

Zend_Yaml - Pádraic Brady

<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_Yaml Component Proposal

Proposed Component Name	Zend_Yaml
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Yaml
Proposers	Pádraic Brady
Revision	0.10 - 26 March 2007 (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

YAML is a machine parsable data serialisation format for storing text, numerical data, arrays and more. It was designed for use with programming languages and has excellent support in Python, Perl and Ruby. Several C implementations exist and at least one (Syck) supports a PHP extension. YAML is a commonly used format for data files as an alternative to XML or INI.

Zend_Yaml aims to implement YAML 1.1 support for the Zend Framework. In the absence of a core YAML extension for PHP, it offers YAML support in native PHP. This will include an LL(1) parser and lexer for the YAML format which drives the majority of functionality. The lexer will deconstruct a YAML string/file into parsable tokens which will allow the deserialisation of YAML data into native PHP types and vice versa. As noted in the comments, JSON is a near compatible subset of YAML however this proposal does not currently intend supporting JSON as this already has excellent support via Zend_Json and ext/json.

It is expected to support almost all of the YAML grammar, with the notable exclusion of Unicode support. This will be omitted from initial versions but will be implemented once the requirements for Unicode support (given PHP's lack of) is more thoroughly assessed. In addition, Zend_Yaml would defer to any YAML C extension support if detected. It is currently unknown whether a PHP implementation would have 100% interoperability with such an extension though differences would be relatively minor (if not negligible).

The interface to Zend_Yaml will be deliberately simple supporting the deserialisation of YAML from either a file resource or a native string. Also supporting will be the serialisation of PHP native types into the YAML format. The default return value will be in the form of an array. Similar to Zend_Json, an option to decode YAML into stdClasses will also be offered for consistency.

2. References

- [The YAML Specification 1.1](#)

- Ongoing implementation in subversion
- Reference implementation in pure Python

3. Component Requirements, Constraints, and Acceptance Criteria

- Zend_Yaml **must** implement the YAML 1.1 specification (excluding Unicode support until a later version).
- Zend_Yaml **must** support both string literal and file based loading.
- Zend_Yaml **must** be capable of parsing all non-Unicode examples in the YAML 1.1 specification.
- Zend_Yaml **should** be interoperable with the reference implementations for Python, Ruby and Perl.
- Zend_Yaml **may not** be fully interoperable with the Syck C library or libyaml. Differences (if any) will be kept in line with other YAML parsers implemented in interpreted programming languages.

4. Dependencies on Other Framework Components

Zend_Yaml will have few dependencies on other Zend Framework classes. The only one currently noted is of course the base Exception class.

- Zend_Exception

5. Theory of Operation

Instantiate Zend_Yaml object, pass in a string or file resource representing a YAML stream, call the decode method to deserialise the YAML into a PHP array of values. If a YAML C extension is detected this may be used to offer a performance boost.

6. Milestones / Tasks

- Milestone 1: Implement basic YAML loading without Unicode support
- Milestone 2: Unit Tests and class refactoring
- Milestone 3: Debugging and Use Case testing
- Milestone 4: Verification of all non-Unicode specification examples being parsable by Zend_Yaml
- Milestone 5: Documentation
- Milestone 6: Add Unicode support

7. Class Index

- Zend_Yaml
- Zend_Yaml_Lexer
- Zend_Yaml_Parser
- Zend_Yaml-Token
- Zend_Yaml_SimpleKey
- Zend_Yaml_Buffer
- Zend_Yaml_Mark
- Zend_Yaml_Exception

This is not an exhaustive list. Also included will be subclasses of Zend_Yaml-Token representing all valid lexical tokens.

8. Use Cases

UC-01

Load a YAML string directly.

```
require_once 'Zend/Yaml.php';
$book = <<<EOS
name: Ulysses
author: James Joyce
category: [fiction, ireland]
isbn: 10
EOS;
$yaml = new Zend_Yaml;
echo $yaml->decode($book);
```

```
array(4) {
  ["name"]=>
  string(7) "Ulysses"
  ["author"]=>
  string(11) "James Joyce"
  ["category"]=>
  array(2) {
    [0]=>
    string(7) "fiction"
    [1]=>
    string(7) "ireland"
  }
  ["isbn"]=>
  string(10) "1555460216"
}
```

A string parameter can of course come from `file_get_contents()`, however file streaming using `fopen()` and `fread()` is also useable.

UC-02

Load a YAML file streamed from an `fopen()` resource.

```
require_once 'Zend/Yaml.php';
$fp = fopen('./data/ulysses.yml', 'rb');
$yaml = new Zend_Yaml;
echo $yaml->decode($fp);
```

9. Class Skeletons

Adding class skeletons will occur some time in the near future. You can check the current code from the subversion address in the reference list to see what the classes currently look like if required.

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