aren't just replaced inside `<img>` tags, however, but anywhere in the HTML message body. If you refer to images by filename elsewhere in your HTML, those references change as well.

When you add an image to a message with `addHTMLImage()`, only the filename portion of the path is preserved. This means you shouldn't add two images with the same name that are in different directories to the same message, for example, `/usr/local/images/header.gif` and `/usr/local/images/sports/header.gif`.

An HTML message body can also include references to images that are hosted on a remote server. Instead of sending `header.gif` along with the message, you can have an `<img>` tag that retrieves `header.gif` from your Web server: `<img src="http://images.example.com/mail/header.gif" width="250" height="20"/>`. To see this image, however, a user must be online when he views the message. Additionally, many people bristle at the privacy implications of alerting a remote server (by downloading an image) that they are looking at an e-mail message. This situation is compounded when image URLs include query string information that uniquely identifies a user. On the other hand, messages that reference external URLs are much smaller because they don't include copies of image data.

Whether you use external images or not, it's always a good idea to include a plain-text message body when you include an HTML message body. Even if the plain-text message body contains only a short explanation that the main thrust of the message is located in the HTML section, it makes things much clearer for users who can't see the HTML. If you're sending out a newsletter that can be delivered in either plain text or HTML, a good solution is to ask users when they sign up for the newsletter whether they would like to receive the plain-text or HTML version. Send a regular non-MIME-encoded plain-text message to users who opt for plain text. To users who opt for HTML, send a message with the newsletter content in the HTML body and a short text body telling them what URL to visit if they want to change their subscription to plain text. This saves bandwidth because you don't have to send every user both the text and HTML copies of the message.

Like the `addAttachment()` method, `addHTMLImage()` also accommodates image data stored in a variable instead of a file. This is good for generating an image dynamically and then embedding it in a message.

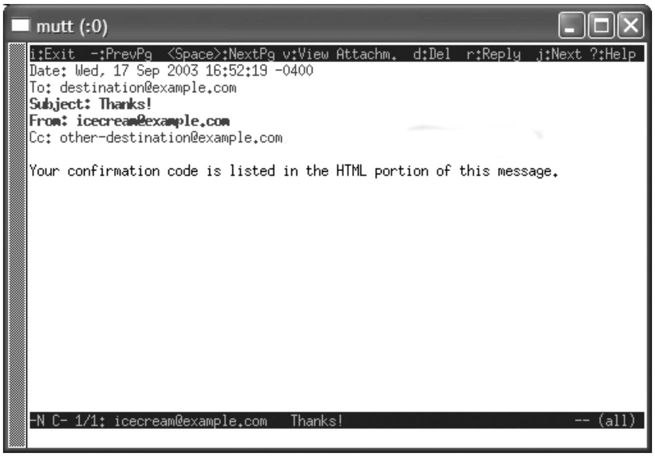
Listing 9-8 shows how to generate an embedded image with functions from the GD graphics drawing extension to send a registration confirmation code in an e-mail message. Sending the code in an image instead of as plain text thwarts many programs that could otherwise parse the message and automatically complete the registration.

Listing 9-8. Sending a Dynamic Image

```
require 'Mail.php';
require 'Mail/mime.php';
$to = 'destination@example.com';
$headers = array('Subject' => 'Thanks!',
                 'From' => 'icecream@example.com',
                 'Cc' => 'other-destination@example.com');

$text_body=<<<_MSG_
Your confirmation code is listed in the HTML portion of this message.
_MSG_;
$html_body=<<<_HTML_
<p>Thanks for signing up with the Free Ice Cream Club. To prevent
automated registrations, please visit the confirmation URL and
enter the confirmation code displayed in the image below:</p>
<img src="conf-code.png"/>
<br clear="all"/>
<p>The confirmation URL is:</p>
<p><b><a href="http://icecream.example.com/confirm.php">
http://icecream.example.com/confirm.php</a></b></p>
_HTML_;

// Create the confirmation code as three sets of five letters,
// separated by hyphens
$code = ''; $code_length = 15;
while ($code_length--) { $code.= chr(65+mt_rand(0,25)); }
$code = substr($code,0,5) . '-' . substr($code,5,5) . '-' . substr($code,10,5);
```

