

Reorganization of documentation - Mickael Perraud & Thomas Weidner

<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: Reorganization of documentation Component Proposal

Proposed Component Name	Reorganization of documentation
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Reorganization of documentation
Proposers	Mickael Perraud Thomas Weidner
Zend Liaison	TBD
Revision	0.1 - 7 January 2009: Initial Draft. 0.2 - 17 January 2009: Add Zend_Tool provider 0.3 - 17 March 2009: of the conversation between Thomas Weidner , Wil Sinclair , Matthew Weier O'Phinney and Mickael Perraud on IRC 0.4 - 22 March 2009: add informations about a new translator tool 0.5 - 25 April 2009: Added documentation standard (wiki revision: 33)

Table of Contents

1. Overview
 2. References
 3. Component Requirements, Constraints, and Acceptance Criteria
 4. Dependencies on Other Framework Components
 5. Theory of Operation
- Simplification without any BC
 - Facilitate translations
 - Having the same manual.xml.in for all languages
 - Cut files by sect2 instead of sect1
 - Put all 'xinclude' in manual.xml.in
 - Creating different snippets
 - Rewrite building process as Zend_Tool provider
 - Sharing examples
 - Documentation standard
 - Line length
 - Intendation
 - Programmlisting
 - Seperation between tags
 - Classes
 - Variables
 - Methods
 - Constants
 - Filenames and Paths
 - Commmands
 - Code
 - Title tag
 - Ending spaces
 - Multiple new lines
 - Empty tags
 - No tabs
 - Line endings
 - XML Tags

Short tags
To go further
Extras documentation
Quickstart translation
Enhance HTML rendering
Adding index
Modify MANUAL TRANSLATION STATUS (<http://framework.zend.com/manual/status>)
Set up a documentation review
Testing examples of the documentation
Unify Migration chapters
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

One main goal => simplification:

- simplification of management
- simplification of translation
- simplification of rendering

2. References

- Official documentation
- Enhancement of CHM rendering:
 - <http://framework.zend.com/issues/browse/ZF-2454>
 - <http://mikaelkael.fr/Zend-Framework-CHM-Compilation.html>
- Creation of user's branch: http://framework.zend.com/code/browse/Zend_Framework/standard/branches/user/mikaelkael/
- Associated issues in IT:
 - ZF-2454: Integrate CHM compiled Documentation within the Homepage
 - ZF-3031: Change HTML <title> depending on page content
 - ZF-3774: Add the date/time of the last successfull docu sync
 - ZF-4968: Add php.net style comments to <http://framework.zend.com/manual/en/>
 - ZF-5355: Zend Framework documentation syntax highlighting

3. Component Requirements, Constraints, and Acceptance Criteria

...

4. Dependencies on Other Framework Components

- <http://framework.zend.com/wiki/display/ZFDEV/Translator++Manual+Compilation+Instructions>
- FOP 0.95.0
- PHP

5. Theory of Operation

Simplification without any BC

This part is fully compatible with actual system without any BC. Goals are:

- centralized for all languages: figures, compilation process
- simplification of the translation and especially of the update
- enhance html rendering with syntax highlighting
- "create" pdf rendering with syntax highlighting

Centralized figures and compilation process:

- creation of a directory: "/trunk/documentation/build"
- creation of a directory: "/trunk/documentation/build/figures" for all figures, this could be modify by using "/trunk/documentation/manual/en/figures" for committing access if you need, Zend_Tool provider can verify if the images exist in language subdirectory and can take them.
- rewrite actual Makefile as Zend_Tool provider
- adding some PHP process
- creation html.xsl.in and pdf.xsl.in (based on actual ones)

Enhance CHM rendering:

- I already explain some parts in <http://mikaelkael.fr/index.php?pages/Zend-Framework-CHM-Compilation> even if it's not exactly what I have rewrite

Creation of pdf rendering:

- creation of the associated stylesheet to not fork docbook-xsl
- using of same images as CHM rendering

Maintain actual system with Zend_Tool provider:

