# Foundations of PEAR

## Rapid PHP Development

Nathan A. Good and Allan Kent

Apress®

**Foundations of PEAR: Rapid PHP Development**



ISBN-13 (pbk): 978-1-59059-739-2

ISBN-10 (pbk): 1-59059-739-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1



Lead Editor: Matt Wade
Technical Reviewer: Ligaya Turmelle
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Denise Santoro Lincoln
Copy Edit Manager: Nicole Flores
Copy Editor: Susannah Davidson Pfalzer
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Susan Glinert
Proofreader: Lori Bring
Indexer: John Collin
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

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, e-mail `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 Source Code/Download section.

# Introduction

This first chapter of the book discusses how to install PEAR and how to use the PEAR package manager. It also discusses how to configure PEAR. If you've already been using the PEAR package manager and some PEAR packages, feel free to skip the first part of this chapter or lightly skim it for review. The second part of this chapter discusses PEAR_Error, a package that you'll run into and use when you're adding exception handling to your application using PEAR packages.

## Installing PEAR

Fortunately, if you're using PHP 5 in a Unix or Linux environment, the PEAR package manager is automatically installed as part of PHP—that is, unless you've specifically compiled PHP with the `--without-pear` option. If you compiled PHP one night at three in the morning while writing your own port of Scorched Earth in assembler in a different window, and can't remember what configuration options you used, check the `info.php` file that's mentioned in the "Prerequisites" section.

As long as the PHP `bin` directory is part of your path, you can access the package manager in a Unix or Linux environment by typing `pear` at the command line.

Installing the PEAR package manager on Windows requires a few more manual steps. First, you must have installed PHP from the ZIP package and *not* from the installer package (a Windows MSI). The Windows installer doesn't include the `go-pear.bat` script referenced here to begin the installation of PEAR.

After extracting the PHP from the ZIP archive, the first step is to execute the `go-pear.bat` file that's located in the directory PHP is installed in. Once you have the `PATH` variable set correctly, you can access the package manager from the command line by typing `pear`.

```
> cd \PHP
> go-pear.bat
```

When you initiate the `go-pear.bat` script from the command line, it will start by prompting you:

```
Are you installing a system-wide PEAR or a local copy?
(system|local) [system] :
```

At this prompt, just hit Enter to continue, unless you need to change `system` to `local`. You'll then be confronted with the following:

```
Below is a suggested file layout for your new PEAR installation.  To
change individual locations, type the number in front of the
directory.  Type 'all' to change all of them or simply press Enter to
accept these locations.

 1. Installation base ($prefix)   : C:\php
 2. Binaries directory            : C:\php
 3. PHP code directory ($php_dir) : C:\php\pear
 4. Documentation directory       : C:\php\pear\docs
 5. Data directory                : C:\php\pear\data
 6. Tests directory               : C:\php\pear\tests
 7. Name of configuration file    : C:\WINNT\pear.ini
 8. Path to CLI php.exe           : C:\php\.

1-8, 'all' or Enter to continue:
```

At this point, you can change one or more of the directories to make sure they're correct. You might consider leaving everything as the default, as long as the PHP directory is set up correctly. Once you're finished, the PEAR setup will continue.

You might be asked to modify the `include_path` setting in your `php.ini` file—make sure that the `include_path` directive includes the directory where PEAR is installed. After you've made this change, the `go-pear.bat` script will finish and you can access the package manager by typing `pear` at the Windows command prompt.

# Configuring PEAR

You can use configuration commands with the `pear` command to set up channels, directories, and proxy settings, to name a few options. To view your current configuration settings, type the following:

```
$ pear config-show
```

The first few lines of the results will look something like this:

```
Configuration (channel pear.php.net):
=====================================
Auto-discover new Channels     auto_discover    <not set>
Default Channel                default_channel  pear.php.net
HTTP Proxy Server Address      http_proxy       <not set>
...
```

The first column contains the description of the configuration key, the second column is the key itself, and the last column contains the value. If you'd like more information about any of the keys, just pass the name of the key along with the `config-help` command, like this:

```
$ pear config-help http_proxy
```

The output of this command will look something like this:

```
Config help for http_proxy
==========================
Name       Type   Description
http_proxy string HTTP proxy (host:port) to use when
                  downloading packages
```

To display and set a single configuration value, use the `config-get` and `config-set` commands, respectively. The next few examples show you how to display, set, and redisplay a setting that could be useful for you if you begin to do advanced work with the `pear` command:

```
$ pear config-help verbose
```

