

## Chapter 2

# Installing PEAR and Smarty

PEAR and Smarty are both Open Source and readily available for download via the Internet. PEAR is licensed under the PHP License, while Smarty is licensed under the LGPL License. Both can be used and distributed in proprietary applications.

## Installing PEAR

PEAR should have been installed when your systems administrator installed PHP. Many Linux distributions now package PEAR with their PHP installations as well. In the rare case the PEAR was not installed when PHP was installed you will have to download it via the archives, which are available on PEAR's website.

## Installing PEAR Packages

PEAR comes complete with its very own package management utility aptly named pear. In the following examples I will outline the command line program, however, it should be noted that a GTK and web based version are also available on PEAR's website.

The pear utility allows you to list installed packages, list all available packages available via PEAR's website, update installed packages as well as install packages. For those of you who are familiar with Debian GNU/Linux's apt-get utility you will feel right at home.

### ***Listing All Available Packages***

The pear utility actually pings PEAR's website each time you request to see all available packages and then returns a nice list of package names with their corresponding version numbers. To view the current PEAR offerings type the following command.

Note: You will need root privileges to perform most if not all of the options the pear utility offers. If you do not have root access to the server you wish to install PEAR packages on you may have to contact your systems administrator

```
localhost# pear list-all
```

You should see a large listing of packages scroll by your window. At the time of this writing there was 130 packages listed as available. Packages that are already installed on your system will have the local version number listed to the right of the current version number as well.

## ***Installing a PEAR Package***

Installing a PEAR package is quite simple. First locate a package that sounds interesting, such as XML\_RSS. Once you have located a package that you think you might be interested in using I would recommend you check out the packages info.

```
localhost# pear remote-info XML_RSS
Package details:
=====
Latest    0.9.2
Installed - no -
Package   XML_RSS
License   PHP License
Category  XML
Summary   RSS parser
Description Parser for Resource Description Framework (RDF)
           Site Summary (RSS)
           documents.
```

The pear utility will query PEAR's website for the latest info about that package and return the desired information. Quickly we can tell what license the package has been released under, whether it has been installed, its category and a brief description. However, before installing our package we will need to check for dependencies and install any dependencies we need. At the time of this writing PEAR's dependency checking via the pear utilities package-dependency option was not working, however, when attempting to install a package it would error out with the required dependencies listed on the screen.

```
localhost# pear install
downloading XML_RSS-0.9.2.tgz ...
...done: 3,515 bytes
requires package `XML_Tree'
XML_RSS: Dependencies failed

localhost# pear install XML_Tree
downloading XML_Tree-2.0b1.tgz ...
...done: 7,360 bytes
install ok: XML_Tree 2.0b1
```

```
localhost# pear install XML_RSS
downloading XML_RSS-0.9.2.tgz ...
...done: 3,515 bytes
install ok: XML_RSS 0.9.2
```

Thanks to the pear utility installing PEAR packages is extremely easy. The pear utility also makes it easy to keep up-to-date with the most recent versions of PEAR packages you have already installed on your system.

## ***Updating Your PEAR Packages***

Imagine that you have found out about a new release of a PEAR package you use often, such as the PEAR and PEAR\_Error classes, which offers a wide range of new features. You have also noticed that a few of your other packages are aging and need to be updated. Have no fear! The pear utility offers the upgrade-all options, which compiles a list of locally installed packages and then searches for new versions on PEAR's website.

```
localhost# pear upgrade-all
Will upgrade mail
Will upgrade pear
downloading Mail-1.1.2.tgz ...
...done: 13,156 bytes
upgrade-all ok: Mail 1.1.2
downloading PEAR-1.2.1.tgz ...
...done: 83,126 bytes
upgrade-all ok: PEAR 1.2.1
```

The update-all utility is extremely useful if you use PEAR extensively and maintain applications which utilize more than a handful of PEAR packages.

Note: The pear utility offers many other options that have not been fully covered here, but you remain free to explore. To see a full listing of all of the pear utility's options simply type pear with no arguments from your command line.

## **Installing Smarty**

Smarty is not currently installed when PHP is installed, which means you will have to install it manually. For this task you will need a utility with the ability to uncompress TAR and Gzip archives.

