Zend_Loader plugin directory support - Stanislav Malyshev

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

This proposal is implemented as Zend_Loader_PluginLoader

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

Zend Framework: Zend_Loader plugin directory support - Stanislav Malyshev Component Proposal

Proposed Component Name	Zend_Loader plugin directory support - Stanislav Malyshev
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Loader plugin directory support - Stanislav Malyshev
Proposers	Stanislav Malyshev
Revision	1.0 (wiki revision: 15)

Table of Contents

1. Overview

While working with the Framework, it might be needed to use certain classes belonging to other libraries or class bundles. These classes do not reside inside Framework directories and sometimes do not follow same file structure conventions as the Framework. However, it would be convenient if such files could participate in Framework loading mechanisms - for example, if the user could configure the directory where the PEAR classes are kept and then use Framework class loader (automatically or manually) to access these classes. This would allow to use the same loading mechanism across the application and not try to navigate between different loading mechanisms using by libraries or re-invent loaders anew.

The present proposal is intended to allow Zend_Loader class support loading (and autoloading) sets of classes that do not reside in main framework directory. Also, it provides base for implementing plugin structures for custom plugin sets - i.e. tag or helper library for template engine or component set for CMS.

2. References

3. Component Requirements, Constraints, and Acceptance Criteria

- The plugin loader will support autoloading classes from directories not residing in main Framework tree
- The plugin loader will support framework naming scheme (X/Y/Z.php) and other naming schemes (e.g. X_Y_Z/X_Y_Z.php).
- The plugin loader will be activated only when needed

4. Dependencies on Other Framework Components

- Zend_Loader
- Zend_Exception

5. Theory of Operation

The proposed functionality would allow the Framework loader to load - automatically or manually - classes that do not reside in the main framework directory, such as component sets, external libraries, etc. The filename that is derived from the class name can be composed in two ways - the "deep" way as Framework and PEAR do - i.e. class `My_Class_Foo` resides in `My/Class/Foo.php` - and the "shallow" way that is more suited for multiple unrelated components when class `My_Class_Foo` resides in `My_Class_Foo/My_Class_Foo.php`.

Also, the notion of **prefix** would be supported - i.e. if prefix `"My_Classes"` is defined, then class name `My_Classes_DB_Table` would be looked up in `DB/Table.php` in deep configuration and in `DB_Table/DB_Table.php` in shallow configuration.

The user would be also able to "hint" the loader which library the loaded class belongs to, thus shortening the search for the right file. It would not be necessary for the user to know where the library is located to load classes from it, and knowing the name would be optional - if it is not used, the search would be performed in all known libraries together with the include path.

The library set would be searched after the main include path is searched, and also would be searched whenever the class loading is attempted without specifying the directory. This should not have negative impact on the lookups, since if the file is not found in the include paths, it means either it does not exist (in which case we'd error out so slowdown doesn't matter) or it is found in one of the library directories, which means we would not find it using regular include path search. In the latter case, as described above using hinted manual loader might be recommended.

The search would go over all the library paths defined, using depth setting defined by the library path in each case, and try to resolve the class name with each path. If no resolution could be found, the exception is thrown.

TBD: we may want to think about joining `loadClass` with `loadLibraryClass` somehow, though because the former receives the directory list and the latter receives a hint it is not clear how to make it.

6. Milestones / Tasks

- Milestone 0: Complete the proposal
- Milestone 1: Collect feedback and refine the proposal
- Milestone 2: Provided the proposal is deemed acceptable, initial implementation checked into the incubator
- Milestone 3: Refine the implementation, document, create unit tests
- Milestone 5: Merge into the main Zend_Loader class

7. Class Index

- Zend_Loader

8. Use Cases

UC-01

Define prefixed class library with shallow naming scheme, use autoloader:

```
Zend_Loader::setLibraryPath("components", "/usr/share/lib/Zend/Components",
"Zend_Component", Zend_Loader::SHALLOW);
//....

$comp = new Zend_Component_Some_Table();
// Class was loaded from /usr/share/lib/Zend/Components/Some_Table/Some_Table.php
```

UC-02

Define non-prefixed library with deep naming scheme, named "pear", then use library hint to manually load class:

```
Zend_Loader::setLibraryPath("pear", "/usr/share/lib/pear");
//.....
ZendLoader::loadLibraryClass("OS_Guess", "pear");
$sys = new OS_Guess($uname);
```

9. Class Skeletons

```
class Zend_Loader {
const DEEP = 0;
const SHALLOW = 1;

public function setLibraryPath($name, $dirs, $prefix = null, $type = ZendLoader::DEEP)
{}
}

public function loadLibraryClass($classname, $library = null) {}
}
}
```

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