Creating ZIP and TAR archives on the fly with PHP

By Dennis Pallett

Abstract

In this tutorial you will learn how to create ZIP and TAR archives dynamically using PHP and the PEAR::Archive_Zip and PEAR::Archive_Tar packages.

Introduction

The easiest way to offer files for downloading on your website is to simply create the necessary archives (using WinZip or another program) on your own computer, and uploading the archives. But in some cases this isn't enough, for instance when you want to generate a unique archive for each person who downloads the archive. This means you'll have to create ZIP and TAR archives on the fly with PHP.

In this tutorial I will show you exactly how to do that. Thankfully there are two excellent libraries in the PHP Extension and Application Repository (PEAR) which makes it a lot easier since all the hard stuff has been written for us already.

You will also learn how to stream these dynamically created archives using the right headers so that the browser will know it's an archive, and not a normal PHP page.

Let's start with the most important: creating the archives.

Creating the archives

To create the archives we'll be using the PEAR::Archive_Zip package and the PEAR::Archive_Tar package. Since they're PEAR packages, you'll also have to download the main PEAR.php file. Either you can download and install the full PEAR library, or you can download the PEAR.php file by clicking here. Since you don't need anything except the PEAR.php file I recommend downloading it from the link I just gave you, instead of installing the full PEAR library.

Let's start of with the PEAR::Archive_Zip package. It's surprisingly easy to use, and it only takes three lines to create a new zip file:

```
<?php
 include ('pear/archive_zip.php');

// Create instance of Archive_Zip class, and pass the name of our zipfile
 $zipfile = New Archive_Zip('myzipfile.zip');
```

```
// Create a list of files and directories
 $list = array('example.txt');

// Create the zip file
 $zipfile->create($list);

echo 'Zip file created';

?>
```

Let's go through this example. First we include the PEAR::Archive_Zip package, after which we create an instance of the Archive_Zip class and pass the name of the zipfile we are going to create as the first parameter.

Then we build a list of all the files and directories that should be included in the zipfile. In the example we include only one file called example.txt which is located in the same directory as the script. This list of files is passed to the create() method of the object, and the actual zip file is created, and will be located in the same path as your script.

All seems very simple, but there are a few catches. For one, if the path isn't writable it won't be possible to create the zip file, and in many cases, the script path isn't writable, so we'll have to think of a solution.

Using the temporary directory

To workaround this problem we can use the system's temporary directory, which is always writable. To get the path of the temporary directory you can use the $_ENV variable, like so:

```
<?php
 // Get temporary directory
 if (!empty($_ENV['TMP'])) {
        $tempdir = $_ENV['TMP'];
 } elseif (!empty($_ENV['TMPDIR'])) {
        $tempdir = $_ENV['TMPDIR'];
 } elseif (!empty($_ENV['TEMP'])) {
        $tempdir = $_ENV['TEMP'];
 } else {
        $tempdir = dirname(tempnam('', 'na'));
 }

if (empty($tempdir)) { die ('No temporary directory'); }

// Make sure trailing slash is there
 $tempdir = rtrim($tempdir, '/');
 $tempdir .= '/';

// Make sure temporary directory is writable
 if (is_writable($tempdir) == false) {
        die ('Temporary directory isn\'t writable');
 }
 ?>
```

As you can see in the above example there are several different variables that can contain the path of the

temporary directory and each is checked. We also make sure that the temporary directory ends with a slash, and that it's writable.

Now that we've got the temporary directory, we have to create a directory where we can safely store the zip file temporarily without any interference, so let's do that:

```
<?php
 // Create temp name     for our own directory
 $dir = tempnam($tempdir, 'temp');

// Make sure another file or directory doesn't already exist with this name
 @unlink($dir);
 @rmdir($dir);

// Create directory
 mkdir($dir);
 $dir .= '/';
 ?>
```

In the above example the tempnam() function is used to create a temporary name for our directory. After that we first make sure the name isn't used by anything else, and then proceed to create the directory.

