Everyday Best Practices for PHP Development

Matthew Weier O'Phinney
Software Architect,
Zend Technologies

What you can do to make your life and your team's lives easier.

What will I gain?

- **Test your code instead of debug it**
- **Discover issues *before* you deploy your code**
- **Documentation of your code**
- **Documentation for your users**
- **Better communication with your team**
- **and, bottom line, maintainability.**

The single best practice...

TEST!

Test?

- **Unit test everything you can**
- **Learn how to do functional and integration testing**
- **Don't put it off**

Benefits of testing

- More testing == Less debugging
- Revise and exercise your object APIs *before* writing code
- Instant feedback when changes are made
- Repeatable
- Stabilize functionality
- Gain confidence in your application quality
- ...

Benefits of testing

- **Do you really need more reasons?**

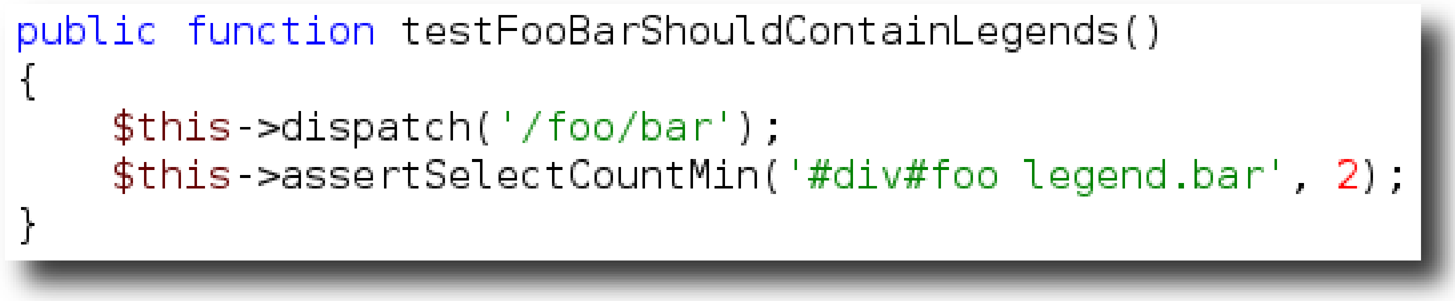
Test behaviors in your models

- **Example:**
  I *should* be able to fetch a user by email

```
public function testFetchUserShouldAllowFetchingByEmail()
{
    $id = $this->model->save(array(
        'username' => 'foo',
        'email'    => 'foo@email.com',
        'fullname' => 'Foo Bar',
        'password' => md5('foobar'),
    ));

    $user = $this->model->fetchUser('foo@email.com');
    $this->assertEquals($id, $user->id);
}
```

Extend testing to your applications

- **Given a request to /foo/bar, I should have at least 2 items matching the CSS selector “div#foo legend.bar”:**



```
public function testFooBarShouldContainLegends()
{
    $this->dispatch('/foo/bar');
    $this->assertSelectCountMin('#div#foo legend.bar', 2);
}
```

Get to know PHPUnit

- **http://phpunit.de/**

And another gem:

Use a Coding Standard

Why use coding standards?

- **Focus on code, not formatting**
- **Consistency**
- **Readability**
- **Collaboration**

Okay, I'll create one...

STOP!

Learn from others

- **The issues have already been debated to death.**
- **Use an established standard, and stick to it.**
  - Minimizes politics when choosing
  - Choose a standard compatible with the libraries or frameworks you use
  - Use the standard as a requirement for hiring or outsourcing.

What does a standard provide?

- **File, class, and variable naming conventions**
- **Code formatting conventions**

Some Zend Framework standards

- **Derived from PEAR standards (which were in turn derived from Horde standards)**

- **One class, one file**
- **Underscores in class names map to directory separators:**
  Zend_Controller_Action:
  Zend/Controller/Action.php

Some Zend Framework standards

Naming conventions:

