Zend_Tool - General

<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_Tool - General Component Proposal

Proposed Component Name	Zend_Tool - General
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Tool - General
Proposers	Ralph Schindler
Zend Liaison	TBD
Revision	1.0 - 1 January 2008: Initial Draft. (wiki revision: 8)

Table of Contents

1. Overview

This proposal is the "umbrella" proposal which will cover all of the requirements and goals for the "tooling" initiative within ZF. The actual implementations for each of the requirements will be provided by sub-proposals of this proposal.

By now, you are probably aware that there are several proposals in ZFPROP that are relate to or have Zend_Tool in their name, after all- thats probably why you are reading this page. If you have read through those proposals, either casually or in depth.. You probably have a few questions specifically as to how they are relate to one another. In this document, we will talk about the driving factors behind the Tooling initiative and what each proposal represents in terms of that initiative.

2. References

- Zend_Tool_Framework
- Zend_Tool_Framework_Endpoint_Cli
- Zend_Tool_Project
- Zend_Tool_CodeGenerator

3. Component Requirements, Constraints, and Acceptance Criteria

There are a few goals that this initiative is attempting to accomplish. Below is a list of the requirements (and their sub requirements) that the Zend Tool initiative sets out to accomplish:

- Ability to issue "tooling system" requests via the command line to facilitate:
    - rapid application development of php based applications
    - jump starting application structures with include the creation and alteration of
        - directories
        - files (classes, html, view scripts, etc.)
        - database schemas
        - config files
        - xml files
        - wsdl generation
        - etc.
- Ability of the "tooling system" to describe its full set of capabilities
- Ability to issue "tooling system" requests from multiple clients to interact with this "tooling system"
    - Zend Studio
    - Command Line
    - Other IDE (example: textmate via perhaps XML-RPC)
- Ability to extend the "tooling system" as simply as creating a php class.
- Ability to attach metadata to "tooling capabilities (provider)" and/or arbitrary "tooling" data to the system
- Ability for this "tooling system" to be bootstrapped automatically by detecting "capabilities" inside the php include-path.
- Ability to customize and use a "project profiling" system to facilitate:
    - defining project structures to be used in rapid application development
    - ability to serialize a "live" project profile inside a project
    - ability to keep project "artifacts" and the project profile in synchronization
- Ability to construct php based files, classes, methods, properties via an OO interface
- Ability to deconstruct and alter the contents of existing PHP files and classes

The above list represents the full set of USE CASES these set of tools will attempt to accomplish

4. Dependencies on Other Framework Components

- Zend_Exception

5. Theory of Operation

At first glance, this appears to be a large and thorough requirement list. That said, it seems only logical that we can further group these requirements into specialized groups (for for the purposes of Zend Framework) - separate but related proposals. In addition to separate proposals, it also makes the most sense if we can organize the code into logical classes and components that can implement, test and deliver for the requirements that they have been designed to accomplish. This allows us to focus on a small group of requirements that have minimal dependencies on other and sometimes larger requirements.

So, to start, its important to note that Zend_Tool, at its top most level it not a component per-se. There will be no Zend_Tool class. At the top most level, Zend_Tool serves only as a namespace which will contain a few components that fit within the "tooling" realm. "Tooling" in this sense, is meant to refer to any components who's primary focus is aiding developers build and deliver application and code and might not necessarily represent any functional aspect of the application the developer is building. To draw an analogy, Zend_Tool components are as similar to what Zend Studio for eclipse is for a ZF powered application development. Its primary purpose is as a "tool" the developer would use in order to aid in the applications development and in most cases would never be used in the actual deployed application. So put another way: this could be considered "development time" components, vs. they typical "runtime" components.

Breaking down the Requirements

Taking the list of requirements, we have been able to deterministically break them down into 3 logical groups - or component proposals. These functional groups should be able to accommodate all of the requirements above while minimizing the cross dependencies they have with one another. This proposal breaks them down into these three groups:

- The Tooling Subsystem - (aka Zend_Tool_Framework) - this system is responsible for creating a framework that will provide developers and consumers with easy to use interfaces for not only executing and dispatching "tooling requests" but also should make it easy for developers to enhance and extend the range of capabilities the system provides.
- The Project Management System - (aka Zend_Tool_Project) - this component is responsible for managing application projects. An application project would essentially be synonymous to a project inside your development environment.
- The Code Generator - (aka Zend_Tool_CodeGenerator) - this system is responsible for creating an API that will allow developers to create, modify and destroy blocks of code, be it Apache conf files, PHP, Javascript, etc.

Zend_Tool_Framework Requirements

- Ability of the "tooling system" to describe its full set of capabilities
- Ability to issue "tooling system" requests from multiple clients to interact with this "tooling system"
    - Zend Studio
    - Other IDE (example: textmate via perhaps XML-RPC)

- Ability to extend the "tooling system" as simply as creating a php class.
- Ability to attach metadata to "tooling capabilities (provider)" and/or arbitrary "tooling" data to the system
- Ability for this "tooling system" to be bootstrapped automatically by detecting "capabilities" inside the php include-path.

Zend_Tool_Framework_Endpoint_Cli Requirements

- Command Line Interface for tooling

Zend_Tool_Project Requirements

- Ability to customize and use a "project profiling" system to facilitate:
    - defining project structures to be used in rapid application development
    - ability to serialize a "live" project profile inside a project
    - ability to keep project "artifacts" and the project profile in synchronization

Zend_Tool_CodeGenerator

- Ability to construct php based files, classes, methods, properties via an OO interface
- Ability to deconstruct and alter the contents of existing PHP files and classes

Cross Dependencies

- Ability to issue "tooling system" requests via the command line to facilitate:
    - rapid application development of php based applications
    - jump starting application structures with include the creation and alteration of
    - directories
    - files (classes, html, view scripts, etc.)
    - database schemas
    - config files
    - xml files
    - wsdl generation
    - etc.

As you can see, the majority of requirements "fit" within one of the sub proposals. The last one, is a requirements that can only be fulfilled by utilizing all of the components together (this creates the super dependency). To understand how these proposals attempt to fulfill these requirements lets now take a look at the actual propsoals:

Zend_Tool_Framework

Zend_Tool_Framework_Endpoint_Cli

Zend_Tool_Project

Zend_Tool_CodeGenerator

6. Milestones / Tasks

Describe some intermediate state of this component in terms of design notes, additional material added to this page, and / code. Note any significant dependencies here, such as, "Milestone #3 can not be completed until feature Foo has been added to ZF component XYZ." Milestones will be required for acceptance of future proposals. They are not hard, and many times you will only need to think of the first three below.

- Milestone 1: design notes will be published here
- Milestone 2: Working prototype checked into the incubator supporting use cases #1, #2, ...
- Milestone 3: Working prototype checked into the incubator supporting use cases #3 and #4.
- Milestone 4: Unit tests exist, work, and are checked into SVN.
- Milestone 5: Initial documentation exists.

If a milestone is already done, begin the description with "[DONE]", like this:

- Milestone #: [DONE] Unit tests ...

7. Class Index

- Zend_Tool is a namespace only, implementations and classes can be found in the sub-proposals.

8. Use Cases

9. Class Skeletons

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