


Tutoriel Zend Studio for Eclipse

par Alain Sahli ([Tutoriels PHP de Alain Sahli](#)) ([Blog](#))

Date de publication : 21.04.2008

Ce tutoriel a pour but de vous présenter l'utilisation des nouvelles fonctionnalités de Zend Studio for Eclipse. Il ne présentera pas les anciennes fonctionnalités présentes dans les versions antérieures. Si vous n'avez jamais utilisé Zend Studio, je vous conseille de lire l'article  [Créer un projet avec Zend Studio \(ZDE\)](#), de Guillaume Rossolini.

- I - Introduction
- II - L'application
 - II-A - Structure
 - II-B - Aperçu
- III - Gérer un projet
 - III-A - Création d'un projet Zend Framework
 - III-B - La vue MVC d'Eclipse
 - III-C - Les tests d'abord (TDD) !
 - III-D - L'interface avec l'éditeur HTML WYSIWYG
 - III-E - Support du code Javascript
 - III-F - Génération de documentation
 - III-G - Débogage
 - III-G-1 - Débogage distant
 - III-G-2 - Débogage local
- IV - Conclusion
 - IV-A - Remerciements

I - Introduction

Zend a décidé de baser sa version 6 de Zend Studio sur l'excellent EDI Eclipse et de la nommer Zend Studio for Eclipse. Voici les principales nouveautés :



- Une vue MVC pour le Zend Framework.
- Intégration des tests unitaires avec PHPUnit.
- Une vue WYSIWYG pour la création de formulaires HTML / CSS.

Les raisons pour lesquelles Zend a décidé de basé la nouvelle version de son EDI sur Eclipse sont les suivantes (tiré du site zend.com/fr) :

- **La standardisation** - grâce à son support pour de multiples langages, on apprend un seul IDE pour tous ses développements. Cela simplifie la création d'un environnement parfait pour des applications multi-langages.
- **Extensibilité** - avec plus de 800 plug-ins disponibles, il est très facile d'ajouter les fonctionnalités dont on a besoin.
- **Une communauté Open Source active** - ce qui permet le développement rapide de nouvelles technologies.

Bien sûr, toutes les fonctionnalités présentes dans Zend Studio 5.5.0 le sont également dans cette nouvelle version.

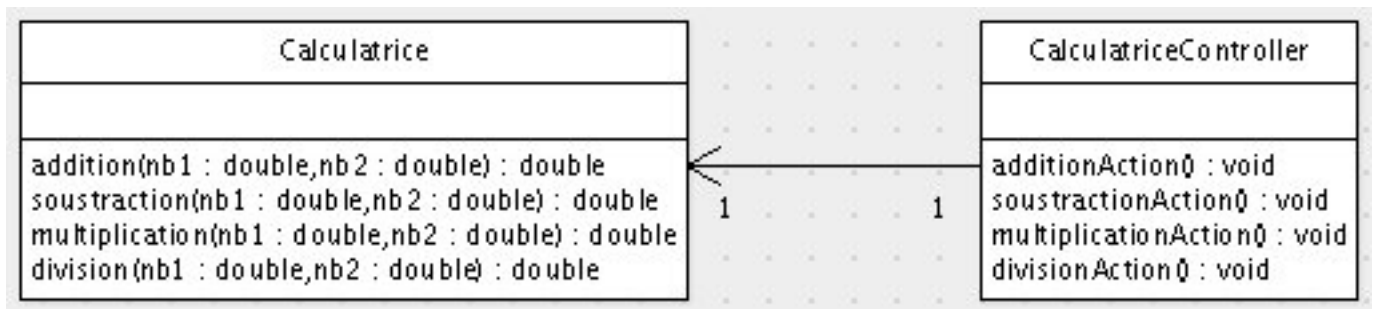
Afin de vous présenter les différentes fonctionnalités de cet EDI, je vais vous guider au travers d'un petit tutoriel qui aboutira sur la réalisation d'une calculatrice simple.

 *Pour de plus amples informations concernant l'environnement Eclipse je vous conseille de consulter la  **rubrique Eclipse** !*

II - L'application

II-A - Structure

Le but de ce tutoriel étant de vous présenter les nouvelles fonctionnalités de Zend Studio for Eclipse, je ne vais pas trop m'attarder sur l'application en elle-même. Nous allons donc développer une calculatrice simple permettant de faire des additions, soustraction, multiplication et divisions. Ci-dessous le schéma UML de l'API.