Now that we've got a temporary directory, let's make sure our zip file gets created there, by changing the current working directory, with the chdir() function, like so:

```
<?php
 // Change current working directory, so that the zip file gets created in the temp dir
 chdir($dir);
 ?>
```

The full code now looks like this:

```
<?php
 include ('pear/archive_zip.php');

// Create instance of Archive_Zip class, and pass the name of our zipfile
 $zipfile = New Archive_Zip('myzipfile.zip');

// Create a list of files and directories
 $list = array('example.txt');

// Get temporary directory
 // Get temporary directory
 if (!empty($_ENV['TMP'])) {
        $tempdir = $_ENV['TMP'];
 } elseif (!empty($_ENV['TMPDIR'])) {
        $tempdir = $_ENV['TMPDIR'];
 } elseif (!empty($_ENV['TEMP'])) {
        $tempdir = $_ENV['TEMP'];
 } else {
        $tempdir = dirname(tempnam('', 'na'));
 }
```

```
if (empty($tempdir)) { die ('No temporary directory'); }

// Make sure trailing slash is there
 $tempdir = rtrim($tempdir, '/');
 $tempdir .= '/';

// Make sure temporary directory is writable
 if (is_writable($tempdir) == false) {
        die ('Temporary directory isn\'t writable');
 }

// Create temp name     for our own directory
 $dir = tempnam($tempdir, 'temp');

// Make sure another file or directory doesn't already exist with this name
 @unlink($dir);
 @rmdir($dir);

// Create directory
 mkdir($dir);
 $dir .= '/';

// Change current working directory, so that the zip file gets created in the temp dir
 chdir($dir);

// Create the zip file
 $zipfile->create($list);

echo 'Zip file created in ' . $dir;

?>
```

There's only one problem now. Because we changed the working directory, the relative paths won't work. The obvious solution that comes to mind is to use absolute paths, but that doesn't work properly either, due to a bug in the Archive_Zip package; it doesn't work with absolute paths.

Fixing the relative paths

So we have to find a solution for that. The easiest solution is to simply copy all the files and directory to our temporary directory, so the relative paths still work. Copying a file is easy, since PHP comes with an inbuilt copy function logically called copy(). We have to add the following to our script:

```
// Copy files & directories so relative paths still work
 foreach ($list as $item) {
        if (is_file($item)) {
                // Copy file
                copy ($item, $dir . $item);
        } elseif (is_dir ($item)) {
                // TODO: we'll do this in a minute
        } else {
                // Invalid file, just ignore
                continue;
```

```
        }
}
```

If the above code is placed above the chdir() call, all the files will be included in the zip file, and work properly. But directories aren't yet included because we haven't copied those yet.

Unfortunately PHP doesn't come with a copy_directory() or something similar, and we have to write our own. A quick Google search will probably turn up something very quickly, but here's one I wrote myself:

```
function move_directory($source, $dest, $copy=false) {
        // Standardize paths
        $source = str_replace('\\\\', '/', $source);
        $source = str_replace('\\', '/', $source);

       $dest = str_replace('\\\\', '/', $dest);
        $dest = str_replace('\\', '/', $dest);

       // Remove trailing slashes
        $source = rtrim($source, '/');
        $dest = rtrim($dest, '/');

       // Add trailing slashes
        $source .= '/';
        $dest .= '/';

       // Source doesn't exist?
        if (file_exists($source) == false) { return false; }

       // Try to create destination
        @mkdir($dest);

       // Destination doesn't exist?
        if (file_exists($dest) == false) { return false; }

       // Copy all files
        $d = dir($source);

       while (false !== ($entry = $d->read())) {
                if ($entry == '.' OR $entry == '..') continue;
                $file = $d->path . $entry;

               // Is a sub dir?
                if (is_dir($file)) {
                        // Try to create sub dir
                        $subdir = $dest . $entry . '/';
                        $result = @mkdir($subdir);
                        if ($result == false) { continue; }

                       // Move files in sub directory
                        move_directory($file, $subdir, $copy);
                        continue;
                }
```

```
                // Copy file
                copy($file, $dest . $entry);

                // Remove old file?
                if ($copy == false) {
                        // Moving the directory, so remove original
                        @unlink($file);
                }
        }

        if ($copy == false) {
                @rmdir($source);
        }

        return true;
}
```

I won't really bother explaining the function, but it basically copies all the files in the directory, and all the sub-directories (and does the same for all the sub-directories, and so forth).

