Making extensions distribution-ready

In this chapter, you learned how to create various types of Yii extensions. Now we’ll talk about how to

share your results with people and why it’s important.

Getting ready

Let’s form a checklist for a good extension first. A good programming product should follow these points:

* Good coding style
* People should be able to find it
* A consistent, easy to read, and easy to use API
* Good documentation
* Extension should apply to the most common use cases
* Should be maintained
* Well-tested code, ideally with unit tests
* You need to provide support for it

Of course, having all these requires a lot of work, but these are necessary to create a good product.

How to do it...

1. Every modern PHP product must follow the PSR4 standards of autoloading and the PSR1 and PSR2 standards of the coding style from the <http://www.php-fig.org/psr/>guide.
2. Let’s review our list in more detail, starting with the API. The API should be consistent, easy to read, and easy to use. Consistent means that the overall style should not change, so no different variable naming, no inconsistent names such as isFlagi() and isNotFlag2(), and so on. Everything should obey the rules you’ve defined for your code. This allows less checking of documentation and allows you to focus on coding.
3. A code without any documentation is almost useless. An exception is a relatively simple code, but even if it’s only a few lines, it doesn’t feel right if there is not a single word about how to install and use it. What makes good documentation? The purpose of the code and its pros should be as visible as possible and should be written loud and clear.
4. A code is useless if developers don’t know where to put it and what should be in the application configuration. Don’t expect that people know how to do framework-specific things. The installation guide should be verbose. A step-by-step form is preferred by a majority of developers. If the code needs SQL schema to work, provide it.
5. Even if your API methods and properties are named properly, you still need to document them with PHPDoc comments specifying argument types and return types, providing a brief description for each method. Don’t forget protected and private methods and properties since sometimes it’s necessary to read these to understand the details of how code works. Also, consider listing public methods and properties in documentation so it can be used as a reference.
6. Provide use case examples with well-commented code. Try to cover the most common ways of extension usage.