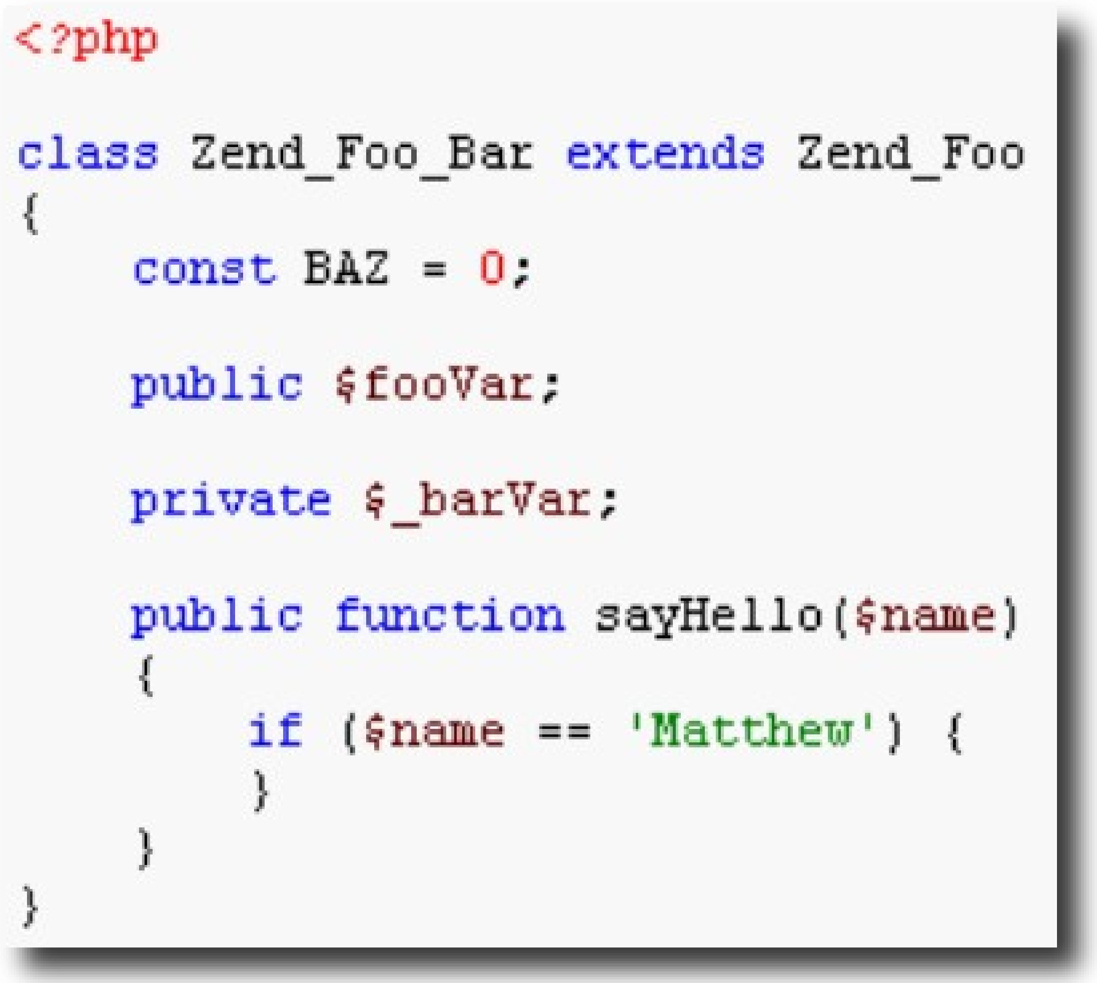
- **Class names are MixedCase**
- **Method names are camelCase**
- **Constants are ALL_CAPS**
- **Properties and variables are camelCase**
- **Private and protected members are _underscorePrefixed**

Some Zend Framework standards

Layout Conventions:

- **No closing ?> tag for files containing only code**
- **Indentation: spaces only, no tabs; 4 spaces per level of indentation**
- **One True Brace:**
  - Classes and methods place opening brace on following line at same indentation.
  - Logical structures place opening brace on same line.
  - All control structures use braces, period.
- **No shell style comments (#)**
- **Keep lines no more than 75-85 characters long**

Example



```
<?php

class Zend_Foo_Bar extends Zend_Foo
{
    const BAZ = 0;

    public $fooVar;

    private $_barVar;

    public function sayHello($name)
    {
        if ($name == 'Matthew') {
        }
    }
}
```

What else should you know?

Design Patterns

What are those?

- **Reusable ideas, not code**
- **Proven solutions to common design problems**
- **Better communication through shared vocabulary**

Some examples, please?

- **I need to be able to notify other objects when I execute a particular event: *Observer***
- **I need to be able to mutate the backend object to which I delegate: *Adapter***
- **I need to modify the output of an object: *Decorator***
- **I need to decorate my application output with general site content: *Two Step View***

Who uses design patterns?

