CHAPTER 11

PEAR

Good programmers write solid code, while great programmers reuse the code of good programmers. For PHP programmers, *PEAR* (`http://pear.php.net`), acronym for *PHP Extension and Application Repository,* is one of the most effective means for finding and reusing good PHP code. Inspired by Perl's wildly popular CPAN (`http://www.cpan.org`), the project was started in 1999 by noted PHP developer Stig Bakken, with the first stable release bundled with PHP version 4.3.0. Formally defined, PEAR is a framework and distribution system for reusable PHP components, and presently offers 442 *packages* categorized under 41 different topics (and increasing all the time). Because PEAR contributions are carefully reviewed by the community before they're accepted, code quality and adherence to PEAR's standard development guidelines are assured. Furthermore, because many PEAR packages logically implement common tasks guaranteed to repeatedly occur no matter the type of application, taking advantage of this community-driven service will save you countless hours of programming time.

This chapter is devoted to a thorough discussion of PEAR, offering the following topics:

- A survey of several popular PEAR packages, intended to give you an idea of just how useful this repository can really be.
- Instructions regarding the installation and administration of PEAR packages via the PEAR console.
- A discussion of PEAR coding and documentation guidelines, which could prove useful not only for building general applications but also for reviewing and submitting PEAR packages.
- An overview of the PEAR submission process, should you be interested in making your own contributions to the repository.

Popular PEAR Packages

To give you a taste of just how popular the PEAR packages are, at the time of this writing the hosted packages have been downloaded almost 14 million times to date! In fact, several packages are so popular that the developers started including them by default as of version 4.0. A list of the presently included packages follows:

- `Archive_Tar`: The `Archive_Tar` package facilitates the management of tar files, providing methods for creating, listing, extracting, and adding to tar files. Additionally, it supports the Gzip and Bzip2 compression algorithms, provided the respective PHP extensions are installed. This package is required for PEAR to run properly.

- `Console_Getopt`: It's often useful to modify the behavior of scripts executed via the command line by supplying options at execution time. For example, you can verify the installed PEAR version by passing `-V` to the `pear` command:

  ```
  %>pear -V
  ```

  The `Console_Getopt` package provides a standard means for reading these options and providing the user with error messages if the supplied syntax does not correspond to some predefined specifications (such as whether a particular argument requires a parameter). This package is required for PEAR to run properly.

- `DB`: The `DB` package provides an object-oriented query API for abstracting communication with the database layer. This affords you the convenience of transparently migrating applications from one database to another potentially as easily as modifying a single line of code. At present there are 12 supported databases, including: dBase, FrontBase, Informix, InterBase, Mini SQL, Microsoft SQL Server, MySQL, Oracle, ODBC, PostgreSQL, SQLite, and Sybase.

- `Mail`: Writing a portable PHP application that is capable of sending e-mail may be trickier than you think, because not all operating systems offer the same facilities for supporting this feature. For instance, by default, PHP's `mail()` function relies on the sendmail program (or a sendmail wrapper), but sendmail isn't available on Windows. To account for this incompatibility, it's possible to alternatively specify the address of an SMTP server and send mail through it. However, how would your application be able to determine which method is available? The `Mail` package resolves this dilemma by offering a unified interface for sending mail that doesn't involve modifying PHP's configuration. It supports three different back ends for sending e-mail from a PHP application (PHP's `mail()` function, sendmail, and an SMTP server) and includes a method for validating e-mail address syntax. Using a simple application configuration file or Web-based preferences form, users can specify the methodology that best suits their needs.

- `Net_Socket`: The `Net_Socket` package is used to simplify the management of TCP sockets by offering a generic API for carrying out connections, and reading and writing information between these sockets.

- `Net_SMTP`: The `Net_SMTP` package offers an implementation of the SMTP protocol, making it easy for you to carry out tasks such as connecting to and disconnecting from SMTP servers, performing SMTP authentication, identifying senders, and sending mail.

- `PEAR`: This package is required for PEAR to run properly.

- `PHPUnit`: A unit test is a particular testing methodology for ensuring the proper operation of a block (or unit) of code, typically classes or function libraries. The `PHPUnit` package facilitates the creation, maintenance, and execution of unit tests by specifying a general set of structural guidelines and a means for automating testing.

