Zend_Build - Wil Sinclair

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

<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_Build Component Proposal

Proposed Component Name	Zend_Build
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Build
Proposers	Wil Sinclair
Revision	1.0 - 29 November 2007: Proposal created. 1.1 - 21 December 2007: Proposal updated. 1.2 - 17 January 2008: Proposal updated after component responsibilities finalized. (wiki revision: 41)

Table of Contents

1. Overview

Zend_Build provides a project management infrastructure for Zend Framework. This system introduces the concepts of 'actions' and 'resources' defined in both core and external components that can be used in managing and building a Zend Framework project. Resources can be nested to create project profiles that can be stored on the filesystem in Zend_Config's XML format to provide metadata to CLI, Rich UI, and other tools for project introspection and management. All actions and resources must be declared in build metadata files, which include metadata such as short name, help, and command line options.

2. References

- Ruby on Rails
- Rails Command Line Reference
- Symfony Generators
- CakePHP
- Zend_Config_Configurable

3. Component Requirements, Constraints, and Acceptance Criteria

- The build infrastructure **must** support the requirements of actions and resources used to perform common tasks on Zend Framework projects, including, but not limited to, project generation/deletion, scaffolding CRUD actions, and unit test CRUD actions. Note that this project will not deliver these specific capabilities, but will provide an infrastructure upon which to implement them easily. Please refer to Zend_Build for Core for specific implementations of actions and resources.
- **Must** allow all actions and resources to be constructed using Zend_Config instances.
- Resources **must** be nestable in both configurations and object graphs.
- Actions and resources **must** work intuitively within the context provided by a project profile.
- Resources **must** be serializable to XML fragments and these XML fragments **must** be composable to form valid XML 1.0/Zend_Config XML project profile documents.
- All Zend_Config objects used in Zend_Build **must** be serializable to the filesystem in XML format. The functionality for rendering and writing these documents in valid Zend_Config_Xml format will be implemented in the Zend_Build component for now because it is TBD whether this would promote best practices if added to Zend_Config.
- All Zend_Config_Xml files **should** support XML attributes in both reading and writing for enhanced readability of project profiles.
- Actions and resources **must** provide full usage documentation in the build manifest files.
- Each action and resource **must** have its own build manifest file with file names of the form '<action or resource classname>-ZFManifest.*'.
- Zend_Build **must** support template files used in creating files for project creation, etc. These files will be valid php files that output the required file.
- Actions **should** be atomic where possible.
- It would be **nice to have** the ability of rewinding actions, including deletes.
- This component **should** be able to read and write project profiles in any format supported by Zend_Config including INI.
- It would be **nice** if the component stored MD5 checksums for all files to detect file changes between operations.
- This componet **must** verify that an application profile is valid before using it for any operations.
- Any instance of Zend_Build_Resource **must** be instantiable using an instance of Zend_Config.

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Config
- Zend_Loader
- Zend_Log