First you will want to grab the latest release of Smarty from <http://smarty.php.net> and then change into your source directory. After you have downloaded the source you will

want to uncompress it. Before going any further you might want to read over the documentation included to make sure nothing has changed since this book was published.

## **Requirements**

At the time of this publication Smarty only required a webserver that was able of running PHP 4.0.6 or greater.

## **Basic Installation**

In the Smarty distribution you will notice a directory called libs. In the libs directory you will find the core Smarty library files. Copy these to a place on your webserver where you store your main PHP include files under the directory name of Smarty.

```
localhost# cp -R ./libs /usr/local/share/Smarty
```

Smarty has a special variable called SMARTY\_DIR that it uses to include plugins, modifiers and other helper classes. You should set this variable to the directory which you copied the Smarty library files to, possibly in a global configuration file.

Smarty expects that you will have four directories: config, cache, templates\_c and templates. The directory config holds configuration files, which are covered later in this book. The directory cache stores cached versions of each template if you enable caching. The directory templates\_c is the directory where Smarty stores the compiled versions of your templates. Finally, the directory templates is where your actual template files are kept.

## **Permissions**

Because Smarty actually writes and manipulates files on the filesystem it is important to remember that certain directories will have to be writable by the web server. At a bare minimum you will have to give the ability to write files to your web server on the templates\_c directory, which is where Smarty writes the compiled versions of your templates. If you choose to enable template caching you will also need to give write permissions to your web server for the cache directory as well.

## **Extended Installation**

By default Smarty assumes your template directories are in the directory in which it is installed. For a number of reasons you may wish to change the default template directories. To accomplish this I recommend creating a simple wrapper class around the default Smarty class that defines where your templates are.

Once you have created a wrapper class you will then use the wrapper class instead of the Smarty class in your applications. Since the wrapper class extends from the Smarty class it should function in exactly the same way the default Smarty class does, with the exception that you can quickly change the template directories. Below is an example of a simple wrapper class.

```
<?php

// Define the SMARTY_DIR variable with the absolute path of your
// Smarty installation
define('SMARTY_DIR','/path/to/Smarty');

// Include the Smarty class
require_once(SMARTY_DIR.'/Smarty.class.php');

/**
 * Template
 *
 * A simple wrapper class for Smarty
 *
 * @author Joe Stump <joe@joestump.net>
 * @package MyProgram
 */
class Template extends Smarty
{
    /**
     * Template
     *
     * Template constructor. Pass this function a the base directory, which
     * holds your template directories and it will create an instance of Smarty
     * with the correct directory structure.
     *
     * @author Joe Stump <joe@joestump.net>
     * @access public
     * @param string $baseDir
     */
    function Template($baseDir)
    {
        if(is_dir($baseDir))
        {
            $this->Smarty();
            $this->template_dir = $baseDir.'/templates';
            $this->compile_dir = $baseDir.'/templates_c';
            $this->config_dir = $baseDir.'/config';
        }
    }
}
```

```

        $this->cache_dir = $baseDir.'/cache';
    }
    else
    {
        $this = new PEAR_Error("Invalid baseDir: ".$baseDir);
    }
}
}
}

?>

```

The example outlines a simple wrapper class that takes \$baseDir as an argument in its constructor and then sets up your Smarty template. The variable \$baseDir is the directory which contains all of your template directories. Below is an example of how to use our newly created wrapper class.

```

<?php

// Include our newly created Template class
require_once('Template.php');

// Replace the following path with the correct path
$template = new Template('/path/to/templates');
if(!PEAR::isError($template))
{
    $template->assign('name','Joe Stump');
    $template->display('foo.tpl');
}
else
{
    // If $template is a PEAR_Error then display the error message
    die($template->getMessage());
}

?>

```

The above example illustrates how our newly created Template class behaves just as the default Smarty class does. There is one exception, which is the introduction of the PEAR\_Error class. If the \$baseDir passed to Template's constructor is not a directory then the instance is turned into a PEAR\_Error class with the appropriate error message included. In the next chapter we will be covering the PEAR\_Error class in more detail and how you can use it to improve error handling and cut down on debugging time.