How to use PHP libraries in legacy extensions

insight.helhum.io/post/148112375750/how-to-use-php-libraries-in-legacy-extensions

tl;dr: We would be better off fully embracing composer in the TYPO3 ecosystem. You should do so for your own projects.

If you are not interested in a little bit of history, you can safely skip the following chapter.

History

This is actually a pretty good question, but also a pretty old one. Interestingly this is nowadays asked much more frequently than a few years ago. And it is always asked by mentioning composer in some way.

Like: “I know how to include this PHP library if I would use composer, but how do get this to work for my TER extension?”

Or: “I would like to use this PHP library in my extension, but it must also work for people not using composer?”

I can only speculate, but I assume the reason that this question pops up so often now is, that composer has already transformed the way people in the PHP world think about code.

The years B.C.

In the years B.C. (before composer), PHP code was handled differently. It was randomly copied from forums, publicly available code repositories or at best taken from PEAR and then burried into an extension and most of the time kept there untouched for ages. Updates or bugfixes in libraries weren’t handled most of the time and sometimes the library code was changed arbitrarily for the needs of the extension, making updates of the library code close to impossible. But it worked well (enough). It worked well enough because nobody knew a better way to handle it.

The age of composer

Then some crazy people decided to change the PHP world and created a set of tools and kickstarted an ecosystem which since then thrives. You need a PHP library in your composer managed project? Go find it on Packagist, type `composer require name-of-vendor/name-of-package` in your project root and be done. Using the library code just works. Updating it works simply by typing `composer update name-of-vendor/name-of-package`. If the library needs another library, or anything needs a special PHP version, or another library needs another version of a library you intend to use, composer will resolve everything for you just nicely (most of the times).

The TYPO3 Extension Repository (TER)

The TER exists for years and it an integral part of the ecosystem that made TYPO3 thrive. However technically it is a bit aged. The TER and the TYPO3 Extension manager can handle updates of TYPO3 versions and even extensions. It can also handle dependencies to other extensions. However resolving these dependencies is less stable and capable than what composer does. Most importantly it can only handle dependencies from the TYPO3 ecosystem, not from the composer ecosystem. This means, if you want to publish an extension to TER, that needs code from a composer library, you need to take care and **include** it into your TYPO3 extension, like in the old B.C. days.

Bundling libraries in phar files

There is a project, which you can use to bundle any composer based library, or even a set of libraries as single phar file. I find it much nicer to ship one file with dependent third party code, instead of hundreds of files.

Benefits

1. **More secure:** The complete phar might be exposed but not single files.
2. **Cleaner:** It is much more clear which is a library and which is your code.
3. **More reliable:** Code scans or code formatters or linters in your CI environment will ignore phar files by default
4. **Smaller:** The phar files are smaller in size than the sum of all files

And probably more, but these are enough to convince me.

Howto

Install the tool globally and make sure `$HOME/.composer/vendor/bin` is set in your path:

```
composer global require clue/phar-composer
```

Create a directory in your extension and create the phar archive (symfony/process for example):

```
mkdir Libraries
phar-composer build symfony/process Libraries/symfony-process.phar
```

In your extension code, before you want to use the libraries in question, include the autoloader file of your phar:

```
use TYPO3\CMS\Core\Utility\ExtensionManagementUtility

@include 'phar://' . ExtensionManagementUtility::extPath('ext-key') . 'Libraries/symfony-process.phar/vendor/autoload.php';
```

Optimization

There are several things I have done to optimize this in my TYPO3 Console package:

1. I excluded the phar from the packages delivered by composer. In that case the phar is not needed.
2. I recreate the lib with the current version on every travis build, which means the TER upload will always include the current version
3. I set `classmap-authoritative` to `true` and `prepend-autoloader` to `false` for the phar class loader. That way I can just include the phar if present and the autoloader of the phar is only used when the class is not loadable.

Summary

These files do not replace a proper dependency management. It still is a relic from the old times to bundle third party code and especially when this becomes more public, we all will run into conflicts when one extension requires another incompatible version of the same library. But I consider this OK, to support legacy systems at least for a while.

But if you can, you should start setting up your TYPO3 projects with composer. Mid term the TER will also be replaced with something that speaks composer but not “Extension Manager”.

Have fun, stay clean.