- **Frameworks; Zend Framework is riddled with them**
- **You do, by using frameworks. :-)**

What next?

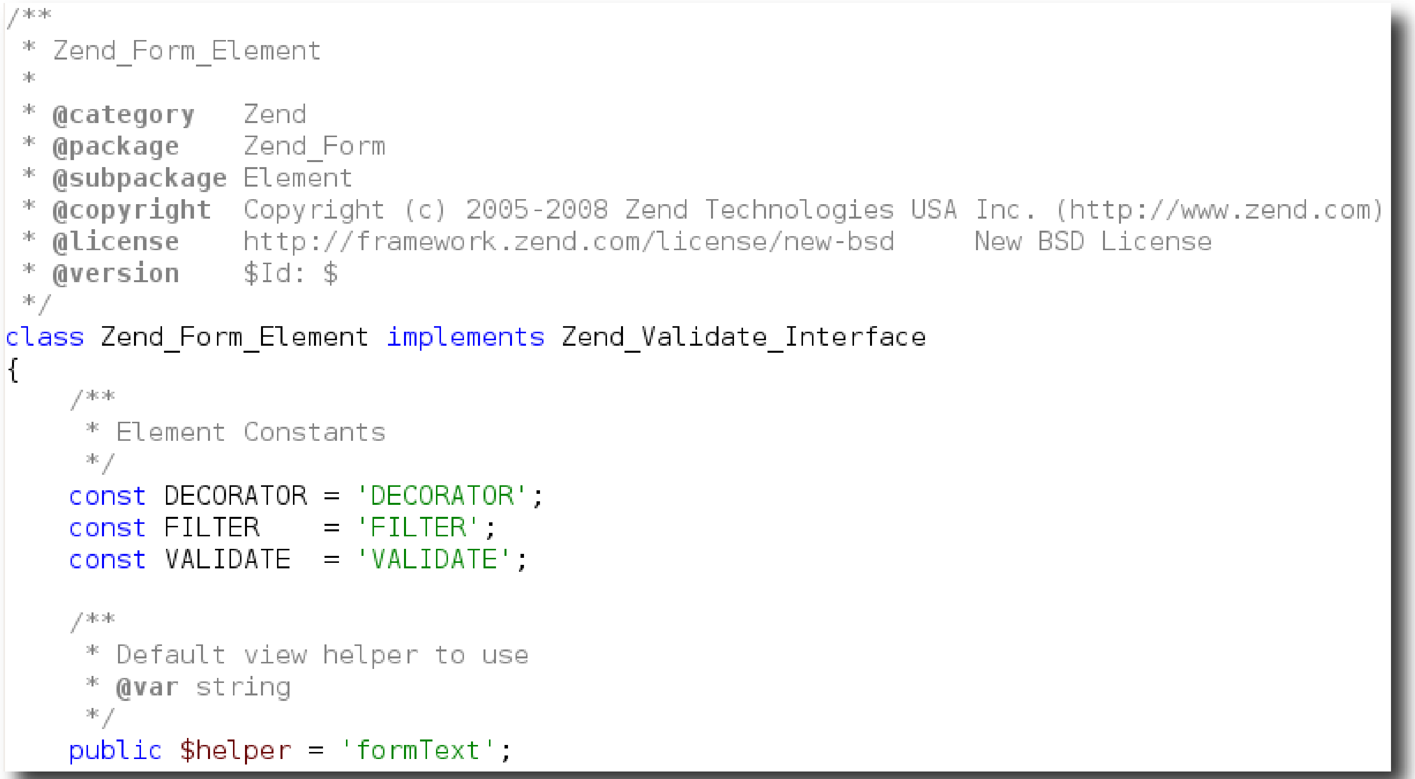
Documentation

But I don't have time!