If we incorporate this function into our script, the part that copies the files and directories now looks like this:

```
// Copy files & directories so relative paths still work
 foreach ($list as $item) {
        if (is_file($item)) {
                // Copy file
                copy ($item, $dir . $item);
        } elseif (is_dir ($item)) {
                // Copy directory
                move_directory ($item, $dir . $item, true);
        } else {
                // Invalid file, just ignore
                continue;
        }
}
```

And if we put everything together our script looks like this:

```
<?php
 include ('pear/archive_zip.php');

// Create instance of Archive_Zip class, and pass the name of our zipfile
 $zipfile = New Archive_Zip('myzipfile.zip');

// Create a list of files and directories
 $list = array('example.txt', 'test');

// Get temporary directory
 if (!empty($_ENV['TMP'])) {
        $tempdir = $_ENV['TMP'];
 } elseif (!empty($_ENV['TMPDIR'])) {
        $tempdir = $_ENV['TMPDIR'];
 } elseif (!empty($_ENV['TEMP'])) {
```

```
        $tempdir = $_ENV['TEMP'];
 } else {
        $tempdir = dirname(tempnam('', 'na'));
 }

if (empty($tempdir)) { die ('No temporary directory'); }

// Make sure trailing slash is there
 $tempdir = rtrim($tempdir, '/');
 $tempdir .= '/';

// Make sure temporary directory is writable
 if (is_writable($tempdir) == false) {
        die ('Temporary directory isn\'t writable');
 }

// Create temp name     for our own directory
 $dir = tempnam($tempdir, 'temp');

// Make sure another file or directory doesn't already exist with this name
 @unlink($dir);
 @rmdir($dir);

// Create directory
 mkdir($dir);
 $dir .= '/';

// Copy files & directories so relative paths still work
 foreach ($list as $item) {
        if (is_file($item)) {
                // Copy file
                copy ($item, $dir . $item);
        } elseif (is_dir ($item)) {
                // Copy directory
                move_directory ($item, $dir . $item, true);
        } else {
                // Invalid file, just ignore
                continue;
        }
 }

// Change current working directory, so that the zip file gets created in the temp dir
 chdir($dir);

// Create the zip file
 $zipfile->create($list);

echo 'Zip file created in ' . $dir;

function move_directory($source, $dest, $copy=false) {
        // Standardize paths
        $source = str_replace('\\\\', '/', $source);
        $source = str_replace('\\', '/', $source);
```

```
$dest = str_replace('\\\\', '/', $dest);
 $dest = str_replace('\\', '/', $dest);

// Remove trailing slashes
 $source = rtrim($source, '/');
 $dest = rtrim($dest, '/');

// Add trailing slashes
 $source .= '/';
 $dest .= '/';

// Source doesn't exist?
 if (file_exists($source) == false) { return false; }

// Try to create destination
 @mkdir($dest);

// Destination doesn't exist?
 if (file_exists($dest) == false) { return false; }

// Copy all files
 $d = dir($source);

while (false !== ($entry = $d->read())) {
        if ($entry == '.' OR $entry == '..') continue;
        $file = $d->path . $entry;

       // Is a sub dir?
        if (is_dir($file)) {
                // Try to create sub dir
                $subdir = $dest . $entry . '/';
                $result = @mkdir($subdir);
                if ($result == false) { continue; }

               // Move files in sub directory
                move_directory($file, $subdir, $copy);
                continue;
        }

       // Copy file
        copy($file, $dest . $entry);

       // Remove old file?
        if ($copy == false) {
                // Moving the directory, so remove original
                @unlink($file);
        }
 }

if ($copy == false) {
        @rmdir($source);
 }
```

```
        return true;
 }

?>
```

And that's pretty much all we need to do.

