Zippy Documentation

Release 0.1

Alchemy

Feb 22, 2018

Contents

CHAPTER 1

Introduction

Zippy is an Object Oriented PHP library that aim to ease the use of the archive manipulation by providing a set of adapters that will use command line utilities or PHP extensions depending on the environment your run it

Zippy currently supports the following utilities :

- GNU TAR
- BSD TAR
- ZIP

And deals with the following archive formats :

- tar
- zip
- tbz2
- tbz
- tgz

CHAPTER 2

Installation

We rely on composer to use this library. If you do not still use composer for your project, you can start with this `composer.json` at the root of your project :

```
{
    "require": {
        "alchemy/zippy": " ~0.1"
    }
}
```

Install composer :

```
# Install composer
curl -s http://getcomposer.org/installer | php
# Upgrade your install
php composer.phar install
```

You now just have to autoload the library to use it :

```
<?php
require 'vendor/autoload.php';

use Zippy\Zippy;

$zippy = Zippy::load();
```

This is a very short intro to composer. If you ever experience an issue or want to know more about composer, you will find help on their web site composer.

CHAPTER 3

Basic Usage

The Zippy library is very simple and consists of a collection of adapters that take over for you the most common (de)compression operations (create, list update, extract, delete) for the chosen format.

Example usage

```php
<?php

use Alchemy\Zippy;

$zippy = Zippy::load();

// creates
$archiveZip = $zippy->create('archive.zip');

// updates
$archiveZip->addMembers(array(
        '/path/to/file',
        '/path/to/file2',
        '/path/to/dir'
    ),
    $recursive = false
);

// deletes
$archiveZip->removeMembers('/path/to/file2');

// lists
foreach ($archiveZip as $member) {
    if ($member->isDir()) {
        continue;
    }

    echo $member->getLocation(); // outputs /path/to/file
}
```

```
// extracts
$archiveZip->extract('/to/directory');
```

Zippy comes with a strategy pattern to get the best adapter according to the platform you use and the availability of the utilities.

The right adapter will be matched when you open or create a new archive.

Creates or opens one archive

```
<?php

use Alchemy\Zippy;

$zippy = Zippy::load();

$archiveZip = $zippy->create('archive.zip');
$archiveTar = $zippy->open('/an/existing/archive.tar');
```

However you may want sometimes gets the adapter for future reuse as the previous example is good for one shot only because it will create a new adapter object instance each time you create or open an archive.

Creates or opens a lot of archives

```
<?php

use Alchemy\Zippy;

$zippy = Zippy::load();

$zipAdapter = $zippy->getAdapterFor('zip');

foreach(array('archive.zip', 'archive2.zip', 'archive3.zip') as $path) {
    $archive = $zipAdapter->open($path);
}
```

Also sometimes you will face the problem where Zippy will not be able to handle a specific archive format because archive extension is not recognized or follow specific named rules.

Luckily with Zippy You can easily define your strategy to get a specific adapter that handle (de)compression for a specific archive format.

The discrimination factor for getting the right adapter is based upon the archive extension.

So every time you will work with an archive format not handled by Zippy you must declare a new strategy for this extension to match the proper adapter, see *Add custom utility strategy*.

CHAPTER 4

Recipes

4.1 Define custom binary path

Each binary utility comes with two binary path one for the inflator and the other for the deflator. By default if none is provided, zippy will look to find the executable by its name;

```
<?php

use Alchemy\Zippy;

$zippy = Zippy::load();

// customize GNU Tar inflator
$zippy->adapters['gnu-tar.inflator'] = '/usr/local/bin/tar';

// customize ZIP deflator
$zippy->adapters['zip.deflator'] = '/usr/local/bin/unzip';
```

The following binary are customisable

- gnu-tar.inflator
- gnu-tar.deflator
- bsd-tar.inflator
- bsd-tar.deflator
- zip.inflator
- zip.deflator

4.2 Add custom utility strategy

Zippy provides a way to define your custom strategy based on the file extension to get the most adapted adapters according to your needs.

Each adapters implements a *isSupported()* method which will be executed for the defined list of adapters. The first supported adapter will be chosen as the archive adapter.

Define your custom adapter

Your custom adapter class must implements the `Alchemy\Zippy\Adapter\AdapterInterface`.

```php
<?php

use Alchemy\Zippy;

class CustomAdapter implements Zippy\Adapter\AdapterInterface
{
    ...
}
```

Define a new strategy

Your custom strategy class must implements the `Alchemy\Zippy\Strategy\FileStrategy`.

```php
<?php

use Alchemy\Zippy;

class CustomStrategy implements Zippy\Strategy\FileStrategy
{
    public function getAdapters()
    {
        return array(CustomAdapter::newInstance());
    }
    public function getFileExtension()
    {
        return 'tar.custom';
    }
}
```

Add your custom strategy into zippy

```php
<?php

$zippy = Alchemy\Zippy::load();

// add your strategy
// This strategy for `tar.custom` files has priority over all previously
// registered strategies for this extension
$zippy->addStrategy(new CustomStrategy());

// use it
$archiveTarCustom = $zippy->create('archive.tar.custom');
```

CHAPTER 5

Handling Exceptions

Zippy throws different types of exception :

- **`\Alchemy\Zippy\Exception\NotSupportedException`** is thrown when current operation is not supported.
- `\Alchemy\Zippy\Exception\RunTimeException`
- `\Alchemy\Zippy\Exception\InvalidArgumentException`

All these Exception implements `\Alchemy\Zippy\Exception\ExceptionInterface` so you can catch any of these exceptions by catching this exception interface.

CHAPTER 6

Report a bug

If you experience an issue, please report it in our issue tracker. Before reporting an issue, please be sure that it is not already reported by browsing open issues.

CHAPTER 7

Contribute

You find a bug and resolved it ? You added a feature and want to share ? You found a typo in this doc and fixed it ? Feel free to send a Pull Request on GitHub, we will be glad to merge your code.

CHAPTER 8

Run tests

Zippy relies on PHPUnit for unit tests. To run tests on your system, ensure you have PHPUnit installed, and, at the root of Zippy execute it :

```
phpunit
```

CHAPTER 9

About

Zippy has been written by the Alchemy dev team for Phraseanet, our DAM software. Try it, it's awesome !

CHAPTER 10

License

Zippy is licensed under the MIT License