- "zf clean manual"
- "zf check manual" => English doc
- "zf check manual fr" => French doc
- "zf build manual" => create HTML English doc
- "zf build manual html fr" => create HTML French doc
- "zf build manual chm" => create CHM English doc (need Html Help Workshoop, could be set ENVIRONMENT variable)
- "zf build manual chm fr" => create CHM French doc
- "zf build manual pdf-fop" => create PDF English doc with fop
- "zf build manual pdf-fop fr" => create PDF French doc with fop

Needed (in addition to <http://framework.zend.com/wiki/display/ZFDEV/Translator+-+Manual+Compilation+Instructions>):

- FOP 0.95.0
- PHP

Facilitate translations

Having the same manual.xml.in for all languages

This could be realized by using xml entities. See <http://framework.zend.com/svn/framework/standard/branches/user/mikaelkael/manual/manual.xml.in> and <http://framework.zend.com/svn/framework/standard/branches/user/mikaelkael/manual/en/language-defs.ent> . We have to create language-defs.ent for each language. The manual.xml.in contains the fall back to English documentation.

Cut files by sect2 instead of sect1

Some actual files are very long to translate (Zend_Db_Adapter), divisions will facilitate the translation and especially the update. In the same idea,

as mentioned in the comments of this page, it would be good to limit the size (in number of words for example) of a generated page (in HTML or CHM format).

Put all 'xinclude' in manual.xml.in

To be sure that all languages will contain all files, we have to limit the using of 'xinclude' in /module_specs/* or /ref/* files. This could be easier if we apply the precedent paragraph (cutting).

Creating different snippets

My main example is 'requirements.xml'. We have to create snippets for table titles and headers, and for the word 'Hard' and 'Soft'. With this, the file 'requirements.xml' don't need to be translate (or update) each time.

Rewrite building process as Zend_Tool provider

See below

Sharing examples

Example updates are the most important documentation updates. The main reason we don't share them at this point is translation of the comments.

The first step is to link examples, this must be done by adding an id to all programlisting tags. After there is two possibilities, translations of the comments or not.

In English, we'll have for example:

```
<programlisting role="php"
id="zend.acl.introduction.role_registry.example1.code"><![CDATA[
$acl = new Zend_Acl();

//#1# Add groups to the Role registry using Zend_Acl_Role
//#2# Guest does not inherit access controls
$roleGuest = new Zend_Acl_Role('guest');
$acl->addRole($roleGuest);

//#4# Staff inherits from guest
$acl->addRole(new Zend_Acl_Role('staff'), $roleGuest);

/*#10#
Alternatively, the above could be written:
$acl->addRole(new Zend_Acl_Role('staff'), 'guest');
*/

//#5# Editor inherits from staff
$acl->addRole(new Zend_Acl_Role('editor'), 'staff');

//#8# Administrator does not inherit access controls
$acl->addRole(new Zend_Acl_Role('administrator'));
]]]><![CDATA[>
</programlisting>
```

In translated files, we'll just have:


```
<sect1>
  <sect2>
  </sect2>
</sect1>
```

It is not allowed to have multiple tags within one line

```
// NOT ALLOWED
<sect1><sect2>
</sect2></sect1>
```

Programmlisting

Beginning programmlisting must apply role of php, cdata and starting intendation

```
<programmlisting role="php"><![CDATA[
```

Programmlistings must not add linebreaks or whitespaces at the beginning or the end, as they would then be rendered.

```
// NOT ALLOWED, EMPTY SPACES AFTER CDATA
  <programmlisting role="php"><![CDATA[
$render = xxx
```

Ending CDATA Tag and programmlisting at the same line

```
  <programmlisting role="php"><![CDATA[
$render = xxx
]]]><![CDATA[></programmlisting>

  <para>
```

Seperation between tags

Tags at the same level must be seperated by a empty line to improve readability

```
<sect1>
  <sect2>
    <para>
      xxx
    </para>

    <para>
      xxxx
    </para>
  </sect2>
</sect1>
```

Classes

The tag classname must be used for all Zend Framework classes. The code tag is not allowed. This rule has only to be applied if the classname stands alone. Other content is not allowed within this tag.

```
<para>
  Text text <classname>Zend_Class</classname> text text
</para>
```