This displays the help for the `verbose` key, which is used to set the level of debugging output:

```
Config help for verbose
=======================
Name    Type    Description
verbose integer verbosity level
                0: really quiet
                1: somewhat quiet
                2: verbose
                3: debug
```

Using the `config-get` command, you can display the current value of the `verbose` setting by typing the following at the command prompt:

```
$ pear config-get verbose
```

The current level will be displayed:

```
1
```

Now, set the `verbose` configuration setting by using the `config-set` command, passing the value as the last parameter. The command will look like this, bumping up the `verbose` setting to "debug":

```
$ pear config-set verbose 3
```

```
config-set succeeded
```

Just to show that you aren't being ignored, use the `config-get` command again to see how the `verbose` configuration setting has been updated:

```
$ pear config-get verbose
```

```
3
```

The location of the configuration file used by PEAR is listed at the end of the output of the `config-show` command. Be forewarned—the file is pretty obscure to be editing by hand and we don't recommend it. If your configuration file becomes corrupt for some reason, you can create a new default one by using the `config-create` command:

```
$ pear config-create /usr/local/php5 pear.conf
```

You can use the `-w` option for Windows installations. As mentioned in the help for this command, it's also useful when creating configuration files for remote PEAR installations.

For the most part, after installation you won't need to adjust the PEAR configuration often.

# Using the PEAR Package Manager

The PEAR package manager allows you to install packages without having to worry about installing dependencies, placing files in particular locations, or tracking them down. You can use a number of commands, although if you're not building your own packages you won't use many of them.

In the table here we've listed the commands that will be covered in this book to install packages. We've also listed a couple of the commands that can be used to list available packages and update your currently installed ones.

*Pear commands*

| Command | Description |
|---|---|
| download | This command allows you to download a package without installing it. You might have used similar commands when banished to a dial-up when the download might take a long time. |
| info | This command prints information about the specified package. |
| install | This command downloads the package and puts it onto your system. |
| list | This command lists all the installed packages on your system. |
| list-all | This command lists all the available packages. You can specify a channel with the `--channel=` option. |
| package-dependencies | This command lists all the dependencies that the given package has. |
| run-scripts | Some packages include scripts that can be run after they've been installed; this command executes those scripts. |
| search | This command allows you to search for a particular package given keywords. |

*Pear commands*

| Command | Description |
|---|---|
| uninstall | This command allows you to remove a package from the system once it's installed. |
| upgrade | This command allows you to bring a specific package up to the latest version of the package. |
| upgrade-all | This command allows you to bring all your installed packages up to the latest version of the packages. |

## Finding a Package

If you're looking for a PEAR package to install, you have a couple different ways to find it. You can use the `list-all` command to get a comprehensive list of packages and scan through the list, or you can pipe the output to another command such as `grep`, or you can use the `search` command to find matching packages. The example here shows how to `search` for services:

```
$ pear search "Services"
```

```
=======================================
Package                Stable/(Latest)         Local
Services_Amazon        -n/a-/(0.4.0 beta)            Provides access to ➥
                                                     Amazon's retail and associate➥
                                                     web services
Services_Delicious     -n/a-/(0.2.0beta beta)        Client for the del.icio.us ➥
                                                     web service.
Services_DynDNS        -n/a-/(0.3.1 alpha)           Provides access to the ➥
                                                     DynDNS web service
Services_Ebay          -n/a-/(0.12.0 alpha)          Interface to eBay's XML-API.
Services_ExchangeRates -n/a-/(0.5.3 beta)            Performs currency conversion
Services_Google        -n/a-/(0.1.1 alpha)           Provides access to the ➥
                                                     Google Web APIs
Services_Pingback      -n/a-/(0.2.0dev2 devel)       A Pingback User-Agent class.
Services_Technorati    -n/a-/(0.6.6beta beta)        A class for interacting ➥
                                                     with the Technorati API
Services_Trackback     -n/a-/(0.5.1 alpha)           Trackback - A generic ➥
                                                     class for sending and ➥
                                                     receiving trackbacks.
Services_Weather       1.3.2/(1.3.2 stable)          This class acts as an ➥
                                                     interface to various online➥
                                                     weather-services.
Services_Webservice    -n/a-/(0.4.0 alpha)           Create webservices
Services_Yahoo         -n/a-/(0.1.1 alpha)           Provides access to the ➥
                                                     Yahoo! Web Services
```

## Installing a Package

Once you've found a package that you want to use, use the `install` command and pass it the package name that you found using `search`.