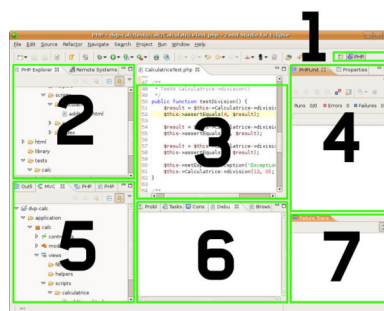


Structure de l'application

La structure est donc extrêmement simple, une classe de la couche « Modèle » et un contrôleur qui fera le lien entre la vue et le modèle. C'est très important de ne pas coder de parties « fonctionnelles » dans les contrôleurs pour deux raisons : premièrement, on ne respecte pas le design pattern MVC et, deuxièmement, cela complique considérablement les tests unitaires. Je dis ça en passant mais ce n'est bien sûr pas l'objet de ce tutoriel ;-)

II-B - Aperçu

Tout au long de cet article, je vais utiliser des termes pour décrire les différentes zones de l'éditeur. Ci-dessous vous retrouvez les différents termes:



Interface de Zend Studio for Eclipse

- 1 Sélecteur de perspective
- 2 Explorateur de fichiers
- 3 Zone principale
- 4 Vue PHPUnit
- 5 Vue MVC
- 6 Onglets d'informations (Debug, Console, Problem, etc...)
- 7 Traçage des erreurs (par rapport à PHPUnit)

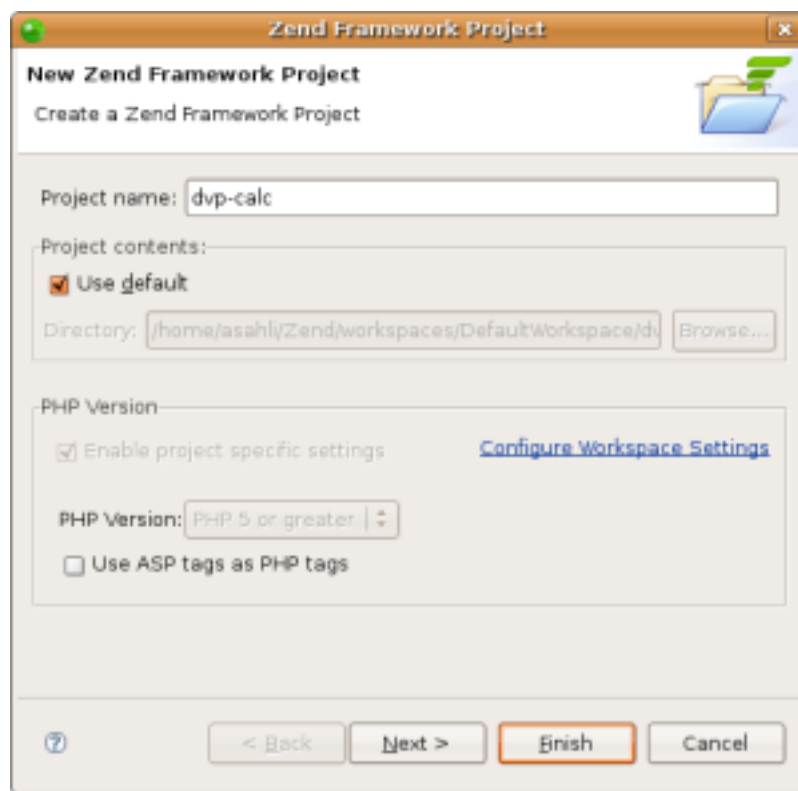
III - Gérer un projet

Ci-dessous, je vais vous décrire étape par étape le déroulement d'un projet avec Zend Studio for Eclipse.