- **You don't have time to code?**

API documentation is easy

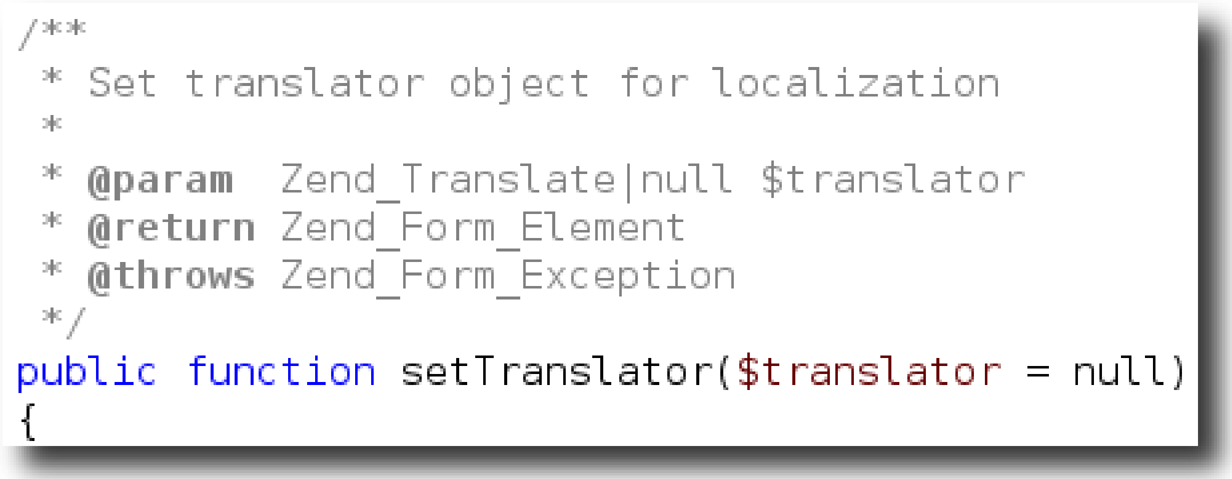
- **Simply prepend PHP docblocks to your methods and classes; your IDE will often do it for you:**



```
/**
 * Zend_Form_Element
 *
 * @category   Zend
 * @package    Zend_Form
 * @subpackage Element
 * @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: $
 */
class Zend_Form_Element implements Zend_Validate_Interface
{
    /**
     * Element Constants
     */
    const DECORATOR = 'DECORATOR';
    const FILTER    = 'FILTER';
    const VALIDATE  = 'VALIDATE';

    /**
     * Default view helper to use
     * @var string
     */
    public $helper = 'formText';
```

What can I document this way?

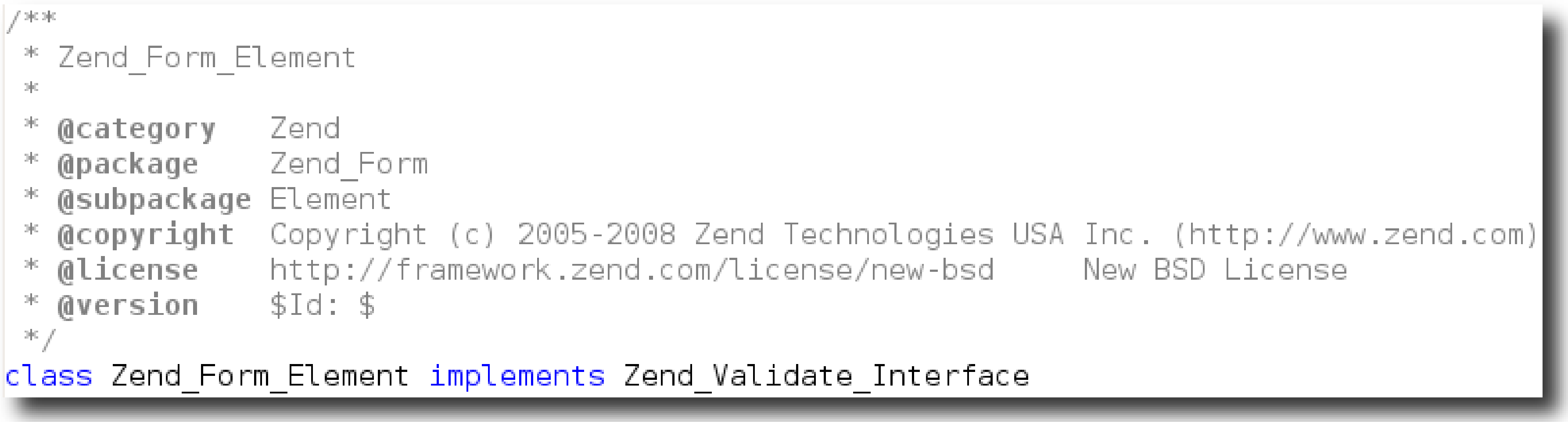
- **Classes, methods, class properties…**
- **Use annotation tags in source comments to provide context: @param, @return, @throws, @see, @todo**



```
/**
 * Set translator object for localization
 *
 * @param  Zend_Translate|null $translator
 * @return Zend_Form_Element
 * @throws Zend_Form_Exception
 */
public function setTranslator($translator = null)
{
```