Here's an example of installing an interface for weather services called Services_Weather. Of course, you'll want to make sure that PEAR handles the dependencies for you—that's one of the benefits of using a package manager—so make sure to specify the `--alldeps` option.

```
$ pear install --alldeps Services_Weather
```

```
Failed to download pear/SOAP, version "0.7.5", latest release is version 0.9.3, ➥
stability "beta", use "channel://pear.php.net/SOAP-0.9.3" to install
Failed to download pear/XML_Serializer, version "0.8", latest release is ➥
version 0.18.0, stability "beta", use ➥
"channel://pear.php.net/XML_Serializer-0.18.0" to install
pear/Services_Weather can optionally use package "pear/SOAP" (version >= 0.7.5)
pear/Services_Weather can optionally use package "pear/XML_Serializer" ➥
(version >= 0.8)
downloading Services_Weather-1.3.2.tgz ...
Starting to download Services_Weather-1.3.2.tgz (44,849 bytes)
.....done: 44,849 bytes
downloading Cache-1.5.4.tgz ...
Starting to download Cache-1.5.4.tgz (30,690 bytes)
...done: 30,690 bytes
downloading DB-1.7.6.tgz ...
Starting to download DB-1.7.6.tgz (124,807 bytes)
...done: 124,807 bytes
downloading HTTP_Request-1.3.0.tgz ...
Starting to download HTTP_Request-1.3.0.tgz (13,808 bytes)
...done: 13,808 bytes
downloading Net_URL-1.0.14.tgz ...
Starting to download Net_URL-1.0.14.tgz (5,173 bytes)
...done: 5,173 bytes
downloading Net_Socket-1.0.6.tgz ...
Starting to download Net_Socket-1.0.6.tgz (4,623 bytes)
...done: 4,623 bytes
install ok: channel://pear.php.net/Net_URL-1.0.14
install ok: channel://pear.php.net/Net_Socket-1.0.6
install ok: channel://pear.php.net/HTTP_Request-1.3.0
install ok: channel://pear.php.net/DB-1.7.6
install ok: channel://pear.php.net/Cache-1.5.4
install ok: channel://pear.php.net/Services_Weather-1.3.2
```

As you can see, the PEAR package manager downloads the dependencies and installs them in the correct order for you so you don't have to worry about them. Another option that you can use is `--onlyreqdeps`, which unlike `--alldeps` only downloads the dependencies that are absolutely required for the package to be installed. If the `--onlyreqdeps` option would have been

used to install the Services_Weather package, no other packages would have been installed because all of them are optional dependencies.

## Upgrading a Package

If you want to upgrade any package that you already have installed, use the `update` command and specify the package:

```
$ pear update Services_Weather
```

The `update-all` command updates all the packages you have installed:

```
$ pear update-all
```

## Working with Channels

A new feature as of PEAR 1.4.0, channels are ways of organizing packages. Using channels, you can subscribe to locations other than the main PEAR repository and still have dependencies managed for you.

The following commands allow you to discover, add, list the contents of, and remove channels using the `pear` command.

*Channel commands*

| Command | Description |
|---|---|
| list-channel | Lists the available channels that you already have in your subscription list. |
| channel-add | Adds a channel from a channel server to your list. |
| channel-delete | Deletes a channel from your list of channels. |
| channel-discover | Discovers (and adds) any channel found at the given URL to your list of channels. |
| channel-info | Displays information about the given channel. |

All the packages covered in this book are from the default PEAR channel at `pear.php.net`.

# Installing the Packages in this Book

After this Introduction, each section's heading is the full name of the package documented in the section. To install the packages, just use PEAR. So, for the Auth package, which is the first package documented in this book after the Introduction, just type:

```
$ pear install auth
```

The case of the package is not important.

Some of the packages in this book were in alpha or beta when this book was written, and still might be. We've avoided calling them out because the package status could change quickly, thus making the book out of date. If a package is in alpha or beta, you'll get an error if you

attempt to install the package with the `pear` command with the default configuration of PEAR. If the package was called PACKAGENAME, you would get the error shown here:

```
Failed to download pear/PACKAGENAME within preferred state "stable", latest ➥
release is version 0.5.0, stability "beta", use ➥
"channel://pear.php.net/PACKAGENAME-0.5.0" to install
Cannot initialize 'PACKAGENAME', invalid or missing package file
Package "PACKAGENAME" is not valid
install failed
```

We suggest that, rather than modify your configuration to install alpha or beta packages by default, explicitly install alpha or beta packages if prompted to do so. You can install the package simply by appending `-alpha` or `-beta` to the package name, like this:

```
$ pear install PackageName-alpha
```

# PEAR_Error

It's useful when working with PEAR packages to be familiar with the `PEAR_Error` class, which is used for error handling. Many of the functions that you'll see throughout this book will return `PEAR_Error` if an error occurs in them. Others will throw `PEAR_Error`, as the `PEAR_ERROR_EXCEPTION` mode throws `Exception`.

