Zend_Loader_Autoloader_Resource - Matthew Weier O'Phinney

<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: Zend_Loader_Autoloader_Resource Component Proposal

Proposed Component Name	Zend_Loader_Autoloader_Resource
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Loader_Autoloader_Resource
Proposers	Matthew Weier O'Phinney
Zend Liaison	Ralph Schindler
Revision	1.0 - 1 January 2008: Initial Draft. (wiki revision: 6)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Loader_Autoloader_Resource is intended to simplify loading resources that do not have 1:1 class prefix/filesystem mappings. The primary use case is for use with modular applications, to allow autoloading of module-specific classes.

2. References

- Pastebin resource loader implementation
- Zend_Controller_Action_Helper_ResourceLoader proposal

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** implement Zend_Loader_Autoloader_Interface.
- This component **will** require specification of both a prefix and base directory

- This component **will** provide default component/subdirectory mappings
    - Users **will** be able to register custom component/subdirectory mappings
    - This component **will** allow autoloading arbitrarily nested classes under existing component maps
- This component **should** also act as an object factory and registry for the given resource namespace

4. Dependencies on Other Framework Components

- Zend_Loader_Exception
- Zend_Loader_Autoloader

5. Theory of Operation

Typically, this component will be instantiated in a module's bootstrap. It's purpose is to provide the ability to autoload components in that subdirectory, assuming a standard directory layout.

For instance, the bootstrap might do the following:

```
$autoloader = new Zend_Loader_Autoloader_Resource(array(
    'prefix'   => 'Spindle',
    'basePath' => realpath(dirname(__FILE__)),
));
Zend_Loader_Autoloader::pushAutoloader($autoloader);
```

This would then allow your module code to use autoloading to arbitrary resources. For instance, the following directory structure would establish the associated classes:

```
application/
    modules/
        spindle/
            models/
                Bug.php   // Spindle_Model_Bug
            forms/
                Bug.php   // Spindle_Form_Bug
```

When the resource autoloader is in play, you can simply instantiate these classes, without worrying where they are:

```
$form = new Spindle_Form_Bug();
```

6. Milestones / Tasks

- Milestone 1: [DONE] Proposal created
- Milestone 2: [DONE] Working prototype checked into public repository
- Milestone 3: Community review of proposal

- Milestone 4: Proposal acceptance
- Milestone 5: Passing unit tests and initial documentation committed to incubator
- Milestone 6: Review for inclusion in trunk

7. Class Index

- Zend_Loader_Autoloader_Resource

8. Use Cases

UC-01

Basic usage:

```
$resourceLoader = new Zend_Loader_Autoloader_Resource(array(
    'prefix'   => 'Spindle',
    'basePath' => realpath(dirname(__FILE__)),
));
Zend_Loader_Autoloader::pushAutoloader($resourceLoader);
```

UC-02

Using as an object factory and registry:

```
/* Spindle_Model_Bug */
$bugModel = $resourceLoader->getModel('bug');
```

UC-03

Registering custom resource paths:

```
// Adds 'validators' resource, mapping to basePath + /validators, using prefix
// $prefix _Validate.
$resourceLoader->addResourceType('validator', 'validators', 'Validate');

// e.g., this loads the class from
// application/modules/spindle/validators/Foo.php
$foo = new Spindle_Validate_Foo();
```

9. Class Skeletons

```
class Zend_Loader_Autoloader_Resource implements Zend_Loader_Autoloader_Interface
{
    public function __construct($options);

    public function initDefaultResourceTypes();

    public function __call($method, $args);

    public function autoload($class);

    public function setOptions(array $options);

    public function setPrefix($prefix);

    public function getPrefix();

    public function setBasePath($path);

    public function getBasePath();

    public function addResourceType($type, $path, $prefix = null);

    public function addResourceTypes(array $types);

    public function setResourceTypes(array $types);

    public function getResourceTypes();

    public function hasResourceType($type);

    public function removeResourceType($type);

    public function clearResourceTypes();

    public function setDefaultResourceType($type);

    public function getDefaultResourceType();

    public function load($resource, $type = null);
}
```

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