Docblocks can organize code

- **Utilize @category, @package, @subpackage; phpDoc uses these to organize documentation.**
- **Prefix your classes; easier to browse, and easier to mix with other libraries.**

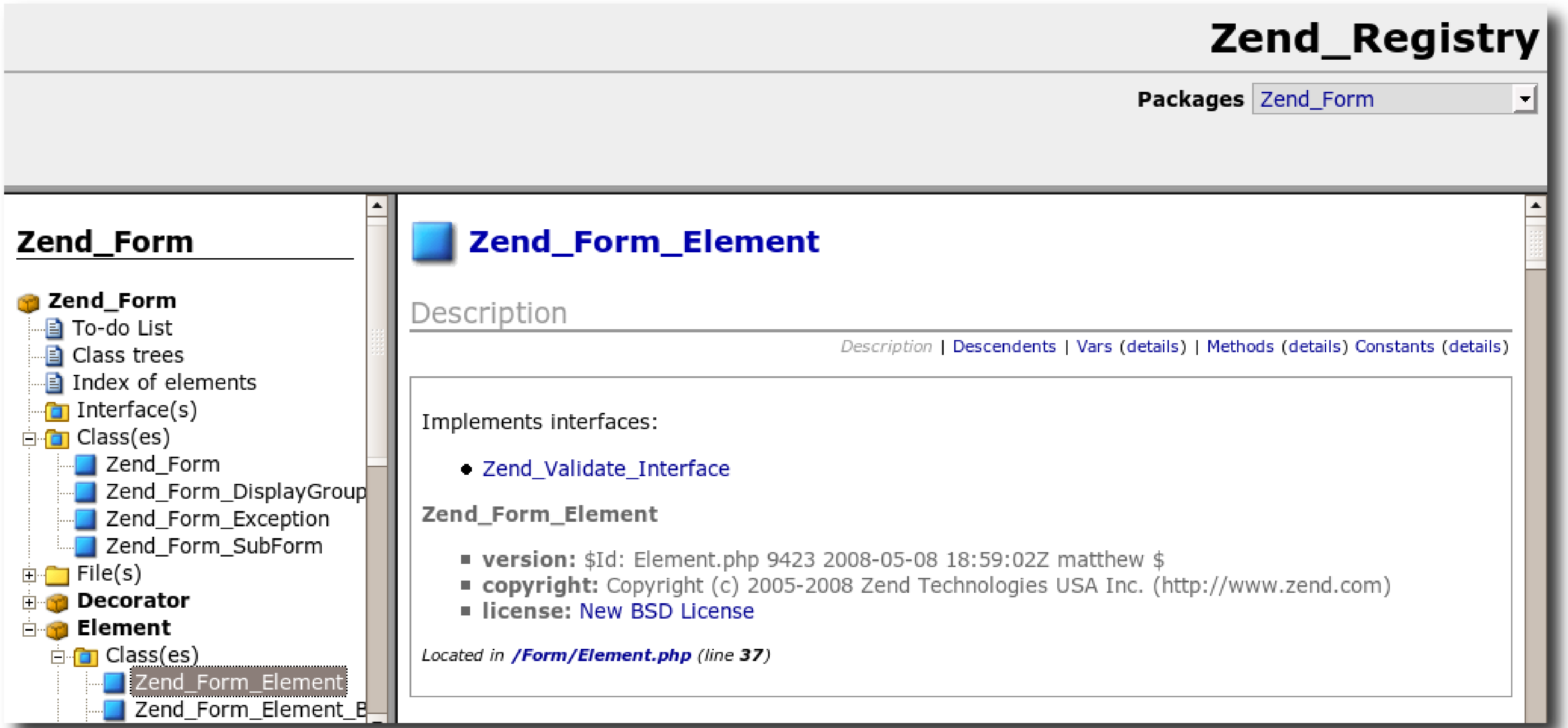


```
/**
 * Zend_Form_Element
 *
 * @category   Zend
 * @package    Zend_Form
 * @subpackage Element
 * @copyright  Copyright (c) 2005-2008 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license/new-bsd     New BSD License
 * @version    $Id: $
 */
class Zend_Form_Element implements Zend_Validate_Interface
```

Generate pretty API docs

- **phpDocumentor: http://phpdoc.org/**
- **Doxygen: http://www.stack.nl/~dimitri/doxygen/**

See?