III-A - Création d'un projet Zend Framework

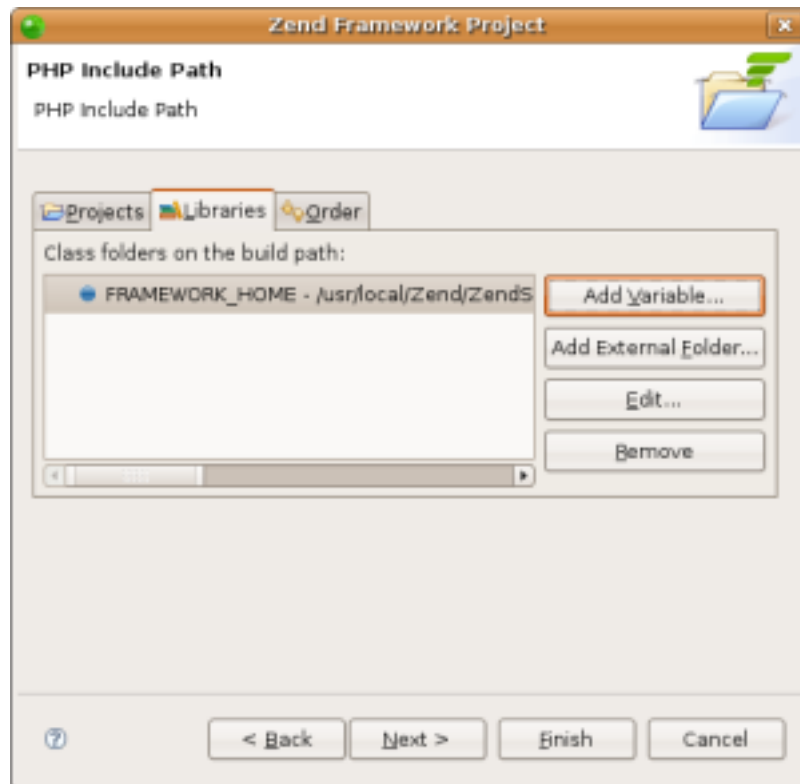
Une fois la modélisation de l'application terminée, on peut commencer par créer les différentes classes avec leurs méthodes vides ! Pour cela, nous allons créer un nouveau projet de type Zend Framework.

Tout d'abord lancer Zend Studio for Eclipse puis cliquer sur File - New - Zend Framework Project . Ensuite la fenêtre suivante apparaît :



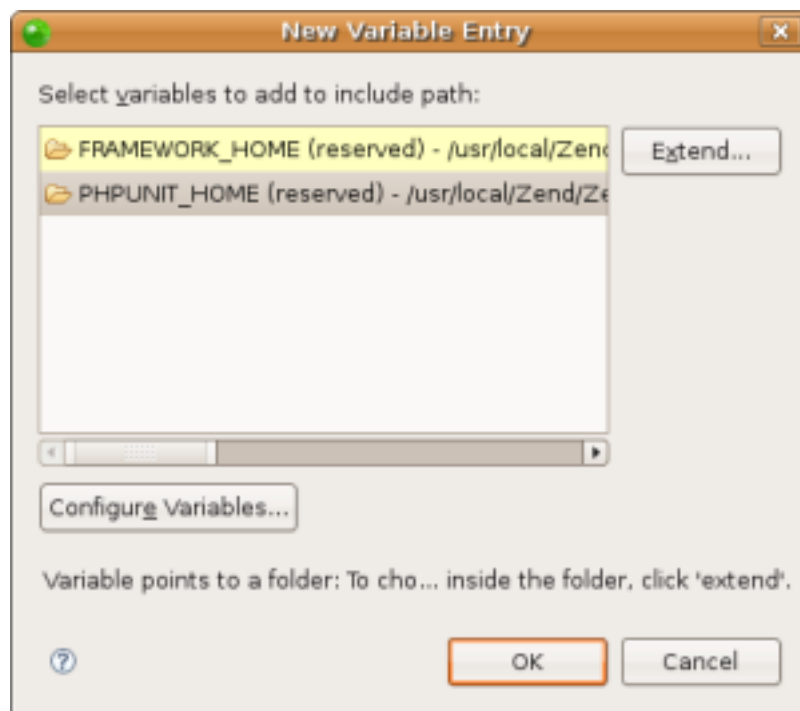
Dialogues pour la création d'un nouveau projet Zend Framework

Donnez un nom au projet (dvp-calc) puis soit vous sauvez le projet dans le répertoire par défaut ou alors vous pouvez spécifiez l'endroit où vous voulez le sauver en décochant « Use default ». Une fois ceci fait, cliquez sur suivant et vous allez arriver sur cette fenêtre :



Etape 2 - Création d'un projet Zend Framework

Cette fenêtre permet de charger des librairies externes et internes. Comme nous avons créé un nouveau projet Zend Framework celui-ci est déjà chargé ; cependant, pour notre projet nous aurons encore besoin de PHPUnit. Cette librairie fait également partie de Zend Studio, on peut donc cliquer sur « Add Variable ». La fenêtre ci-dessous apparaît :