Variables

Variables must be wrapped in the varname tag. Variables must be written with \$ otherwise they are not recognised as variables. Other content is not allowed within this tag.

```
<para>
  Text text <varname>${variable}</varname> text text
</para>
```

Methods

Methods must be wrapped in the methodname tag. Methods must be written with () otherwise they are not recognised as methods. Other content is not allowed within this tag.

```
<para>
  Text text <methodname>getOptions()</methodname> text text
</para>
```

Constants

Constants must be wrapped in the constant tag. Constants must be written UPPERCASE otherwise they are not recognised as constants. Other content is not allowed within this tag.

```
<para>
  Text text <constant>MY_CONSTANT</constant> text text
</para>
```

Filename and Paths

File names and paths must be wrapped in the filename tag. They must include either a "." or "/" char. Other content is not allowed within this tag.

```
<para>
  Text text <filename>/path/to/example</filename> text text
</para>

<para>
  Text text <filename>example.php</filename> text text
</para>
```

Commands

Commands and program calls must be wrapped in the command tag. Commands must include a whitespace otherwise they are not recognised as variables. Other content is not allowed within this tag.

```
<para>
  Text text <command>zf param1 param2</command> text text
</para>
```

Code

The usage of the code tag is not allowed in favor of separation to filename, varname and constant.

```
// NOT ALLOWED
<para>
  Text text <code>xxxxx</code> text text
</para>
```

Title tag

The title is not allowed to hold any tags

```
// NOT ALLOWED
<sect1>
  <title>title <code>xxx</code> xxx</title>
</sect1>
```

Ending spaces

Ending spaces are not allowed

```
// NOT ALLOWED  
<sect1>WHITESPACE
```

Multiple new lines

Multiple new lines are not allowed

```
// NOT ALLOWED  
<para>  
  
    text text  
</para>
```

```
// NOT ALLOWED  
<para>  
    text text  
</para>  
  
<para>  
    text text  
</para>
```

Empty tags

Empty tags (para, example and others) are not allowed. They must contain text or underlying tags.

```
// NOT ALLOWED  
<para>  
</para>
```

No tabs

The usage of TABS is not allowed. Instead indentation of 4 spaces must be used.

Line endings

Only Unix lineendings are allowed. Other line endings like Windows or Mac are must not be used.

XML Tags

Each manual file has to include the following xml declarations at file start

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Reviewed: no -->
```

XML Files from translated languages must also include a revision tag which must notify the revision of the english file this translation is based of.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- EN-Revision: 14978 -->
<!-- Reviewed: no -->
```

Short tags

Short tags must not be used. Wether in description nor in code.

Note that there already exists a tool which checks parts of this standard. You can find it under: <http://mikaelkael.dyndns.org/checker>

To go further

Extras documentation

After establishing rules (with this proposal) on standard manual, we just have to apply same rules in extras documentation.

Quickstart translation

Actual quickstart (<http://framework.zend.com/docs/quickstart>) could be translate in Docbook format and put in <http://framework.zend.com/svn/framework/standard/trunk/documentation/manual/en/quickstart> or in <http://framework.zend.com/svn/framework/standard/trunk/documentation/quickstart/en>

Enhance HTML rendering

For the moment, table of contents ar only rendered for level 1 section (<http://framework.zend.com/manual/en/zend.acl.html> has a toc but not <http://framework.zend.com/manual/en/zend.acl.refining.html>). We could add TOC for all rendered file by using <http://docbook.sourceforge.net/release/xsl/current/doc/html/generate.section.toc.level.html>

Adding index

This part is a big one. An automatic index could be rendered by adding index tag inside all the actual docuementation. For example, to add an index 'Zend_Acl' in Zend_Acl-Advanced.xml, we have to replace:

```
<sect2 id="zend.acl.advanced.storing">

  <title>Storing ACL Data for Persistence</title>

  <para>
    Zend_Acl was designed in such a way that it does not require any particular
    backend technology such...
```

by:

```
<sect2 id="zend.acl.advanced.storing">

  <title>Storing ACL Data for Persistence</title>

  <para>
    <index>Zend_Acl</index> was designed in such a way that it does not require any
    particular backend technology such...
```

The job of determination of the words (or sentences) to index must be do in English and can change in translated files.