Zend_Registry

Packages Zend_Form

Zend_Form

- Zend_Form
  - To-do List
  - Class trees
  - Index of elements
  - Interface(s)
  - Class(es)
    - Zend_Form
    - Zend_Form_DisplayGroup
    - Zend_Form_Exception
    - Zend_Form_SubForm
  - File(s)
  - Decorator
  - Element
    - Class(es)
      - Zend_Form_Element
      - Zend_Form_Element_B

Zend_Form_Element

Description

Description | Descendents | Vars (details) | Methods (details) Constants (details)

Implements interfaces:

- Zend_Validate_Interface

Zend_Form_Element

- **version:** $Id: Element.php 9423 2008-05-08 18:59:02Z matthew $
- **copyright:** Copyright (c) 2005-2008 Zend Technologies USA Inc. (http://www.zend.com)
- **license:** New BSD License

Located in ***/Form/Element.php*** *(line* ***37****)*

Be careful what you say...

- **Docblocks can go out of date. Be general, except when it comes to the parameters and return values.**
- **When in doubt, unit tests don't lie.**

IDEs like documentation, too

- **IDE's introspect DocBlocks to provide typehinting, return values, and method descriptions.**

So does Zend Framework!

- **Various Server classes utilize DocBlocks to provide hinting for parameter and return value types, as well as method descriptions**
  - Zend_XmlRpc_Server
  - Zend_Rest_Server
  - Zend_Json_Server (coming soon!)
  - Zend_Soap_Wsdl (coming soon!)
  - Zend_Tool (coming soon!)

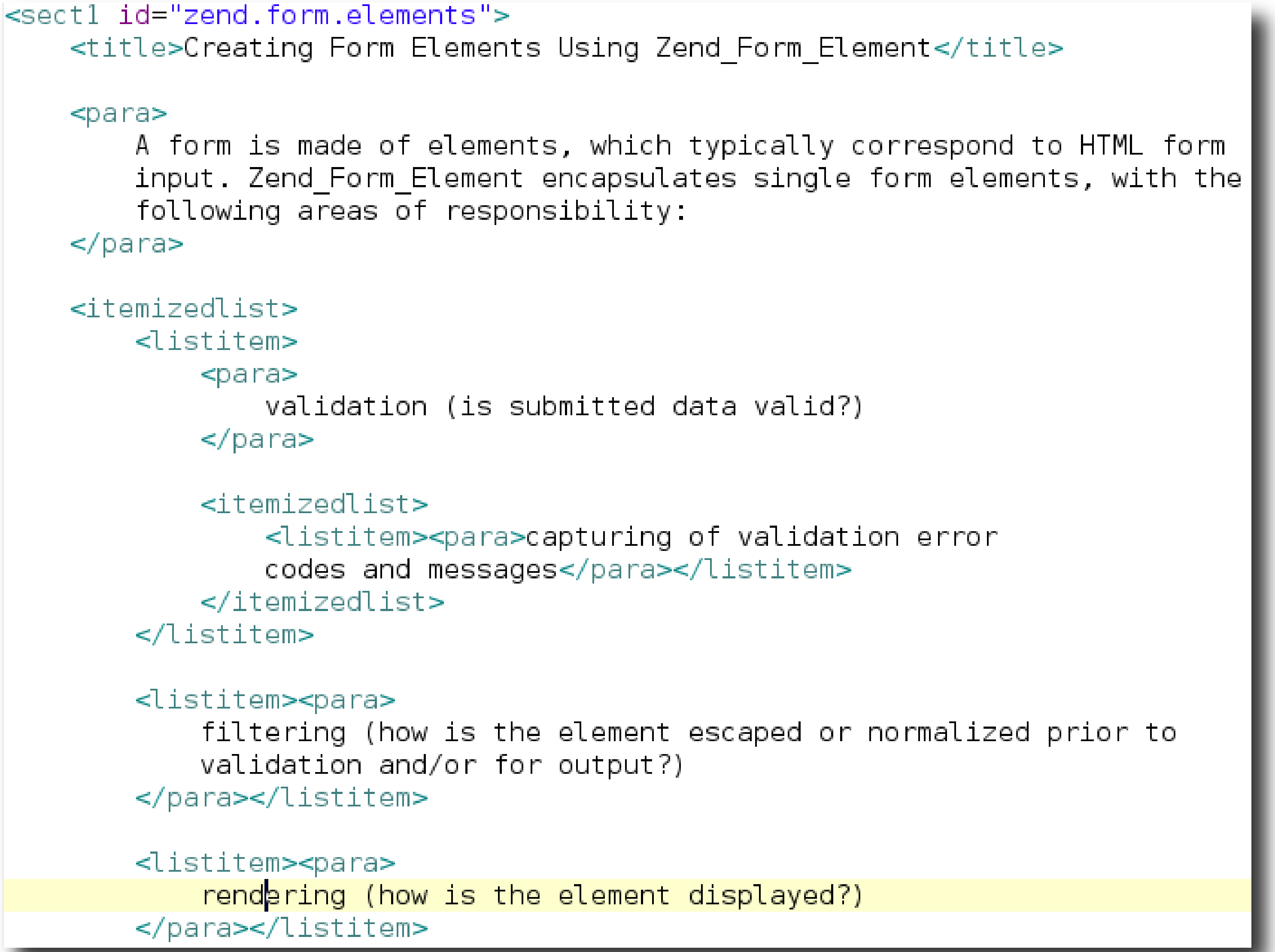
Don't forget your users!