5. Theory of Operation

The Zend_Build component will provide a basic project management system for Zend Framework. This component introduces the concepts of 'actions', 'resources', and 'project profiles'. In general an action is always required to perform an operation on the project, and some actions perform these operations on or with a specified resource. Project profiles are nested compositions of resources that make up a project. Resources- and, by extension, project profiles- genrally contain metadata about project resources. That is to say, a project profile is useful in creating a project skeleton or operating on an existing project, but generally does not contain the code and can not be used to fully re-generate a project after the user has modified the system-generated project skeleton.
This component will deliver basic actions and resources to support basic filesystem operations- such as a 'delete' action and a file 'resource' to support deleted a file- but more specific actions and resources will be developed in the component to which they pertain (see Zend_Build for Core). This component will also provide resources and actions that pertain to most or all projects, such as library directory and public directory. The Zend_Build component will generally follow ZF's 'use at will' architecture and may be used to describe and manage projects that don't use all ZF core components- or any core components besides Zend_Build's direct and indirect dependencies.
Actions and resources can be added to the library by first implementing Zend_Build_Action_Interface and Zend_Build_Resource_Interface, respectively. Next, the developer must add one manifest file per action or resource to his/her component with the name '<classname>-ZFManifest.*'. These manifest files can be any file type supported by Zend_Config, but must contain all information about the resource for building the project and any other functionality that requires metadata, such as that outlined in [Zend_Console]. Zend_Build requires the 'type' and 'name' attributes and other attributes or elements may be used by Zend_Build. The manifest format is extensible, however, so the developer defining the action or resource may add metadata required by other components unknown to Zend_Build. All such metadata will simply be ignored by Zend_Build. The following is an example of such a file in the xml format defining a resource that may be used by Zend_Console:

```
<?xml version="1.0"?>
<configdata>
    <command_context type = 'resource' name = 'library_dir' alias = 'ld'
class='Zend_Build_Resource_LibraryDirectory'>
        <attribute getopt='apple|a'>
            <usage>
                This is how you use apple with library_dir.
            </usage>
        </attribute>
        <attribute getopt='banana|b=i'>
            <usage>
                This is how you use banana with library_dir: banana must have an
integer param.
            </usage>
        </attribute>
        <attribute getopt='pear|p-s'>
            <usage>
                This is how you use pear with library_dir: pear can have a string
param.
            </usage>
        </attribute>
     </command_context>
</configdata>
```

A project profile contains information about the resources in a project with a tree-like structure. This project profile may be serialized to the XML 1.0 dialect defined by Zend_Config and used to store any metadata about the project that is useful to Zend_Build actions or resources. As a rule, no PHP code will be serialized to the project profile, although components may be able to generate code using a combination of the data in a project profile and separate templates stored in the Zend Framework components to which they pertain. The following is an example of a project profile for a basic MVC application:

```
<?xml version="1.0"?>
<configdata>
<project name="basic_mvc_project">
  <application_dir name="application">
    <controller_dir name="controllers"/>
    <model_dir name="models"/>
    <view_dir name="views"/>
  </application_dir>
  <data_dir name="data"/>
  <public_dir name="htdocs"/>
  <library_dir name="lib"/>
  <trash_dir maxSize="100" name="trash"/>
</project>
</configdata>
```

Since Zend_Config objects are used heavily in this component, all actions and resources will be instantiable from valid Zend_Config objects. A Configurable interface is provided to bring consistency to such instantiations. If the Zend_Config_Configurable is approved for core in the 1.5 release, this component will use the Zend_Config_Configurable.

6. Milestones / Tasks

- Milestone 1: [DONE] Draft proposal submitted for community review
- Milestone 2: Working prototype checked in to incubator for community evaluation
- Milestone 3: Proposal finalized
- Milestone 4: All unit test passing with coverage at 90% or higher for all code
- Milestone 5: Documentation finished in English

7. Class Index

- Zend_Build_Configurable
- Zend_Build_AbstractConfigurable
- Zend_Build_Exception
- Zend_Build_Manifest
- Zend_Build_Manifest_NameConflictException
- Zend_Build_FileTemplate
- Zend_Build_Action
- Zend_Build_Action_Interface
- Zend_Build_Action_Abstract
- Zend_Build_Action_Create
- Zend_Build_Action_Delete
- Zend_Build_Resource_Interface
- Zend_Build_Resource_Abstract
- Zend_Build_Resource_Factory
- Zend_Build_Resource_ConfigXmlWriter
- Zend_Build_Resource_File
- Zend_Build_Resource_Directory
- Zend_Build_Resource_ApplicationDirectory
- Zend_Build_Resource_ControllerDirectory
- Zend_Build_Resource_DataDirectory
- Zend_Build_Resource_LibraryDirectory
- Zend_Build_Resource_ModelDirectory
- Zend_Build_Resource_ModuleDirectory
- Zend_Build_Resource_Project
- Zend_Build_Resource_PublicDirectory
- Zend_Build_Resource_TempDirectory
- Zend_Build_Resource_TrashDirectory
- Zend_Build_Resource_ViewDirectory

8. Use Cases

This component is primarily consumed by Zend_Console and other build interfaces. The use cases below show examples of defining an action and a resource.

