

Sunday, August 20, 2006

Creating PEAR-installable nightly builds

If you incorporate a "release early, release often" policy for your projects, it helps you to detect bugs in an early development stage of your application. In many cases making a release takes some time and you do not release new versions as often as you intended. We had these problems with our projects over at www.php-tools.net. That's why we created snaps.php-tools.net, a site where you can download nightly builds of our projects as ZIP, TAR.GZ or TAR.BZ2 archive. This saves the users, who want to test the latest development versions the hassle checking out the latest version from our subversion repository. The problem of this technique was, that most users use the PEAR installer to deploy our packages and if they downloaded a ZIP file, it's not possible to use the PEAR installer on this file. That's why we came up with a solution to create nightly builds that are installable with the PEAR installer. Read on, if you are interested in how this can be achieved...

To create a PEAR installable package, all you need is a package.xml file, that contains all information about the project and the related files. So to create a nightly build, you will need a current package.xml that describes the package. Creating this by hand can be very cumbersome, as it has to contain all dependencies as well as all files that you want to include in the resulting package. But help has arrived, as Greg Beaver provides a PEAR package that aids you in the creation of package.xml files: PEAR_PackageFileManager. This package provides a PHP-API to read, modify and write the package.xml file needed to create an installable package. To automate this process, you need to follow these steps: Export the latest version from Subversion/CVSDetermine a version number to identify the nightly buildGenerate a package.xml fileUse the pear package command to create the archiveThe first task at hand is extremely easy, as Subversion and CVS both offer the export command. To get the latest version of patTemplate, the following command does the trick:

```
$ svn export http://www.php-tools.net/svn/patTemplate/trunk patTemplate
```

Now that you have exported the latest code-base the next steps is to create a unique version number. We decided that we wanted to use the latest officially released version (e.g. 3.1.0) and add a version suffix. This suffix could either be the latest SVN revision, or the date that the last change has been made.

As the revision in Subversion is incremented for the complete repository, once a commit has been made and all our projects share a common repository, we decided to use the date of the last change in the project folder as the version suffix. Once you have checked out the contents, this can be easily accomplished using the SPL's RecursiveDirectoryIterator:

```
function getLastModifiedDate($folder) {
    $modified = 0;
    $dir = new RecursiveDirectoryIterator($folder, RecursiveIteratorIterator::LEAVES_ONLY);

    foreach (new RecursiveIteratorIterator($dir) as $file) {
        if ($file->getMTime() > $modified) {
            $modified = $file->getMTime();
        }
    }
    return $modified;
}
```

This information now has to be used in a script, that will generate the package.xml file. For snaps.php-tools.net, we decided to split these tasks. The SVN export and the generation of the version suffix is handled by a simple script `createSnaps.php` which also build the plain ZIP or TAR archives.

After exporting the latest SVN version, this script searches for a file called `autopackage2.php`. If the file is available, it will execute the file and pass the version suffix as the sole parameter.
`$createPackageFile = "php autopackage2.php " . date('YmdHi', $mtime);`
`exec($createPackageFile);`
The `autopackage2.php` file then will use PEAR_PackageFileManager to create the current package.xml file. I will not go into detail here, you may take a look at an example in the `patTemplate` subversion repository. The most important part is, how the version number is created:
`// Base version`
`$baseVersion = '3.1.0';`

```
// current version
$version = $baseVersion . 'dev' . $argv[1];
```

Furthermore it uses a `filelistgenerator` provided by PEAR_PackageFileManager to include all files that have been exported from SVN. This `autopackage2.php` script will generate a package.xml file and save it on disk. To create a PEAR installable package from this newly generated

Blog Export: a programmer's best friend, <http://blog.php-tools.net/>

package.xml file, all that's left to do is execute the following command:
`$ pear package -n`
The `-n` switch is used to echo the full path to the generated `.tgz` package file after it has been generated. We use it to move this file to the document root, so it can be downloaded. And voilà you created a PEAR-installable nightly build.

For `snaps.php-tools.net` we also implemented some more features: It is possible to not only create a version from the current trunk, but as many branches as you like. We also create ZIP, TAR.GZ and TAR.BZ2 archives for users who do not use the PEAR installer. We implemented a simple class that acts as a webfrontend to download the nightly builds. If you are interested in using the same technique for your nightly builds, feel free to use the code of `snaps.php-tools.net` as a starting point. You can get it from our subversion repository:
`$ svn co`
`http://www.php-tools.net/svn/patSnaps/trunk`
Beware that some of the code is several years old and while it works with PHP 5.1 it surely could be optimized. Examples for the `autopackage2.php` file are available in the subversion repositories for `patTemplate`, `patForms` and `patBBCode`.

Posted by schst in PHP, PAT at 11:23