

Doing that thing called PEAR: Packaging Source Code for PEAR Distribution

All right then! You [read the last blog entry advocating PEAR](#) (or found it during the week 

) and you want to know all about packaging code so your users can install your library or application using the PEAR installer. A few things first:

1. You don't need to propose a package to PEAR to do this.
2. You don't need to create a PEAR "channel" although it's very much recommended (another future blog post no doubt).
3. You don't need a whole lot of effort.

Taking your source code, and generating a PEAR package is a relatively simple task.

A PEAR package is a gzip tarball using the .tgz extension. It contains the source code to be installed, and a package definition file, package.xml. The package.xml file is how the PEAR installer knows which files to copy where, among other things like package details, installation tasks and post-installation scripts. I'd post a simple package.xml file to illustrate but that would be a bit dull (horrible XML!) and we're not going to edit any XML anyway (who does anymore?). Anyway, inside the tarball will be the source code inside a directory reflecting your library/app name and a package.xml definition.

This all sounds fun already!

So let's get dug in. Since editing XML is a dirty business, we're going to let PEAR do all the hard work. All we really need to is setup a PHP script to generate the package.xml dynamically. This is made easier with the availability of the PEAR_PackageFileManager (which includes version 2 of the manager for the more modern v2.0 package.xml files). You can install this manager using:

```
pear install PEAR_PackageFileManager
```

I'm going to assume you've installed PEAR. If you have it, and it's an old copy run this first ;):

```
pear upgrade PEAR
```

Now, we'll use the Manager the script a PHP file to generate our package.xml. At the outset this will look complex, but it's actually not difficult. Most of the stuff should be pretty obvious. Follow the comments for more information and save the file in the root directory of your source code. We'll assume a directory layout (in cvs/svn) of:

```
trunk
|- tests
|- docs
|- examples
|- MyCoolApp
|- library files/classes under top-level namespace
|- MyCoolApp.php
|- generate_package_xml.php
```

If your source code does not match this exactly, you can amend the `dir_roles` option below,

File: `generate_package_xml.php`

```
<?php
require_once('PEAR/PackageFileManager2.php');
PEAR::setErrorHandler(PEAR_ERROR_DIE);
$options = array(
    'filelistgenerator' => 'cvs',          // this copy of our source code is a CVS checkout
    'simpleoutput'      => true,
    'baseinstalldir'  => '/',            // The PEAR directory install location
    'packagedirectory' => dirname(<u>__FILE__</u>), // We've put this file in the root source code dir
    'clearcontents'   => true,          // dump any old package.xml content (set false to append release)
                                        // no bundling of cvs/svn files or this generator file

    'ignore'          => array('generate_package_xml.php', '.svn', '.cvs*'),
    'dir_roles'       => array(        // set up roles for some directories; the default is php
        'docs'        => 'doc',
        'examples'    => 'doc',
        'tests'       => 'test',
    ),
);
// Oddly enough, this is a PHP source code package...
$packagexml = &PEAR_PackageFileManager2::importOptions($packagefile, $options);
$packagexml->setPackageType('php');
// Package name, summary and longer description
$packagexml->setPackage('MyCoolLibrary');
$packagexml->setSummary('MyCoolLibrary does x, y and z');
$packagexml->setDescription("MyCoolLibrary does x, y and z. It follows Specification W 1.0.");
// The channel where this package is hosted. Since we're installing from a local
// downloaded file rather than a channel we'll pretend it's from PEAR.
$packagexml->setChannel('pear.php.net');
// Add some release notes!
$notes = <<<EOT
- Fixed bugs 1, 3, 6, 9
- Added documentation
- Implemented directory and file filtering with SPL FilterIterator
EOT;
$packagexml->setNotes($notes);
// Add any known dependencies such as PHP version, extensions, PEAR installer
$packagexml->setPhpDep('5.1.4');
$packagexml->setPearinstallerDep('1.4.0');
$packagexml->addPackageDepWithChannel('required', 'PEAR', 'pear.php.net', '1.4.0');
// Other info, like the Lead Developers. license, version details and stability type
$packagexml->addMaintainer('lead', 'padraic', 'Pádraic Brady', 'padraic@example.net');
$packagexml->setLicense('New BSD License', 'http://opensource.org/licenses/bsd-license.php');
$packagexml->setAPIVersion('0.0.1a');
$packagexml->setReleaseVersion('0.0.1a');
$packagexml->setReleaseStability('alpha');
$packagexml->setAPIStability('alpha');
// Add this as a release, and generate XML content
$packagexml->addRelease();
$packagexml->generateContents();
// Pass a "make" flag from the command line or browser address to actually write
// package.xml to disk, otherwise just debug it for any errors
```

```
if (isset($_GET['make']) || (isset($_SERVER['argv']) && @$_SERVER['argv'][1] == 'make')) {
    $packagexml->writePackageFile();
} else {
    $packagexml->debugPackageFile();
}
```

Ye gods! This stuff is just too complicated! Yes, it's really that simple 

. There is other stuff you could add depending on your library/app. Maybe you need to install command scripts, or run specific post-installation PHP scripts? All possible with a few extra easy lines. But this is a simple form with no such complications. We'll do a more complicated one another time.

Now comes the really hard part. Visit this file in a browser adding "?make=1" to the URL, or with the command line:

```
php generate_package_xml.php make
```

If you recheck the directory (the output will give it away) you'll see a new package.xml file. You can open it up to see why hand editing XML is such a bore. The PEAR_PackageFileManager2 is a massive time saver, and adds automation (secret ingredient to avoiding complex repetitive tasks).

The next step is generating the PEAR installable package. This requires three steps:

1. Validate package.xml
2. Generate the package tarball
3. Test install the package


Or, from the command line:

```
pear package-validate package.xml
pear package
pear install --force MyCoolLibrary-1.0.0a.tgz [use --force if similar version is already installed]
```

If you can follow these simple steps, you would be able to generate a PEAR package in seconds. Then the whole world can download your package file, and run "pear install" on it.


In my next visit to the world of PEAR, I'll add a little more automation (yes, more) using PHP's answer to Java Ant or the GNU's infamous [make](#) - [Phing](#).

Posted by Pádraic Brady in PHP General, PHP Security at 05:58

Why would someone go to all of that trouble when they could just use Phing and my d51pearpkg2 task? 

Anonymous on Oct 26 2007,

05:41


Because I haven't blogged it yet? 

Anonymous on Oct 26 2007, 06:02

Thanks a lot for this post and the package. It will be very usefull not only for standalone pear packages but also for building symfony plugins which are packaged the same way. Anonymous on Oct 27 2007, 18:05

What about PPFM_CLI?

http://pear.php.net/package/PEAR_PackageFileManager_Cli Anonymous on Oct 28 2007, 03:48

I haven't used the CLI package because I integrate with Phing - so there's no human interaction with the build process. Sometimes there not even a Human, just cron 

. Anonymous on Oct 28 2007, 06:27

thanks for the GREAT post! Very useful... Anonymous on Nov 22 2007, 00:09