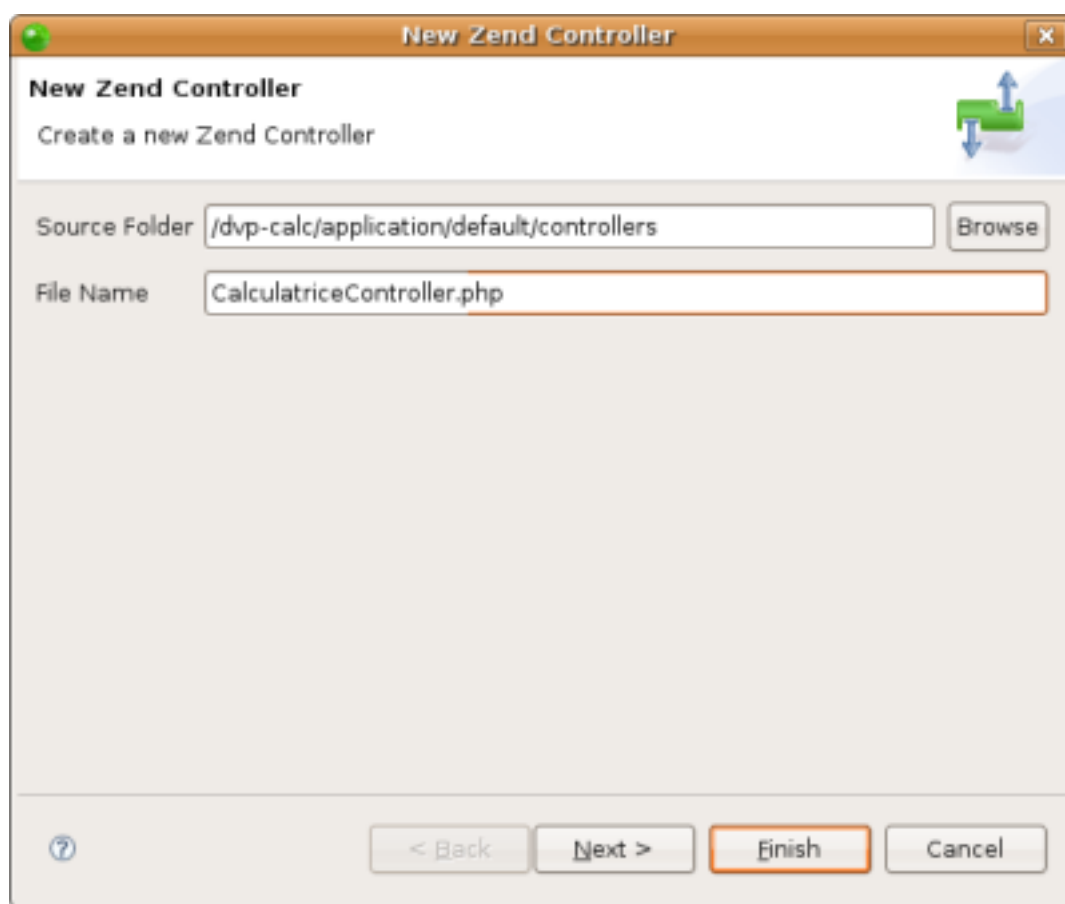


Sélection de la variable PHPUNIT_HOME

Cliquez sur PHPUNIT_HOME puis sur « OK ». L'initialisation du projet est maintenant faite, vous pouvez cliquer sur le bouton « Finish » ! Une fenêtre de dialogue va vous demander si vous voulez directement ouvrir la perspective PHP associée au projet, cliquez sur « OK ».