Modify MANUAL TRANSLATION STATUS (<http://framework.zend.com/manual/status>)

This page doesn't really show if a translation is outdated or not. If I take SVN 13906 to SVN 13930, these commits are just to modify tabs to spaces. Actual manual status is not able to see difference between real changes or 'esthetic' changes. Since a long time, some translators (French, Japanese or Deutsch) use a revision tag at the top of their translations to differentiate real and 'esthetic' changes:

```
<!-- EN-Revision: 13913 -->
```

Here is a link to test a new translator tool: <http://mikaelkael.dyndns.org/checkzf>. This tool can:

- check if you use the revision tag and takes it in this case, otherwise it takes the last revision change (like today in <http://framework.zend.com/manual/status>)
- show you the out-of-date and the non translated files, instead of showing you all the files.
- can retrieve for an out-of-date file the difference of the English file between the last translated revision and the last revision of English file, so if they are multiples revisions of the English file, they are all compile (it take into account reorganization of the SVN in revision 9506).
- can show you the last building check and associated output.
- can show you difference of number of tags between English and translated file. This is available for multiple tags: 'sectX', 'para', example'...

Set up a documentation review

Because many of us are no native English, it would be great to add a documentation review. This could be simply do by adding a special tag at the top of the file:

```
<!-- Reviewed: yes -->
```

When a new part is added, the tag is setting to no. You will find below a script that can render reviewed page based on this tag. This could be

used of course for English files but also for translated ones.

Testing examples of the documentation

At least, we should test for parse error. Then we could link examples between them with special comments which would be cleared at the time of rendering. For example with Zend_Db, the first with id 'zend.db.adapter.connecting.constructor.example' that describe the connection could be call by the second one:

```
<example id="zend.db.adapter.select.fetchall.example">
  <title>Using fetchAll()</title>
  <programlisting
role="php"><![CDATA[#Need:zend.db.adapter.connecting.constructor.example;...;...
$sql = 'SELECT * FROM bugs WHERE bug_id = ?';

$result = $db->fetchAll($sql, 2);
]]]><![CDATA[>
  </programlisting>
</example>
```

Unify Migration chapters

Actually we have a migration chapter for each component. This makes it very difficult when you want to migrate to a new release as the user must search every single component if there are migration notes.

To make things easier we should add a single migration chapter where all migration notes from all components are collected. So the user has one place where all migration notes are displayed. Within the component chapter a link can be places to the migration chapter. This makes live much easier for all people who want to update to a new release.

6. Milestones / Tasks

...

7. Class Index

...

8. Use Cases

UC-01

... (see good use cases book)

9. Class Skeletons

```

<?php
/**
 * Zend Framework
 *
 * LICENSE
 *
 * This source file is subject to the new BSD license that is bundled
 * with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://framework.zend.com/license/new-bsd
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@zend.com so we can send you a copy immediately.
 *
 * @category    Zend
 * @package     Zend_Tool
 * @copyright   Copyright (c) 2005-2008 Zend Technologies USA Inc.
 * (http://www.zend.com)
 * @license     http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id$
 */

/**
 * @see Zend_Tool_Framework_Provider_Interface
 */
require_once 'Zend/Tool/Framework/Provider/Interface.php';

/**
 * @see Zend_Tool_Framework_Client_Registry
 */
require_once 'Zend/Tool/Framework/Client/Registry.php';

class Zend_Tool_Framework_System_Provider_Manual implements
Zend_Tool_Framework_Provider_Interface
{
    /**
     * Docbook DTD
     * could be locally define with environment variable DOCBOOK_DTD
     * @var string
     */
    private $_docbookDtd = 'http://framework.zend.com/docbook/xml/4.5/docbookx.dtd';

    /**
     * Docbook Xsl for HTML rendering
     * could be locally define with environment variable DOCBOOK_XSL
     * @var string
     */
    private $_docbookXsl = 'http://framework.zend.com/docbook-xsl/html/chunk.xsl';

    /**
     * Docbook Xsl for HTML Help rendering
     * could be locally define with environment variable DOCBOOK_XSL_HTMLHELP
     * @var string
     */
    private $_docbookHtmlhelpXsl =
'http://framework.zend.com/docbook-xsl/htmlhelp/htmlhelp.xsl';