- `XML_Parser`: The XML_Parser package offers an easy, object-oriented solution for parsing XML files.
- `XML_RPC`: The `XML_RPC` package is a PHP-based implementation of the XML-RPC protocol (`http://www.xmlrpc.com/`), a means for remotely calling procedures over the Internet. Using this package, you can create XML-RPC-based clients and servers. This package is required for PEAR to run properly.

While the preceding packages are among the most popular, keep in mind that they are just a few of the packages available via PEAR. A few other prominent packages follow:

- `Auth`: The `Auth` package facilitates user authentication across a wide variety of mechanisms, including LDAP, POP3, IMAP, RADIUS, SOAP, and others.
- `HTML_QuickForm`: The `HTML_QuickForm` package facilitates the creation, rendering, and validation of HTML forms.
- `Log`: The `Log` package offers an abstract logging facility, supporting logging to console, file, SQL, SQLite, syslog, mail, and mcal destinations.

It might not come as a surprise that the aforementioned packages are so popular. After all, if you haven't yet started taking advantage of PEAR, it's likely you've spent significant effort and time repeatedly implementing some of these features.

Converting Numeral Formats

To demonstrate the power of PEAR, it's worth calling attention to a package that exemplifies why you should regularly look to the repository before attempting to resolve any significant programming task. While some might consider this particular choice of package a tad odd, it is meant to show that a package may be available even for a particularly tricky problem that you may think is too uncommon for a package to have been developed, and thus not bother searching the repository for an available solution. The package is `Numbers_Roman`, and it makes converting Arabic numerals to Roman and vice versa a snap.

Returning to the problem, suppose you were recently hired to create a new Web site for a movie producer. As we all know, any serious producer uses Roman numerals to represent years, and the product manager tells you that any date found on the Web site must appear in this format. Take a moment to think about this requirement, because fulfilling it isn't as easy as it may sound. Of course, you could look up a conversion table online and hard code the values, but how would you ensure that the site copyright year in the page footer is always up to date? You're just about to settle in for a long evening of coding when you pause for a moment to consider whether somebody else has encountered a similar problem. "No way," you mutter, but taking a quick moment to search PEAR certainly would be worth the trouble. You navigate over and, sure enough, encounter `Numbers_Roman`.

For the purposes of this exercise, assume that the `Numbers_Roman` package has been installed on the server. Don't worry too much about this right now, because you'll learn how to install packages in the next section. So how would you go about making sure the current year is displayed in the footer? By using the following script:

```
<?php
   // Make the Numbers_Roman package available
   require_once("Numbers/Roman.php");

   // Retrieve current year
   $year = date("Y");

   // Convert year to Roman numerals
   $romanyear = Numbers_Roman::toNumeral($year);

   // Output the copyright statement
   echo "Copyright &copy; $romanyear";
?>
```

For the year 2005, this script would produce:

```
Copyright © MMV
```

The moral of this story? Even though you may think that a particular problem is obscure, other programmers likely have faced a similar problem, and if you're fortunate enough, a solution is readily available and yours for the taking.

Installing and Updating PEAR

The easiest way to manage your PEAR packages is through the PEAR Package Manager. This is a command-line program that offers a simple and efficient interface for performing tasks such as inspecting, adding, updating, and deleting packages, and browsing packages residing in the repository. In this section, you'll learn how to install and update the PEAR Package Manager on both the Unix and Windows platforms. Because many readers run Web sites on a shared hosting provider, this section also explains how to take advantage of PEAR without running the Package Manager.

Installing PEAR

PEAR has become such an important aspect of efficient PHP programming that a stable release has been included with the distribution since version 4.3.0. Therefore, if you're running this version or later, feel free to jump ahead and review the section "Updating Pear." If you're running PHP version 4.2.X or earlier on Unix, or are using the Windows platform, the installation process is trivial, as you'll soon learn.

Unix

Installing PEAR on Unix is a rather simple process, done by retrieving a script from the `http://go-pear.org/` Web site and executing it with the PHP binary. Open up a terminal and execute the following command:

```
%>lynx -source http://go-pear.org/ | php
```

Note that you need to have the lynx Web browser installed, a rather standard program on the Unix platform. If you don't have it, search the appropriate program repository for your particular OS distribution; it's guaranteed to be there. Alternatively, you can just use a standard Web browser such as Firefox and navigate to the preceding URL, save the retrieved page, and execute it using the binary.