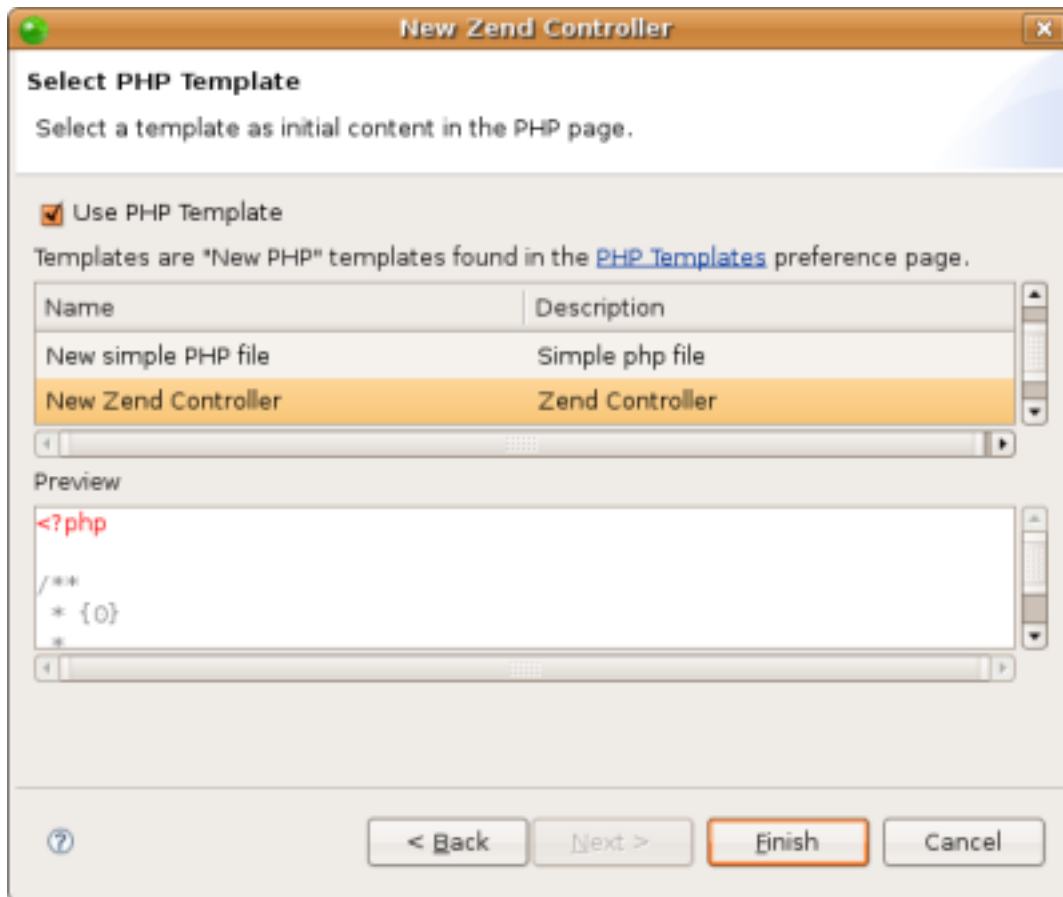
III-B - La vue MVC d'Eclipse

Il s'agit maintenant de créer notre classe de contrôleur et de modèle. Pour ceci nous allons utiliser la vue MVC. Faites un clic droit sur le module « Default » puis New - Zend Controller. Cela va vous ouvrir une nouvelle fenêtre pour la création du contrôleur, la voici :



Création d'un nouveau contrôleur

Donnez-lui le nom que nous avons défini dans le diagramme UML, c'est-à-dire « CalculatriceController » puis cliquez sur « Next ». Une deuxième fenêtre apparaît qui vous permet de choisir un template de classe.



Sélection d'une template pour le contrôleur

Ci-dessous, vous avez les différents templates de classe :

```

New PHP file - HTML Frameset
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=${encoding}">
  <title>Insert title here</title>
</head>
<frameset>
  <frame>
  <frame>
  <noframes>
  <body>
    <p>This page uses frames. The current browser you are using does not support frames.</p>
    <?php
    ${cursor}
  ?>
  </body>
  </noframes>
</frameset>
</html>

```

```

New Zend Controller
<?php

/**
 * {0}

```

New Zend Controller

```
*
* @author
* @version
*/

require_once 'Zend/Controller/Action.php';

class {0} extends Zend_Controller_Action
{
    /**
     * The default action - show the home page
     */
    public function indexAction()
    {
        // TODO Auto-generated {0}::indexAction() default action
    }
    ${cursor}
}
```

New Zend Model

```
<?php

/**
 * {0}
 *
 * @author ${user}
 * @version
 */

require_once 'Zend/Db/Table/Abstract.php';

class {0} extends Zend_Db_Table_Abstract
{
    /**
     * The default table name
     */
    protected $_name = '{1}';
    ${cursor}
}
```

Sélectionnez la template « New Zend Controller » puis cliquez sur finish.

La démarche est très semblable pour la création de notre classe Calculatrice : il vous suffit de faire un clique droit sur le module « Default » puis New - Zend Model, de lui donner le nom Calculatrice et enfin de lui assigner un template vide (New simple PHP file).

Voilà, notre contrôleur et notre modèle sont créés ! Il nous faut maintenant créer les méthodes vides ! J'insiste sur le fait qu'elles doivent être vides.

Classe CalculatriceController

```
public function additionAction(){
}
public function soustractionAction(){
}
public function multiplicationAction(){
}
public function divisionAction(){
}
```