## PEAR_Error()

The constructor for the `PEAR_Error` class.

```
PEAR_Error PEAR_Error([string $message = 'unknown error'] [, integer $code = null]
                      [, integer $mode = null] [, mixed $options = null]
                      [, string $userinfo = null])
```

| Parameters | Type | Description |
|---|---|---|
| $message | string | The error message. |
| $code | integer | An error code. |
| $mode | integer | The mode to use to process the error. See the "Error Modes" table for more information about available modes. |
| $options | mixed | The options to use for the error. |
| $userinfo | string | Additional information about the error. |

## addUserInfo()

Adds more information about the error.

```
void addUserInfo($info)
```

| Parameters | Type | Description |
|---|---|---|
| mixed | $info | Additional information about the error. |

## getBacktrace()

Returns the call trace where the error occurred.

```
array getBacktrace([integer $frame = null])
```

| Parameters | Type | Description |
|---|---|---|
| $frame | integer | The index of the frame to fetch. This is the index of the array returned by the function `debug_backtrace()`, which is used to populate the `backtrace` variable. |

## getCallback()

Returns the callback, if any.

```
mixed getCallback()
```

## getCode()

Returns the error code for the current `PEAR_Error`.

```
integer getCode()
```

## getDebugInfo()

Returns the same result as `getUserInfo()`.

```
string getDebugInfo()
```

## getMessage()

Returns the message associated with the current `PEAR_Error`.

```
string getMessage()
```

## getMode()

Returns the mode of the current `PEAR_Error`. See the "Error Modes" table for more information about the available modes.

```
integer getMode()
```

## getType()

Returns the name of the exception as a string.

```
string getType()
```

## getUserInfo()

Returns the user information that was added with `addUserInfo()` or set in the constructor.

```
string getUserInfo()
```

## toString()

Prints the `PEAR_Error` as a string.

```
string toString()
```

*Error modes*

| Mode | Description |
|---|---|
| PEAR_ERROR_CALLBACK | Tells `PEAR_Error` to call the function provided in the options when an error occurs. |
| PEAR_ERROR_DIE | Dies, or quits, when an error occurs. |
| PEAR_ERROR_EXCEPTION | An PHP 5 exception is created when an error occurs. |
| PEAR_ERROR_PRINT | The error message is printed when an error occurs. |
| PEAR_ERROR_RETURN | The error just returns and does no more action. |
| PEAR_ERROR_TRIGGER | Calls the `trigger_error()` method, which creates a user-level error message. |

## Creating a Simple PEAR_Error Object

This example demonstrates how to create an instance of `PEAR_Error` and how it looks when printed as `toString()`.

```
<?php

require_once 'PEAR.php';

$error = new PEAR_Error('An error has occurred!', 100, PEAR_ERROR_RETURN);

print $error->toString() . "\n";

?>
```

When you run the script, it will print out the following message:

```
[pear_error: message="An error has occurred!" code=100 mode=return level=notice ➥
prefix="" info=""]
```

## Using a Callback with PEAR_Error

This example shows how to use a callback with `PEAR_Error`. The name of the function is passed in as an option.

```
<?php

require_once 'PEAR.php';

function printError() {
    print "Running printError now!\n";
}

$error = new PEAR_Error('An error has occurred!', 3000, PEAR_ERROR_CALLBACK, ➥
'printError');

?>
```

When the script runs, the output looks like this:

```
Running printError now!
```

## Dying with an Error

This example shows how to use the `PEAR_ERROR_DIE` mode to exit the script when the error occurs. The script won't get to the last line because it will exit when the `PEAR_Error` object is created.

```
<?php

require_once 'PEAR.php';

$error = new PEAR_Error('An error has occurred!', 3000,
    PEAR_ERROR_DIE, "Additional information\n");

print "You won't get here!\n";

?>
```

The output looks like this:

```
Additional information
```

# Looking Ahead

Armed with the commands that you'll be using to manage packages, now it's time to start installing and using some of the PEAR packages. In the following section, you'll see packages that are used for authentication.