Proposing PHK as distribution format

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: PHK Component Proposal

Proposed Component Name	PHK
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/PHK
Proposers	Francois Laupretre Darby Felton, Zend liaison
Revision	0.1 - 22 Jan 2007: Creation (wiki revision: 15)

Table of Contents

1. Overview

PHK is a new PHP-oriented package software. Its purpose is to distribute and run a software/library as a single package file (you may think of it as a very distant descendant of PHAR or a PHP-oriented jar-like package format). Distributing the Zend Framework libraries and documentations as PHK packages would be easy to implement and would bring many benefits.

I built some demonstration packages from the Zend framework original distribution:

- 2 library packages (core and incubator)
- 2 API documentation packages (core and incubator)
- 12 End-user documentation packages: 11 for core (1 per language) and 1 for incubator (english)

Here are the main benefits I see :

- keeping a library or any functional unit as a single file makes it more robust, consistent, easy to distribute, manage, remove, upgrade.
- The most important, IMHO, is consistency. It is critical for software users to ensure that what they use is exactly what it is supposed to. The main problem I see with most PHP software is that it is provided as a file tree. With a PHK package, it is impossible to inadvertently remove a source file, there cannot be access modes problems, everything is done to detect data corruption... The point here is that a software is an entity, it is consistent and must remain exactly this way. Keeping the software as a single file is a good starting point but PHK also provides additional features as CRC checking and digital signatures.
- PHK provides a 'webinfo' mode which gives users a standard way to access package information, license, a documentation page, and other information you may customize at will. As every other feature, all these information are served from the package as it is, keeping it consistent.
- Embedded unit tests allow to keep unit tests with the code. There is no performance impact as the test code is included in a sub-package, which is ignored until you explicitely run the test procedure.
- As noted above, PHK includes a digital signature feature. It is trivial to note how important it can be for corporate users... I think that it can be a very interesting feature for the Zend Framework. Please note that, unlike the jar signature feature, PHK signatures are verified

offline and don't have any impact on runtime performance.
- PHK includes an embedded autolad manager. It means that, in the case of 100% OO libraries like the Zend framework, it allows to remove every include/require/load_class from the user code. This feature is transparent and extracts the symbol names from the PHP source code at package creation time (no map to maintain by hand). IMHO, this one is very interesting because it provides a sort of 'linker' feature to the library with every benefits of just-in-time loading. Of course it replaces the current 'class name to file path' mapping mechanism, making you free to locate your classes and interfaces anywhere you want. As a side effect, it allows to put more than one class/interface in a single source file. I don't say it is always better but it removes the previous constraint.
- A minor one is the ability to specify the list of PHP extensions needed by the package. I personnally hate receiving an ugly error message just because some needed extension is not loaded. This happens today with the Zend Framework when the iconv extension, for instance, is unavailable. Using the PHK 'needed_extensions' option, if the extension cannot be loaded when the package starts its execution, an exception is thrown with an explicit message giving the list of missing extensions. Such a small feature allows to spare much support time and user frustration !
- This one is very personal: The Framework is split in 2 parts but I personally would prefer a more layered approach. For instance, I don't see why the Cache backends, Db adapters, or services are not implemented as optional plugins. PHK could help for this, especially since the 0.4.4 release, which comes with a new plugin facility. Now, it can easily be the basis for an architecture where single-file components communicate with the core through standard interfaces and check their mutual dependencies through a core package manager. OK, that's all, you all know what a plugin approach provides. But, once again, it is just a personal view, no flame for this, please  ...

2. References

- The PHK project home page (contains demonstration packages of the Zend framework libraries/documentations)

3. Component Requirements, Constraints, and Acceptance Criteria

- Needs PHP 5.1.0 or newer.

4. Dependencies on Other Framework Components

In the Zend framework example packages, as in the original version, the unit tests need PHPUnit to run. It could be included in the packages (just a matter of choice). Also, as in the original non-packaged version, the incubator unit tests need the core library.

5. Theory of Operation

Look at the Zend Framework building kit downloadable from http://phk.tekwire.net. It is quite simple and shows how the packages are generated.

6. Milestones / Tasks

As noted above, demonstration packages are available. They are not perfectly functional because some of the unit tests still fail, but take these packages as proofs of concept: their purpose is to demonstrate that it is possible to adapt the libraries and documentations to PHK with minimal efforts. So, for instance, I didn't spend more time trying to fix the 3 unit tests which keep failing in the core libraries because I considered that it was more important to keep the building kit easy to understand. In this respect, I must conclude that, considering the minimal filter which are applied to the PHP sources before building the package, failing only 5 unit tests out of 775 can be considered as a success (it is even beyond my initial expectations  .

Please note that, except for these unit tests, all the tests/demos I tried using the Zend framework packages worked exactly the same as with the original libraries.

Although it was not PHK's primary goal, I also discovered that PHK can be a very interesting tool to build documentation packages, e.g. static or pseudo-static content. Once again, I didn't have that in mind when I started the project but, if you have a look at the PHK documentation demo packages, you will see that, for instance, distributing and serving the API documentations as PHK single-file packages can bring several benefits like more consistency, on-the-fly compression, an embedded mime resolution table (independant of web server config)...

The only data I don't have is the performance overhead to consider. I tried to get timings but the tests and demos included in the Zend framework are too small and I didn't get anything reliable. One can note that it is intrinsically hard to measure because the overhead essentially depends on disk access times. With a fast Linux server (Dell 6850), I even measured that the Pdf demo always ran slightly faster when using the library as a package instead of original separate source files! It is just a matter of disk cache and access/transfer speed. So, I think it will be very hard to determine an average overhead value. Until somebody finds a better value, I keep considering it is small because, as I noted above, in the tests I did, I could not measure any difference.
<Edit 26-MAR-2007> I just started an 'accelerator' C extension for PHK. The 'accelerator' approach was chosen because the PHK runtime spends more than 70% of its time in less than 10% of the code. This CPU-intensive code is now well identified and the corresponding C accelerator extension will be available soon. Please note that the accelerator will always remain optional: if it is present, it will be used. If not, the result will be the same, just slower.

As you want milestones, please note that the PHK project today represents more than 9,000 lines of code and about 700 hours of work.

PHK status: PHK is considered alpha version because it is not used by enough people yet, but it is fully functional and just lacks testing. If the Zend Framework, for instance, decides to use it, it can go to production release very fast.

7. Class Index

PHK brings an extensive PHP API. In the case of the Zend Framework, I didn't have to use any API call.

8. Use Cases

Case 1: Running the core unit tests

1. Download the PHPUnit demonstration package and save the file in your include path.
2. Download the 'Zend Framework (core library)' demonstration package and save it in your current directory.
3. Run: php zfw_core.phk @check
4. You can do the same with the incubator unit tests but they also need the core library, which means that they need to find the zfw_core.phk file in the include path.

Case 2: Running the PDF demo

1. Download the 'Zend Framework (core library)' demonstration package and save the file in your include path.
2. Download the Zend Framework and extract the 'demos/Zend/pdf' directory.
3. In this directory, edit the 'demo.php' file :

- Remove the line starting with 'set_include_path'.
- replace the line containing
  require_once 'Zend/Pdf.php'
  by
  require_once 'zfw_core.phk'.
  4. Ensure your PHP contains the GD extension (with jpeg support).
  5. Run 'php demo.php test.pdf result.pdf'

9. Class Skeletons

None

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>