Once the installation process begins, you'll be prompted to confirm a few configuration settings such as the location of the PHP root directory and executable; you'll likely be able to accept the default answers (provided between square brackets) without issue. During this round of questions, you will also be prompted as to whether the six optional default packages should be installed. It's presently an all-or-none proposition; therefore, if you'd like to immediately begin using any of the packages, just go ahead and accede to the request.

Windows

PEAR is not installed by default with the Windows distribution. To install it, you need to run the `go-pear.bat` file, located in the PHP distribution's root directory. This file installs the PEAR command, the necessary support files, and the aforementioned six PEAR packages. Initiate the installation process by changing to the PHP root directory and executing `go-pear.bat`, like so:

```
%>go-pear.bat
```

You'll be prompted to confirm a few configuration settings such as the location of the PHP root directory and executable; you'll likely be able to accept the default answers (provided between square brackets) without issue. During this round of questions, you will also be prompted as to whether the six optional default packages should be installed. It's presently an all-or-none proposition; therefore, if you'd like to immediately begin using any of the packages, just go ahead and accede to the request.

At the conclusion of the installation process, a registry file named `PEAR_ENV.reg` is created. Executing this file will create environment variables for a number of PEAR-specific variables. Although not critical, adding these variables to the system path affords you the convenience of executing the PEAR Package Manager from any location while at the Windows command prompt.

■ **Caution** Executing the `PEAR_ENV.reg` file will modify your system registry. Although this particular modification is innocuous, you should nonetheless consider backing up your registry before executing the script. To do so, go to Start ➤ Run, execute `regedit`, and then export the registry via File ➤ Export.

PEAR and Hosting Companies

If your hosting company doesn't allow users to install new software on its servers, don't fret, because it likely already offers at least rudimentary support for the most prominent packages. If PEAR support is not readily obvious, contact customer support and inquire as to whether they would consider making a particular package available for use on the server. If they accede, you're all set. If they deny your request, not to worry, because it's still possible to use the packages,

although installing them is accomplished by a somewhat more manual mechanism. This process is outlined in the later section, "Installing a PEAR Package."

Updating PEAR

Although it's been around for years, the PEAR Package Manager is constantly the focus of ongoing enhancements. That said, you'll want to occasionally check for and update the system. Doing so is a trivial process on both the Unix and Windows platforms, done by executing the `go-pear.php` script found in the `PHP_INSTALLATION_DIR\PEAR` directory:

```
%>php go-pear.php
```

Executing this command essentially restarts the installation process, overwriting the previously installed Package Manager version.

Using the PEAR Package Manager

The PEAR Package Manager allows you to browse and search the contributions, view recent releases, and download packages. It executes via the command line, using the following syntax:

```
%>pear [options] command [command-options] <parameters>
```

To get better acquainted with the Package Manager, open up a command prompt and execute the following:

```
%>pear
```

You'll be greeted with a list of commands and some usage information. This output is pretty long, so we'll forego reproducing it here and instead introduce just the most popular commands available to you. Note that, because the intent of this chapter is to familiarize you with only the most commonplace PEAR features, this introduction is not exhaustive. Therefore, if you're interested in learning more about one of the commands not covered in the remainder of this chapter, execute that command in the Package Manager, supplying the `help` parameter like so:

```
%>pear help <command>
```

■ **Tip** If PEAR doesn't execute because the command was not found, you need to add the PEAR directory to your system path.

Viewing Installed Packages

Viewing the packages installed on your machine is simple; just execute the following:

```
%>pear list
```

Here's some sample output:

```
Installed packages:
===================
Package        Version  State
Archive_Tar    1.3.1    stable
Console_Getopt 1.2      stable
DB             1.7.6    stable
HTTP           1.2.2    stable
Mail           1.1.3    stable
Net_SMTP       1.2.6    stable
Net_Socket     1.0.1    stable
PEAR           1.3.5    stable
PhpDocumentor  1.3.0RC3 beta
XML_Parser     1.0.1    stable
XML_RPC        1.2.2    stable
```