UC-01

```
class Zend_Build_Resource_DummyFile extends Zend_Build_Resource_File
{
    public saySomething() { print('duh'); }
}
```

```
<?xml version="1.0"?>
<configdata>
    <context type = 'resource' name = 'dummy_file' alias = 'df'
class='Zend_Build_Resource_DummyFile' />
</configdata>
```

```
class Zend_Build_Action_SaySomething extends Zend_Build_Action_Abstract {}
```

```
<?xml version="1.0"?>
<configdata>
    <context type='action' name = 'say_something' alias = 'ss'
class='Zend_Build_Action_SaySomething'>
    </context>
</configdata>
```

The command **'zf say_something dummy_file'** on the command line (assuming Zend_Console is installed) would now return 'duh'.

9. Class Skeletons

```
interface Zend_Build_Configurable
{
    /**
     * Configure the component with Config object set previously
     */
    public function configure();

    /**
     * Set the Config object on this object
     *
     * @param Zend_Config $config Configuration object
     */
    public function setConfig(Zend_Config $config);

    /**
     * Get configuration object
     *
     * @return Zend_Config|void
     */
    public function getConfig();

    /**
     * Instantiate component from configuration object
     *
     * @param Zend_Config $config  Configuration object
     * @return object     Instance of the configured component
     */
    public static function getConfigurable(Zend_Config $config);
}
```

```
abstract class Zend_Build_AbstractConfigurable implements Zend_Build_Configurable
{
    /**
     * @see Zend_Build_Configurable::getConfig()
     */
    public function getConfig();

    /**
     * @see Zend_Build_Configurable::getConfig()
     */
    public function setConfig(Zend_Config $config);

    /**
     * @see Zend_Build_Configurable::getConfigurable()
     */
    public static function getConfigurable(Zend_Config $config);
}
```

```
class Zend_Build_Manifest_NameConflictException extends Zend_Build_Exception {}
```

```
class Zend_Build_FileTemplate
{
    public function evaluate($filename);
}
```

```
interface Zend_Build_Action_Interface
{
    /**
     * Validate all attributes of this command.
     */
    public function validate ();

    /**
     * Execute this command.
     */
    public function execute ();

    /**
     * Set the profile to execute against.
     */
    public function setProfile ($profile);

    /**
     * Get the profile that would be executed against.
     */
    public function getProfile ();

    /**
     * Set the options to execute with. Options added later override previous options
with the same name.
     */
    public function setOptions (array $options);

    /**
     * Get the options to execute with.
     */
    public function getOptions ();

    /**
     * Set the resource to execute this command on.
     */
    public function setResources (array $resources);

    /**
     * Get the resource to execute this command on.
     */
    public function getResources ();
}
```

```
abstract class Zend_Build_Action_Abstract implements Zend_Build_Action_Interface
{
```

```
    /**
     * Get the profile that would be executed against.
     */
    public function getProfile ()
    {
        return $_profile;
    }

    /**
     * Set the profile to execute against.
     */
    public function setProfile ($profile)
    {
        $_profile = $profile;
    }

    /**
     * Get the options to execute with.
     */
    public function getOptions ()
    {
        return $_options;
    }

    /**
     * Set the options to execute with. Options added later override previous options
with the same name.
     */
    public function setOptions (array $options)
    {
        $_options = $options;
    }

    /**
     * Get the resources to execute this command on.
     */
    public function getResources ()
    {
        return $_resouces;
    }

    /**
     * Set the resources to execute this command on.
     */
    public function setResources (array $resources)
    {
        $_resources = $resources;
    }

    /**
     * Default implementation of execute(). Should work or offer reuse for many
commands.
     */
    public function execute ();

 /**
     * Default implementation of execute(). Should work or offer reuse for many
commands.
     */
```

```
    public function validate ();

    /**
     * Return string representation (which will also be a valid CLI command) of this
command.
     */
```

```
    public function __toString ();
}
```

```
class Zend_Build_Action_Create extends Zend_Build_Action_Abstract {}
```

```
class Zend_Build_Action_Delete extends Zend_Build_Action_Abstract {}
```

```
interface Zend_Build_Resource_Interface
{
    /**
     * Returns true if an instance of this resource has been updated since it was