Creating a re-usable function

To make it even easier to create zip files, let's change our script into a re-usable function:

```
<?php
 function create_archive ($name, $list) {
        // Get temporary directory
        if (!empty($_ENV['TMP'])) {
                $tempdir = $_ENV['TMP'];
        } elseif (!empty($_ENV['TMPDIR'])) {
                $tempdir = $_ENV['TMPDIR'];
        } elseif (!empty($_ENV['TEMP'])) {
                $tempdir = $_ENV['TEMP'];
        } else {
                $tempdir = dirname(tempnam('', 'na'));
        }

       if (empty($tempdir)) { die ('No temporary directory'); }

       // Make sure trailing slash is there
        $tempdir = rtrim($tempdir, '/');
        $tempdir .= '/';

       // Make sure temporary directory is writable
        if (is_writable($tempdir) == false) {
                die ('Temporary directory isn\'t writable');
        }

       // Create temp name     for our own directory
        $dir = tempnam($tempdir, 'temp');

       // Make sure another file or directory doesn't already exist with this name
        @unlink($dir);
        @rmdir($dir);

       // Create directory
        mkdir($dir);
        $dir .= '/';

       // Copy files & directories so relative paths still work
        foreach ($list as $item) {
                if (is_file($item)) {
                        // Copy file
                        copy ($item, $dir . $item);
                } elseif (is_dir ($item)) {
```

```
                // Copy directory
                move_directory ($item, $dir . $item, true);
            } else {
                // Invalid file, just ignore
                continue;
            }
        }

        // Change current working directory, so that the zip file gets created in the temp dir
        chdir($dir);

        // Create instance of Archive_Zip class, and pass the name of our zipfile
        $zipfile = New Archive_Zip($name . '.zip');

        // Create the zip file
        $zipfile->create($list);

        // Return file data
        $data = implode('', file($name . '.zip'));
        return $data;
}
?>
```

The above function will create the zip file, and return the data, which can then be used to stream to the user (which I will show you later in this article). It's used like this:

```
<?php
// Create a list of files and directories
$list = array('example.txt', 'test');

// Create the archive
$data = create_archive ('myzipfile', $list);

echo 'Zip file created, size is ' . strlen($data) . ' bytes';
?>
```

Because we've now got a nice compact and re-usable function, it's easy to create TAR archives as well. The Archive_Tar package works almost the same, so we have to change very little.

First include the Achive_Tar package:

```
<?php
include ('pear/Tar.php');
?>
```

And then change the create_archive function slightly:

```
function create_archive ($name, $list, $type='zip') {
        // Get temporary directory
        if (!empty($_ENV['TMP'])) {
            $tempdir = $_ENV['TMP'];
        } elseif (!empty($_ENV['TMPDIR'])) {
            $tempdir = $_ENV['TMPDIR'];
```

```
 } elseif (!empty($_ENV['TEMP'])) {
        $tempdir = $_ENV['TEMP'];
 } else {
        $tempdir = dirname(tempnam('', 'na'));
 }

if (empty($tempdir)) { die ('No temporary directory'); }

// Make sure temporary directory is writable
 if (is_writable($tempdir) == false) {
        die ('Temporary directory isn\'t writable');
 }

// Create temp name     for our own directory
 $dir = tempnam($tempdir, 'temp');

// Make sure another file or directory doesn't already exist with this name
 @unlink($dir);
 @rmdir($dir);

// Create directory
 mkdir($dir);
 $dir .= '/';

// Copy files & directories so relative paths still work
 foreach ($list as $item) {
        if (is_file($item)) {
                // Copy file
                copy ($item, $dir . $item);
        } elseif (is_dir ($item)) {
                // Copy directory
                move_directory ($item, $dir . $item, true);
        } else {
                // Invalid file, just ignore
                continue;
        }
 }

// Change current working directory, so that the zip file gets created in the temp dir
 chdir($dir);

// Create zip or tar.gz file
 if ($type == 'zip') {
        $name .= '.zip';
        $zipfile = New Archive_Zip($name);
        $zipfile->create($list);
 } else {
        $name .= '.tar.gz';
        $tarfile = New Archive_Tar($name, 'gz');
        $tarfile->create($list);
 }

// Get file data
```

```
        $data = implode('', file($name));
        return $data;
}
```

As you can see we've added a third parameter to the function called $type to specify the type of archive. At the end of the function we've also made a few changes, making it possible to create a zip file or tar.gz file.