Classe Calculatrice

```
public function addition($nb1, $nb2){
```

Classe Calculatrice

```

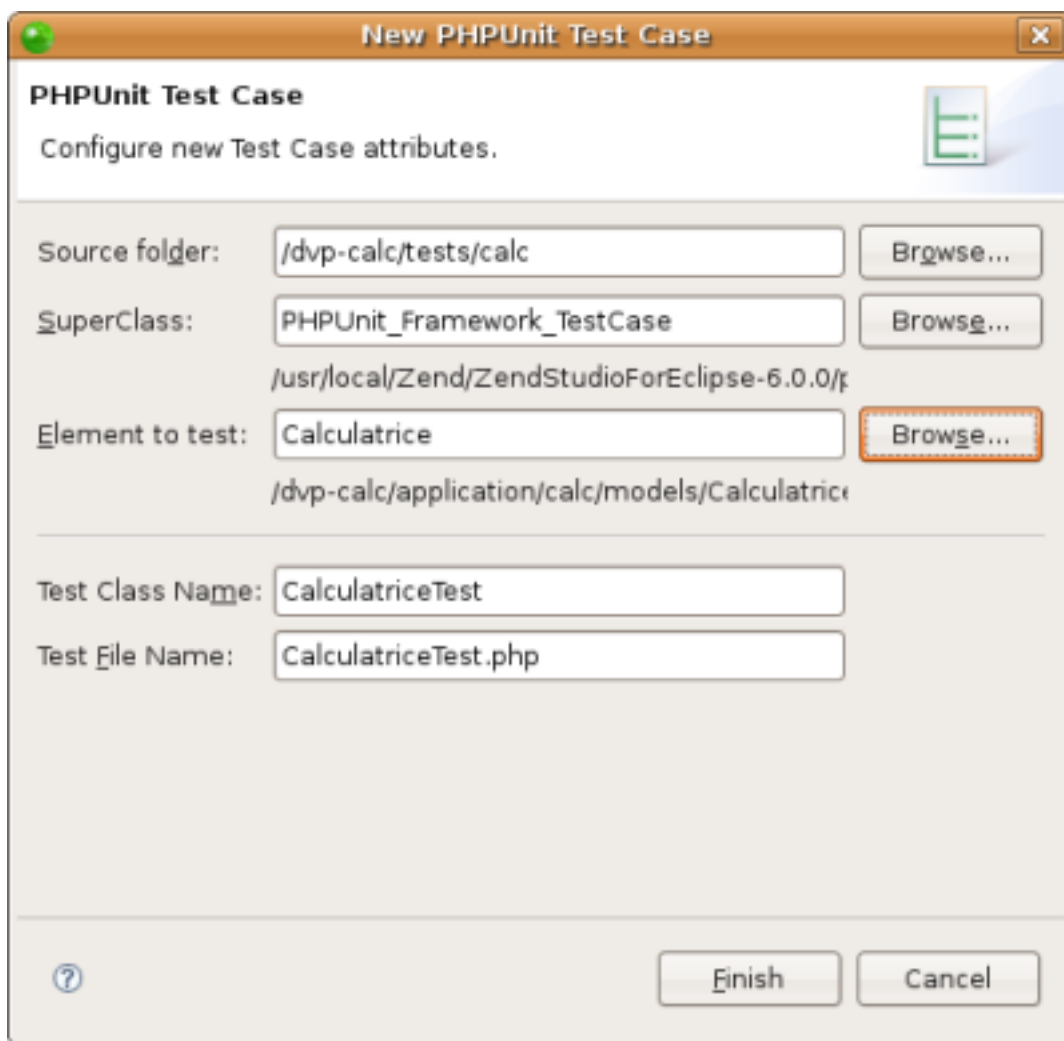
}
public function soustraction($nb1, $nb2){
}
public function multiplication($nb1, $nb2){
}
public function division($nb1, $nb2){
}

```

III-C - Les tests d'abord (TDD) !

Nous devons maintenant écrire une série de tests pour chaque méthode de la classe Calculatrice. Heureusement, Zend Studio for Eclipse possède une vue et des fonctionnalités pour nous faciliter la tâche !

Tout d'abord créez un dossier au même niveau que le dossier application dans lequel nous placerons tous nos tests. Puis, dans le dossier « tests », créez un dossier qui porte le même nom que notre module, c'est-à-dire « calc ». Une fois cette structure de dossiers en place, faites un clique droit sur le dossier « calc » puis New - PHPUnit test case. Dans la fenêtre qui s'ouvre, cliquez sur « Browse » et sélectionnez la classe Calculatrice :



Nouvelle classe de test PHPUnit

Puis cliquez sur finish. Une nouvelle classe nommée CalculatriceTest va apparaître dans le dossier « tests/calc ». Ouvrez cette classe et videz le contenu des méthodes testAddition, testSoustraction, testMultiplication et testDivision, vous pouvez également supprimer le constructeur, nous n'en aurons pas besoin. Votre classe doit maintenant ressembler à ceci :