created with CLI, false otherwise.
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function updated ();

 /**
     * Returns true if an instance of this resource already exists in this project,
false otherwise.
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function exists ();

    /**
     * Gets the parent of this resource instance
     */
    public function getParent ();

    /**
     * Gets the children of this resource instance
     */
    public function getChildren ();

    /**
     * Adds a child to the end of the list of children for this resource
     */
    public function addChild (Zend_Build_Resource_Interface $child);

    /**
     * Removes a child from the list of children for this resource and returns the new
list of children
     *
     * @return array New list of children with $child removed
     */
    public function removeChild (Zend_Build_Resource_Interface $child);

    /**
     * Adds all children to the end of the list of children for this resource
```

```
     */
    public function addAllChildren (array $children);

    /**
     * Removes all children from the list of children for this resource and returns
all removed children
     *
     * @return array All children removed from this build resource
     */
    public function removeAllChildren ();

    /**
     * Gets the type of this resource instance
     */
```

```
    public function getType ();
}
```

```
abstract class Zend_Build_Resource_Abstract implements Zend_Build_Resource_Interface
{
    public $name;
    protected $_parent;
    protected $_children;
    public function __construct ($name, array $children = array())
    {
        $this->name = $name;
        $this->addAllChildren($children);
    }

    /**
     * Returns true if an instance of this resource has been updated since it was
created with CLI, false otherwise.
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function updated ()
    {}

 /**
     * Returns true if an instance of this resource already exists in this project,
false otherwise.
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function exists ()
    {}

    /**
     * Creates this instance of the resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function create ()
    {}

    /**
     * Deletes this instance of this resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function delete ()
    {}

    /**
     * Gets the parent of this resource instance
     */
    public function getParent ()
```

```
    {
        return $this->_parent;
    }

    /**
     * Removes the parent from this instance of build resource
     */
    protected function removeParent (Zend_Build_Resource_Interface $child)
    {
        $child->_parent = NULL;
    }

    /**
     * Gets the children of this resource instance
     */
    public function getChildren ()
    {
        return $this->_children;
    }

    /**
     * Adds a child to the end of the list of children for this resource
     */
    public function addChild (Zend_Build_Resource_Interface $child)
    {
        $child->_parent = $this;
        $this->_children[Zend_Font - Karol Babioch] = $child;
    }

    /**
     * Removes a child from the list of children for this resource and returns the new
list of children
     *
     * @return array New list of children with $child removed
     */
    public function removeChild (Zend_Build_Resource_Interface $child)
    {
        $child->removeParent();
        $this->_children = array_diff($_children, array($child));
        return $_children;
    }

    /**
     * Adds all children to the end of the list of children for this resource
     */
    public function addAllChildren (array $children)
    {
        foreach ($children as $child) {
            $this->addChild($child);
        }
    }

    /**
     * Removes all children from the list of children for this resource and returns
all removed children
     *
     * @return array All children removed from this build resource
     */
    public function removeAllChildren ()
```

```
{
    $removed_children = $this->_children;
    foreach ($removed_children as $child) {
        $child . removeParent();
    }
    $this->_children = array();
    return $removed_children;
}

/**
 * Gets the type of this resource instance
 */
public function getType ()
{
    get_class($this);
}

protected function readChecksum ()
{}

protected function writeChecksum ()
{}

/**
 * Default implementation of toString for all build resources
 */
public function __toString ()
{
    return $this->name;
}

/**
 * Default implementation of equals should work for all resources
 */
public function __equals ($other)
{
    if (! isset($other))
        return false;