And that's all there is to it. It's now possible to dynamically create ZIP and TAR.GZ archives. Let's look at the final part: sending the archive to the visitor.

Streaming files

To stream a file to a user browser, and have it downloading instead of displaying the data, you have to send various headers using the header() function. This isn't necessary in our case, because there's a great function on PHPit that does this all automatically for us. The function, called force_download(), looks like this:

```
function force_download ($data, $name, $mimetype='', $filesize=false) {
    // File size not set?
    if ($filesize == false OR !is_numeric($filesize)) {
        $filesize = strlen($data);
    }

   // Mimetype not set?
    if (empty($mimetype)) {
        $mimetype = 'application/octet-stream';
    }

   // Make sure there's not anything else left
    ob_clean_all();

   // Start sending headers
    header("Pragma: public"); // required
    header("Expires: 0");
    header("Cache-Control: must-revalidate, post-check=0, pre-check=0");
    header("Cache-Control: private",false); // required for certain browsers
    header("Content-Transfer-Encoding: binary");
    header("Content-Type: " . $mimetype);
    header("Content-Length: " . $filesize);
    header("Content-Disposition: attachment; filename=\"" . $name . "\";" );

   // Send data
    echo $data;
    die();
}

function ob_clean_all () {
    $ob_active = ob_get_length () !== false;
    while($ob_active) {
        ob_end_clean();
        $ob_active = ob_get_length () !== false;
    }
```

```
    return true;
 }
```

The function requires three parameters: the data of the file, in our case the data of our archive which we get from the create_archive() function, the name of the file, which is in our case something like 'myfile.zip', and the mimetype. This last parameter is used to tell the browser what type of file we're sending. If we're sending a zip file, the mimetype should be "application/zip" and if we're sending a tar.gz file it should "application/x-tar-gz". Let's put together our download script:

```
<?php
 include ('pear/archive_zip.php');
 include ('pear/Tar.php');

// Name of the archive
 $name = (!empty($_GET['name'])) ? $name = $_GET['name'] : $name = 'script';

// Just some protection
 $name = str_replace("\\\\", '', $name);
 $name = str_replace("\\", '', $name);
 $name = str_replace('/', '', $name);

// Type specified?
 $type = (!empty($_GET['type'])) ? $type = $_GET['type'] : $type = 'zip';
 if ($type != 'zip' AND $type != 'tar') { $type = 'zip'; }

// Add this script to the list
 $list = array('download.php');

// Create the archive
 $data = create_archive ($name, $list, $type);

if ($type == 'zip') {
        $mimetype = 'application/zip';
        $name .= '.zip';
 } else {
        $mimetype = 'application/x-tar-gz';
        $name .= '.tar.gz';
 }

// Send file
 force_download ($data, $name, $mimetype);
 die();

// â€¦ all the functions go here
 ?>
```

That's all. Click here to download the full script and that's also a live demo of the above example. The archive you'll be downloading is generated dynamically.

Conclusion

In this tutorial I've shown you how to create ZIP and TAR archives on the fly using PHP and two PEAR

packages. Using these packages is really simple, but you have to workaround a few problems before you can really use them. The function I've given you in this tutorial works around most of these problems, and allows you to dynamically create archives in almost any situation.

There's still a problem with global paths, since these don't work, and the create_archive() function doesn't account for this. This is only a minor problem, and can easily be fixed. This is left as an exercise for the reader.

If you have any comments or questions on this tutorial, feel free to leave them in the comments below or join us at PHPit Forums to talk more about this tutorial and other PHP topics.

About this PDF