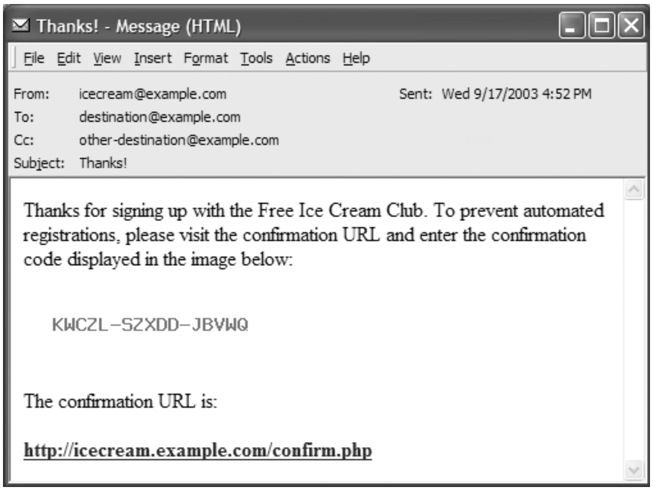
```
// Use GD to create a 200x50 image with the confirmation code
// in the middle of it. The imagecreatetruecolor() function
// is for GD 2.0.1 or later. Use imagecreate() if you have
// an earlier version of GD.
$im = imagecreatetruecolor(200,50);
$bg_color = imagecolorallocate($im, 0xff, 0xff, 0xff);
$text_color = imagecolorallocate($im, 102, 153, 0);
imagefill($im, 0, 0, $bg_color);
imagestring($im, 5, 23, 15, $code, $text_color);

// GD has no way to put the image data directly into a string, so we use
// output buffering to capture what GD would otherwise send to the browser
ob_start();
imagepng($im);
$image_data = ob_get_contents ();
ob_end_clean();
imagedestroy($im);

// Adding the dynamic image to the message is like adding an image file, but
// with two more arguments to addHTMLImage()
$mime = new Mail_mime();
$mime->setTXTBody($text_body);
$mime->setHTMLBody($html_body);
$mime->addHTMLImage($image_data,'image/png','conf-code.png',false);
$body = $mime->get();
$headers = $mime->headers($headers);
$mailer =& Mail::factory('mail');
$res = $mailer->send($to, $headers, $body);

// If the message can't be sent, send() returns a
// PEAR::Error object
if (PEAR::isError($res)) {
   print "Couldn't send message: " . $res->getMessage();
}
```

When adding a dynamic image with `addHTMLImage()`, four arguments are required. The first argument is the variable holding the image data, and the second argument is the MIME media type of the image. The third argument is the filename to use inside the message for the image. In Listing 9-8, the filename is `conf-code.png` to match the filename in the `src` attribute of the `<img>` tag in the HTML message body. The last argument, `true`, tells `addHTMLImage()` that the first argument holds image data, not the filename of an image. Figure 9-3 shows what the message generated by Listing 9-8 looks like in a text-based e-mail client, and Figure 9-4 shows the message in a graphical e-mail client.


mutt (:0)
i:Exit -:PrevPg <Space>:NextPg v:View Attachm. d:Del r:Reply j:Next ?:Help

```
Date: Wed, 17 Sep 2003 16:52:19 -0400
To: destination@example.com
Subject: Thanks!
From: icecream@example.com
Cc: other-destination@example.com

Your confirmation code is listed in the HTML portion of this message.

-N C- 1/1: icecream@example.com   Thanks!                          -- (all)
```

Figure 9-3. Mutt displays the text part of the message.



From: icecream@example.com Sent: Wed 9/17/2003 4:52 PM
To: destination@example.com
Cc: other-destination@example.com
Subject: Thanks!

Thanks for signing up with the Free Ice Cream Club. To prevent automated registrations, please visit the confirmation URL and enter the confirmation code displayed in the image below:

KWCZL-SZXDD-JBVWQ

The confirmation URL is:

http://icecream.example.com/confirm.php

Figure 9-4. Outlook displays the HTML part of the message, including the dynamic image.

A confirmation code embedded in an image is harder for an automated program to process than a plain-text confirmation code, but not impossible.

For more information about distorted embedded text and other mechanisms that are more difficult for computers to parse, visit the CAPTCHA project at `http://www.captcha.net/`.