Classe CalculatriceTest

```
<?php
require_once 'application/calc/models/Calculatrice.php';
require_once 'PHPUnit/Framework/Testcase.php';
/**
 * Calculatrice test case.
 */
class CalculatriceTest extends PHPUnit_Framework_TestCase {

    /**
     * @var Calculatrice
     */
    private $Calculatrice;

    /**
     * Prepares the environment before running a test.
     */
    protected function setUp() {
        parent::setUp ();
        $this->Calculatrice = new Calculatrice();
    }

    /**
     * Cleans up the environment after running a test.
     */
    protected function tearDown() {
        $this->Calculatrice = null;
        parent::tearDown ();
    }

    /**
     * Tests Calculatrice->addition()
     */
    public function testAddition() {

    }

    /**
     * Tests Calculatrice->division()
     */
    public function testDivision() {

    }

    /**
     * Tests Calculatrice->multiplication()
     */
    public function testMultiplication() {

    }

    /**
     * Tests Calculatrice->soustraction()
     */
    public function testSoustraction() {

    }
}
```

Il s'agit maintenant d'écrire les tests et voir s'ils passent, ce qui ne devrait pas être le cas ! Nous allons donc écrire quelques lignes de tests dans chaque méthode.

Classe CalculatriceTest

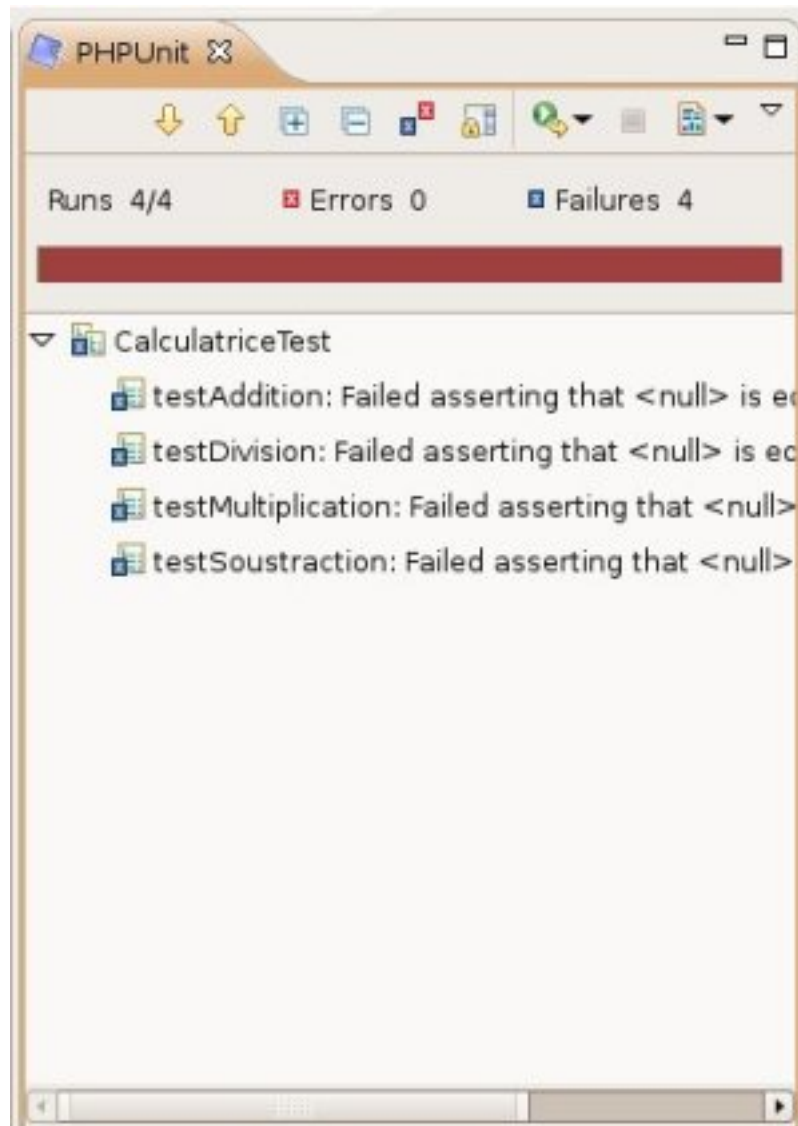
```
/**
 * Tests Calculatrice->addition()
 */
public function testAddition() {
    $result = $this->Calculatrice->addition(3, 4);
    $this->assertEquals(7, $result);
    $result = $this->Calculatrice->addition(-3, -4);
    $this->assertEquals(-7, $result);
    $result = $this->Calculatrice->addition(-3, 4);
    $this->assertEquals(1, $result);
}

/**
 * Tests Calculatrice->division()
 */
public function testDivision() {
    $result = $this->Calculatrice->division(12, 3);
    $this->assertEquals(4, $result);
    $result = $this->Calculatrice->division(-12, 3);
    $this->assertEquals(-4, $result);
    $result = $this->Calculatrice->division(-12, -3);
    $this->assertEquals(4, $result);
    $this->setExpectedException('Exception', 'Division par zéro');
    $this->Calculatrice->division(12, 0);
}

/**
 * Tests Calculatrice->multiplication()
 */
public function testMultiplication() {
    $result = $this->Calculatrice->multiplication(3, 4);
    $this->assertEquals(12, $result);
    $result = $this->Calculatrice->multiplication(-3, -4);
    $this->assertEquals(12, $result);
    $result = $this->Calculatrice->multiplication(-3, 4);
    $this->assertEquals(-12, $result);
}

/**
 * Tests Calculatrice->soustraction()
 */
public function testSoustraction() {
    $result = $this->Calculatrice->soustraction(3, 4);
    $this->assertEquals(-1, $result);
    $result = $this->Calculatrice->soustraction(-3, -4);
    $this->assertEquals(1, $result);
    $result = $this->Calculatrice->soustraction(3, -4);
    $this->assertEquals(7, $result);
}
```