- **End users like to know how to *use* your code and applications**
- **Give them a manual!**

XML is not a four-letter word

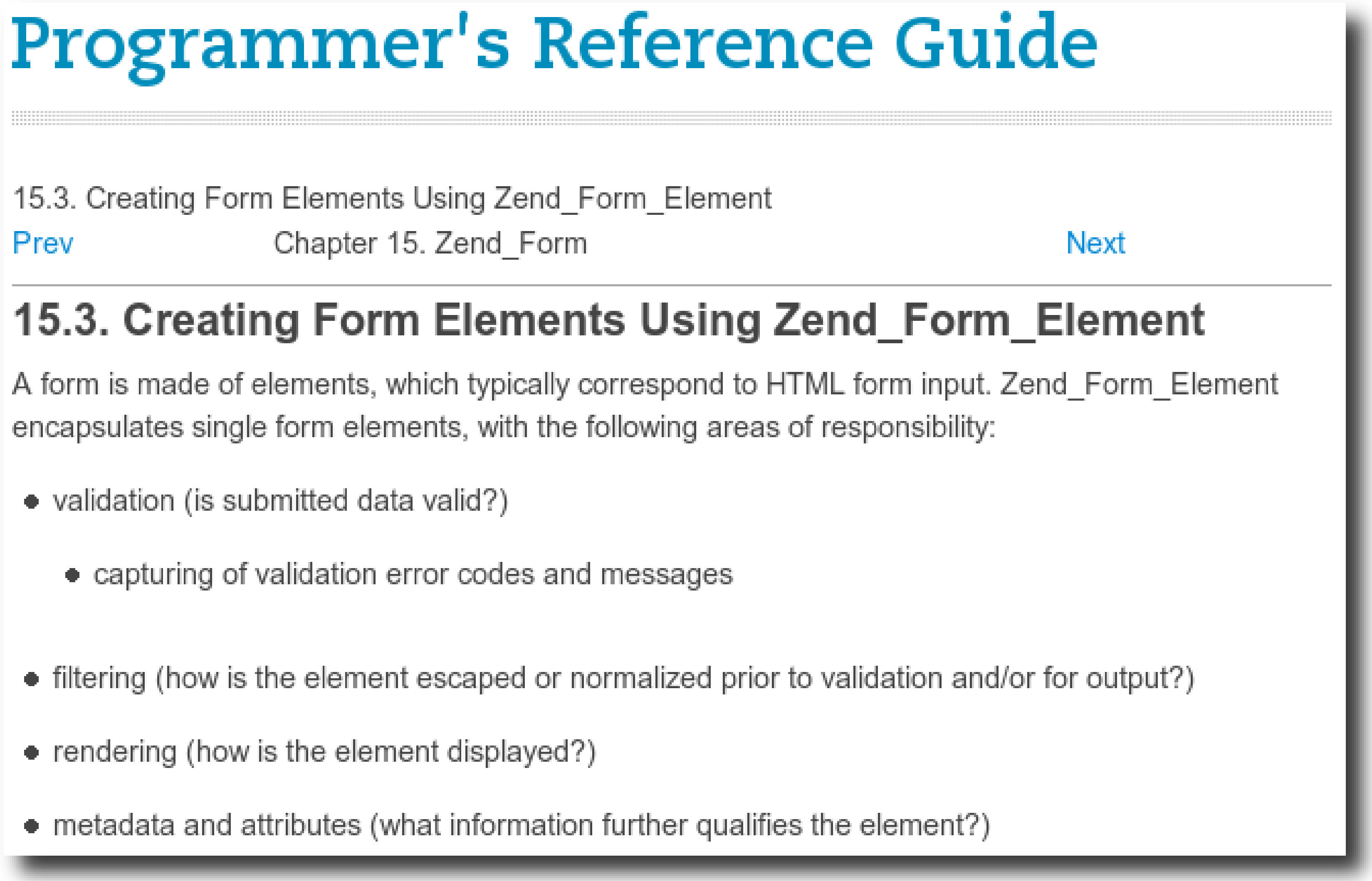
- DocBook is the most common format for open source documentation
- DocBook can be compiled to a variety of formats: HTML, Windows Help files (CHM), PDF, and more.
- Often used by book publishers (O'Reilly)
- It powers the PHP.net manual
- It powers Zend Framework's manual