```

```

/**
 * Docbook Xsl for Pdf rendering
 * could be locally define with environment variable DOCBOOK_FO_XSL
 * @var string
 */
private $_docbookFoXsl = 'http://framework.zend.com/docbook-xsl/fo/docbook.xsl';

/**
 * Path to standard documentation
 * @var string
 */
private $_documentationPath = null;

/**
 * Path translated manual
 * @var string
 */
private $_workingDirectory = null;

/**
 * Path to standard documentation build
 * @var string
 */
private $_buildDirectory = null;

/**
 * Current language
 * @var string
 */
private $_lang = null;

/**
 * Response object
 * @var Zend_Tool_Framework_Client_Response
 */
private $_response = null;

/**
 * Information if current manual is valid
 * @var boolean
 */
private $_manualIsValid = false;

/**
 * Minor version of Zend Framework like (1.8.x)
 * @var string
 */
private $_zfVersion = null;

/**
 * Clean temporary elements and all rendering
 */
public function cleanAction()
{

}

/**
 * Check if a language is valid
 * @param string $lang Two letters representing language

```

```

    */
public function checkAction($lang = 'en')
{

    /**
     * Build the manual with given format
     * @param string $format    Could be 'html', 'htmlhelp', 'chm', 'pdf-fop',
'pdf-xep'
     * @param string $lang      Two letters representing language
     */
public function buildAction($format = 'html', $lang = 'en')
{

    /**
     * Prepare internal variable according to the environment
     * Create temporary directories
     * Retrieve Zend Framework version
     */
private function _setEnv()
{

    /**
     * Verify given language
     */
private function _checkLang($lang)
{

    /**
     * Create temporary manual
     */
private function _createTemporaryManual()
{

    /**
     * Insert additional informations in temporary manual before rendering
     */
private function _completeTemporaryManual()
{

    /**
     * Create temporary stylesheet using environment variable
     */
private function _createTemporaryStylesheet()
{

    /**
     * Check the current temporay manual
     * @return boolean
     */
private function _checkManual()
{

    /**
     * As we use internal libxml errors, this allow
     * to format errors before sending them to user
     *
     * @param unknown_type $xml
     * @return boolean
     */

```

```

private function _checkNoDomErrors($xml)
{

/**
 * Insert current Subversion revision of the language in the manual
 * @param string $manual Content of the manual
 */
private function _insertSvnRevision($manual)
{

/**
 * Insert current Zend Framework version in the manual
 * @param string $manual Content of the manual
 */
private function _insertZfVersion($manual)
{

/**
 * Insert image links otherwise they won't be included in CHM
 * @param string $fileName
 */
private function _insertImageLink($fileName)
{

/**
 * Highlight all HTML files
 * @param string $directory
 */
private function _highlightHtml($directory)
{

/**
 * Highlight a portion of code
 * @param string $text
 */
private function _highlightHtmlCode($text)
{

/**
 * Highlight a PDF file
 * @param string $fileName
 */
private function _highlightPdf($fileName)
{

/**
 * Highlight a portion of code
 * @param string $code
 * @return string
 */
private function _highlightBlock($code)
{

/**
 * Create a FO block from an highlighted string
 * @param string $code
 * @param DomDocument $mainDom
 * @return DomElement
 */

```

```
private static function _createBlockCode($code, $mainDom)
{}

/**
 * Build PDF file from FO file
 * @param string $fileName
 * @param string $renderer
 */
private function _createPdf($fileName, $renderer)
{}

/**
 * Create a CHM file from an HHP file
 * @param string $fileName
 */
private function _createChm($fileName)
{}

/**
 * Create a dir if not exists
 * @param string $path
 */
private function _createDir($path)
{}

/**
 * Remove a dir and all of his content
 * @param string $path
 */
private function _removeDir($path)
{}

/**
 * Remove a file if exists
 * @param string $path
 */
private function _removeFile($path)
```

```
}  
{}
```

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