Learning More About an Installed Package

The preceding output indicates that 11 packages are installed on the server in question. However, this information is quite rudimentary and really doesn't provide anything more than the package name and version. To learn more about a package, execute the `info` command, passing it the package name. For example, you would execute the following command to learn more about the `Console_Getopt` package:

```
%>pear info Console_Getopt
```

Here's an example of output from this command:

```
ABOUT CONSOLE_GETOPT-1.2
========================
Provides        Classes: Console_Getopt
Package         Console_Getopt
Summary         Command-line option parser
Description     This is a PHP implementation of "getopt"
                supporting both short and long options.
Maintainers     Andrei Zmievski <andrei@php.net> (lead)
                Stig Bakken <stig@php.net> (developer)
Version         1.2
Release Date    2003-12-11
Release License PHP License
Release State   stable
Release Notes   Fix to preserve BC with 1.0 and allow correct
                behaviour for new users
Last Modified   2005-01-23
```

As you can see, this output offers some very useful information about the package.

Installing a Package

Installing a PEAR package is a surprisingly automated process, accomplished simply by executing the `install` command. The general syntax follows:

```
%>pear install [options] package
```

Suppose for example that you want to install the `Auth` package, first introduced earlier in this chapter. The command and corresponding output follows:

```
%>pear install Auth
```

```
pear install auth
downloading Auth-1.2.3.tgz ...
Starting to download Auth-1.2.3.tgz (24,040 bytes)
........done: 24,040 bytes
Optional dependencies:
package 'File_Passwd' version >= 0.9.5 is recommended to utilize some features.
package 'Net_POP3' version >= 1.3 is recommended to utilize some features.
package 'MDB' is recommended to utilize some features.
package 'Auth_RADIUS' is recommended to utilize some features.
package 'File_SMBPasswd' is recommended to utilize some features.
install ok: Auth 1.2.3
```

In addition to offering information regarding the installation status, many packages also present a list of optional dependencies that, if installed, will expand the available features. For example, installing the `File_SMBPasswd` package enhances `Auth`'s capabilities, enabling it to authenticate against a Samba server.

Assuming a successful installation, you're ready to begin using the package. Forge ahead to the section "Using a Package" to learn more about how to make the package available to your script. If you run into installation problems, it's almost certainly due to a failed dependency. Read on to learn how to resolve this problem.

Failed Dependency?

In the preceding example, `File_SMBPasswd` is an instance of an optional dependency, meaning it doesn't have to be installed in order to use `Auth`, although a certain subset of functionality will not be available via `Auth` until `File_SMBPasswd` is installed. However, it is also possible for there to be required dependencies involved when installing a package, if developers can save development time by incorporating existing packages into their project. For instance, because `Auth_HTTP` requires the `Auth` package in order to function, any attempt to install `Auth_HTTP` without first installing this requisite package will fail, producing the following error:

```
downloading Auth_HTTP-2.1.4.tgz ...
Starting to download Auth_HTTP-2.1.4.tgz (7,835 bytes)
.....done: 7,835 bytes
requires package 'Auth' >= 1.2.0
Auth_HTTP: Dependencies failed
```

Automatically Installing Dependencies

Of course, chances are that if you need a particular package, then installing any dependencies is a foregone conclusion. To install required dependencies, pass the `-o` (or `--onlyreqdeps`) option to the `install` command:

```
%>pear install -o Auth_HTTP
```

To install both optional and required dependencies, pass along the `-a` (or `--alldeps`) option:

```
%>pear install -a Auth_HTTP
```

Installing a Package from the PEAR Web Site

The PEAR Package Manager by default installs the latest stable package version. But what if you were interested in installing a previous package release, or were unable to use the Package Manager altogether due to administration restrictions placed on a shared server? Navigate to the PEAR Web site at `http://pear.php.net` and locate the desired package. If you know the package name, you can take a shortcut by entering the package name at the conclusion of the URL `http://pear.php.net/package/`.

Next, click on the Download tab, found toward the top of the package's home page. Doing so produces a linked list of the current package and all previous packages released. Select and download the appropriate package to your server. These packages are stored in TGZ (tar'red and gzipped) format.

Next, extract the files to an appropriate location. It doesn't really matter where, provided you're consistent in placing all packages in this tree. If you're taking this installation route because of the need to install a previous version, then it makes sense to place the files in their appropriate location within the PEAR directory structure found in the PHP root installation directory. If you're forced to take this route in order to circumvent ISP restrictions, then creating a PEAR directory in your home directory will suffice. Regardless, be sure this directory is found in the `include_path`.