DocBook is easy



```
<sect1 id="zend.form.elements">
    <title>Creating Form Elements Using Zend_Form_Element</title>

    <para>
        A form is made of elements, which typically correspond to HTML form
        input. Zend_Form_Element encapsulates single form elements, with the
        following areas of responsibility:
    </para>

    <itemizedlist>
        <listitem>
            <para>
                validation (is submitted data valid?)
            </para>

            <itemizedlist>
                <listitem><para>capturing of validation error
                codes and messages</para></listitem>
            </itemizedlist>
        </listitem>

        <listitem><para>
            filtering (how is the element escaped or normalized prior to
            validation and/or for output?)
        </para></listitem>

        <listitem><para>
            rendering (how is the element displayed?)
        </para></listitem>
```

One possible rendition:



Programmer's Reference Guide

15.3. Creating Form Elements Using Zend_Form_Element

Prev   Chapter 15. Zend_Form   Next

15.3. Creating Form Elements Using Zend_Form_Element

A form is made of elements, which typically correspond to HTML form input. Zend_Form_Element encapsulates single form elements, with the following areas of responsibility:

- validation (is submitted data valid?)
    - capturing of validation error codes and messages
- filtering (how is the element escaped or normalized prior to validation and/or for output?)
- rendering (how is the element displayed?)
- metadata and attributes (what information further qualifies the element?)

Don't forget to backup...

Source Control

Why do I need it?

- **How do I know if somebody did something?**
- **How do others know I did something?**
- **How do I get my updates from others?**
- **How do I push my updates out to others?**
- **Do we have the old version? What changed?**

What are my options?

- **Distributed Source Control:**
  Developers work on their own repositories and share changesets
  - Git
  - Darcs
  - Arch
- **Non-Distributed Source Control**
  Developers work on local checkouts, and check in to a central repository
  - Subversion

How do I use source control?

- **Perform local checkout**
- **Write code**
- **Record changes**
- **Check changes in to repository**
- **Check for repository updates**
- **Lather, rinse, repeat**

What do *you* use?

Subversion

- **Extensible and supported by excellent tools**
  - Write scripts to perform actions before and after checkins
- **Popular with many open source projects; integrate with them using svn:externals**
- **Easily move files between directories while preserving histories**
- **Simplified process of tagging and branching**
- **Transactions for when things go wrong**

Review

How do I make my life easier?

- **TEST!**

How do I make my life easier?

- **TEST!**
- **Use a coding standard**

How do I make my life easier?

- **TEST!**
- **Use a coding standard**
- **Learn and utilize design patterns**

How do I make my life easier?

- **TEST!**
- **Use a coding standard**
- **Learn and utilize design patterns**
- **Document my code**

How do I make my life easier?

- **TEST!**
- **Use a coding standard**
- **Learn and utilize design patterns**
- **Document my code**
- **Document my application**

How do I make my life easier?

- **TEST!**
- **Use a coding standard**
- **Learn and utilize design patterns**
- **Document my code**
- **Document my application**
- **Use source control**

So what are *you* waiting for?

Thanks for listening!