Composer: Getting Started

THE BASICS OF COMPOSER

Jonathan Klein
DIRECTOR OF ENGINEERING
@jonathanklein www.jonathanklein.net

Course Summary

The Basics of Composer

Including Third Party Code

Autoloading Your Code

Composer Scripts

Publishing Your Own Composer Package

Module Summary

Core Problems Composer Solves

Composer vs. PECL / Pear

Installing Composer

Your First Composer Project

Two Main Problems Composer Solves

Including 3rd Party Code

Autoloading Your Code

Two Main Problems Composer Solves

Including 3rd Party Code

Autoloading Your Code

Dependency Management

Introduction

Composer is a tool for dependency management in PHP. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.

Dependency management

Composer is **not** a package manager in the same sense as Yum or Apt are. Yes, it deals with "packages" or libraries, but it manages them on a per-project basis, installing them in a directory (e.g. `vendor`) inside your project. By default it does not install anything globally. Thus, it is a dependency manager. It does however support a "global" project for convenience via the global command.

This idea is not new and Composer is strongly inspired by node's npm and ruby's bundler.

Suppose:

1. You have a project that depends on a number of libraries.
2. Some of those libraries depend on other libraries.

Composer:

1. Enables you to declare the libraries you depend on.
2. Finds out which versions of which packages can and need to be installed, and installs them (meaning it downloads them into your project).

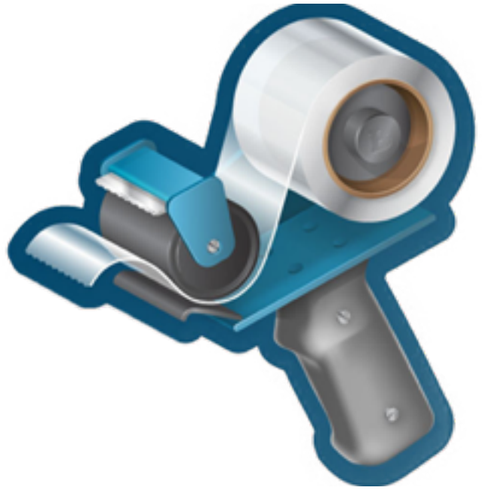
See the Basic usage chapter for more details on declaring dependencies.

Composer Compared To Other Software

PHP: Composer

Python: PIP

Node.js: NPM



Ruby: Bundler

Common Tasks in Applications

Logging

Routing

Unit Testing

Email

HTTP Requests

Composer Packages Solving These Problems

Logging:
monolog/monolog

Routing:
symfony/routing

Unit Testing:
phpunit/phpunit

Email:
swiftmailer/swiftmailer

HTTP Requests:
guzzlehttp/guzzle

Two Main Problems Composer Solves

Including 3rd Party Code

Autoloading Your Code

```
include('sqlfunctions.php');
include('editfunctions.php');
include('db_class.php');
include('utilities.php');
include('models/user.php');
include('models/account.php');
include('controllers/user.php');
include('controllers/account.php');
```

Loading Code Can Be a Pain

Composer makes this much easier

```
require __DIR__ . '/vendor/autoload.php';
```

With Composer You Only Need One Line

Isn’t that better?

```
{
    "autoload": {
        "psr-4": {"Acme\\": "src/"}
    }
}
```

Autoloading With Composer

```
{
    "autoload": {
        "psr-4": {"Acme\\": “src/“},
        "files": ["src/functions.php"]
    }
}
```

Autoloading With Composer

Composer vs. PECL and PEAR

PECL:

PHP Extensions

PEAR:

Global Dependencies

Composer:

Local First, Global Optionally

PEAR

It is possible to install packages from any PEAR channel by using the `pear` repository. Composer will prefix all package names with `pear-{channelName}/` to avoid conflicts. All packages are also aliased with prefix `pear-{channelAlias}/`

Example using `pear2.php.net`:

```
{
    "repositories": [
        {
            "type": "pear",
            "url": "https://pear2.php.net"
        }
    ],
    "require": {
        "pear-pear2.php.net/PEAR2_Text_Markdown": "*",
        "pear-pear2/PEAR2_HTTP_Request": "*"
    }
}
```

In this case the short name of the channel is `pear2`, so the `PEAR2_HTTP_Request` package name becomes `pear-pear2/PEAR2_HTTP_Request`.

> **Note:** The `pear` repository requires doing quite a few requests per package, so this may considerably slow down the installation process.

Demo

Installing Composer

Demo

Setting Up Your First Composer Project