The package should now be ready for use, so move on to the next section to learn how this is accomplished.

Using a Package

Using an installed PEAR package is simple. All you need to do is make the package contents available to your script with `include` or preferably `require`. Examine the following example, where PEAR `DB` package is included and used:

```
<?php
   // Make the PEAR DB package available to the script
   require_once("DB.php");

   // Connect to the database
   $db = DB::connect("mysql://jason:secret@localhost/book");
   ...
?>
```

Keep in mind that you need to add the PEAR base directory to your include_path directive; otherwise, an error similar to the following will occur:

```
Fatal error: Class 'DB' not found in /home/www/htdocs/book/11/Roman.php on line 9
```

Those of you with particularly keen eyes might have noticed in the preceding example that the require_once statement directly references the DB.php file, whereas in the earlier example involving the Numbers_Roman package, a directory was also referenced:

```
require_once("Numbers/Roman.php");
```

A directory is referenced because the Numbers_Roman package falls under the Numbers category, meaning that, for purposes of organization, a corresponding hierarchy will be created, with Roman.php placed in a directory named Numbers. You can determine the package's location in the hierarchy simply by looking at the package name. Each underscore is indicative of another level in the hierarchy, so in the case of Numbers_Roman, it's Numbers/Roman.php. In the case of DB, it's just DB.php.

■ **Note** See Chapter 2 for more information about the include_path directive.

Upgrading a Package

All PEAR packages must be actively maintained, and most are in a regular state of development. That said, to take advantage of the latest enhancements and bug fixes, you should regularly check whether a new package version is available. The general syntax for doing so looks like this:

```
%>pear upgrade [package name]
```

For instance, on occasion you'll want to upgrade the PEAR package, responsible for managing your package environment. This is accomplished with the following command:

```
%>pear upgrade pear
```

If your version corresponds with the latest release, you'll see a message that looks like:

```
Package 'PEAR-1.3.3.1' already installed, skipping
```

If for some reason you have a version that's greater than the version found in the PEAR repository (for instance, you manually downloaded a package from the author's Web site before it was officially updated in PEAR), you'll see a message that looks like this:

```
Package 'PEAR' version '1.3.3.2'  is installed and 1.3.3.1 is > requested '1.3.0',
skipping
```

Otherwise, the upgrade should automatically proceed. When completed, you'll see a message that looks like:

```
downloading PEAR-1.3.3.1.tgz ...
Starting to download PEAR-1.3.3.1.tgz (106,079 bytes)
.....................done: 106,079 bytes
upgrade ok: PEAR 1.3.3.1
```

Upgrading All Packages

It stands to reason that you'll want to upgrade all packages residing on your server, so why not perform this task in a single step? This is easily accomplished with the `upgrade-all` command, executed like this:

```
%>pear upgrade-all
```

Although unlikely, it's possible some future package version could be incompatible with previous releases. That said, using this command isn't recommended unless you're well aware of the consequences surrounding the upgrade of each package.

Uninstalling a Package

If you have finished experimenting with a PEAR package, have decided to use another solution, or have no more use for the package, you should uninstall it from the system. Doing so is trivial using the `uninstall` command. The general syntax follows:

```
%>pear uninstall [options] package name
```

For example, to uninstall the `Numbers_Roman` package, execute the following command:

```
%>pear uninstall Numbers_Roman
```

Because the options are fairly rarely used, you can perform additional investigation on your own, by executing:

```
%>pear help uninstall
```

Downgrading a Package

There is no readily available means for downgrading a package via the Package Manager. To do so, download the desired version via the PEAR Web site (`http://pear.php.net`), which will be encapsulated in TGZ format, uninstall the presently installed package, and then install the downloaded package using the instructions provided in the earlier section, "Installing a Package."

Summary

PEAR can be a major catalyst for quickly creating PHP applications. Hopefully this chapter convinced you of the serious time savings this repository can present. You learned about the PEAR Package Manager, and how to manage and use packages.

Forthcoming chapters introduce additional packages, as appropriate, showing you how these packages can really speed development and enhance your application's capabilities.