    if ($this->getType() != $other->getType())
        return false;
    if ($this != $other)
        return false;
    foreach ($this->getChildren() as $key => $child) {
        $other_children = $other . getChildren();
        if (! isset($other_children) || ! array_key_exists($key, $other_children))
            return false;
        if ($child != $other_children[$key])
            return false;
    }
    return true;
```

```
    }
}
```

```
/**
 * This is a static class with utility methods for writing Zend_Config objects to
Zend_Config XML format.
 */
class Zend_Build_Resource_ConfigXmlWriter
{
    public static function writeConfigToXmlFile (Zend_Config $config, $filename) {}

    /**
     * Returns a SimpleXMLElement for the given config.
     *
     * @param Zend_Build_Resource_Interface Resource to convert to XML
     * @return string String in valid XML 1.0 format
     * @see Zend_Build_Resource_Interface
     */
    public static function getXmlForConfig (Zend_Config $config) {}
```

```
class Zend_Build_Resource_Directory extends Zend_Build_Resource_Abstract
{
    /**
     * @see Zend_Build_Resource_Interface
     */
    public function exists () {}

    /**
     * Creates this instance of the resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function create () {}

    /**
     * Deletes this instance of this resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function delete () {}

    /**
     * Returns the full path of this filesystem resource relative to the project root
     */
    public function getPath () {}
}
```

```
class Zend_Build_Resource_File implements Zend_Build_Resource_Interface
{
 public function __construct (string $name, Zend_Build_Resource $parent) {}

    /**
     * @see Zend_Build_Resource_Interface
     */
    public function exists () {}

    /**
     * Creates this instance of the resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function create (string $content = '') {}

    /**
     * Deletes this instance of this resource in a project
     *
     * @throws Zend_Build_Profile_Resource_Exception If authentication cannot be
performed
     */
    public function delete ();

    /**
     * Reads this file into a string and returns it.
     */
    protected function read() {}

    /**
     * Returns the full path of this filesystem resource relative to the project root
     */
    public function getPath () {}
}
```

```
class Zend_Build_Resource_ApplicationDirectory extends Zend_Build_Resource_Directory
{}
class Zend_Build_Resource_ControllerDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_DataDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_LibraryDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_ModelDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_ModuleDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_Project extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_PublicDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_TempDirectory extends Zend_Build_Resource_Directory {}
class Zend_Build_Resource_ViewDirectory extends Zend_Build_Resource_Directory {}

class Zend_Build_Resource_TrashDirectory extends Zend_Build_Resource_Directory
{
    /**
     * Maximum size of this directory in MB. This is a soft limit, since this limit is
enforced AFTER
     * the current command has run.
     */
    public $maxSize = 100;

    /**
     * Set maximum size in MB
     */
    public function setMaxSize (int $size)
    {
        $this->size = $size;
    }

    /**
     * Get maximum size in MB
     */
    public function getMaxSize ()
    {
        return $this->size;
    }
}
```

Sample project profile for MVC application:

```
$project = new Zend_Build_Resource_Project('basic_mvc_project', array(
   new Zend_Build_Resource_ApplicationDirectory('application', array(
    new Zend_Build_Resource_ControllerDirectory('controllers'),
    new Zend_Build_Resource_ModelDirectory('models'),
    new Zend_Build_Resource_ViewDirectory('views'))),
   new Zend_Build_Resource_DataDirectory('data'),
   new Zend_Build_Resource_PublicDirectory('htdocs'),
   new Zend_Build_Resource_LibraryDirectory('lib'),
   new Zend_Build_Resource_TrashDirectory('trash')));
```

The same sample profile for MVC application rendered as an XML configuration file:

```
<?xml version="1.0"?>
<configdata>
<project name="basic_mvc_project">
  <application_dir name="application">
    <controller_dir name="controllers"/>
    <model_dir name="models"/>
    <view_dir name="views"/>
  </application_dir>
  <data_dir name="data"/>
  <public_dir name="htdocs"/>
  <library_dir name="lib"/>
  <trash_dir maxSize="100" name="trash"/>
</project>
</configdata>
```

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