Une fois nos tests en place, il faut les exécuter et constater qu'ils ne passent pas. Pour cela faites un clic droit sur la classe CalculatriceTest puis dans le menu contextuel sélectionnez Run As - PHPUnit Test. Vous verrez ensuite dans la vue PHPUnit une barre rouge pour vous indiquer que des tests ont échoué, en regardant plus en détail vous verrez que tous vos tests ont échoué, ce qui est normal. Voici le résultat que vous devez obtenir dans la vue PHPUnit :



Les tests unitaires ne passent pas

Etant donné que les tests échouent, il faut aller coder le corps des méthodes afin qu'il passent ! Nous allons également commenter les méthodes afin de générer, par la suite, une documentation avec PhpDoc. Voici la classe Calculatrice :

Classe Calculatrice

```

<?php
**
* Classe permettant de réaliser des calculs arithmétiques simples
* @author Alain Sahli
* @name Calculatrice
* @package calc
* @version 1.0
*
*/
class Calculatrice
{
/**
 * Additionne deux nombres
 * @param double $nb1
 * @param double $nb2
 * @return double

```

Classe Calculatrice

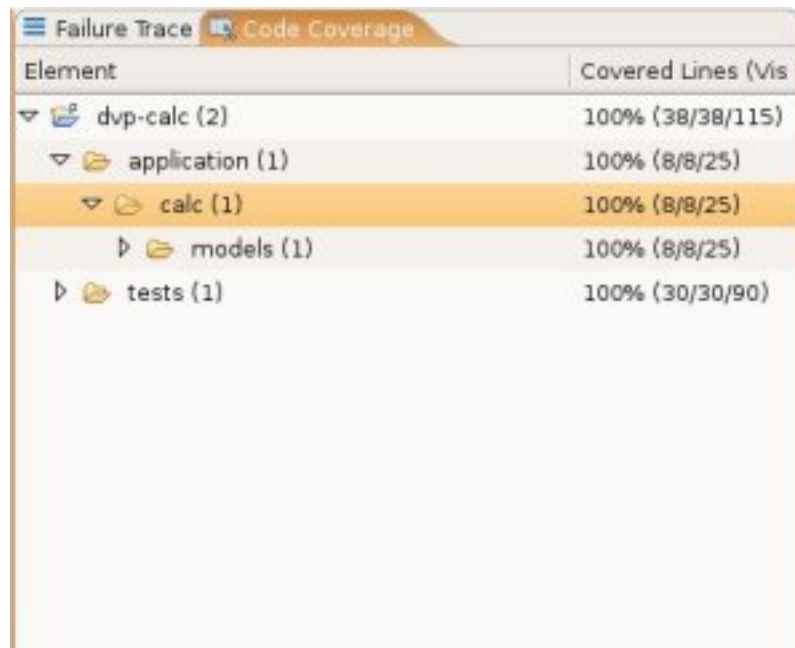
```
*/
public function addition($nb1, $nb2){
    return $nb1 + $nb2;
}

/**
 * Soustrait deux nombres
 * @param double $nb1
 * @param double $nb2
 * @return double
 */
public function soustraction($nb1, $nb2){
    return $nb1 - $nb2;
}

/**
 * Multiplie deux nombres
 * @param double $nb1
 * @param double $nb2
 * @return double
 */
public function multiplication($nb1, $nb2){
    return $nb1 * $nb2;
}

/**
 * Divise deux nombres
 * @param double $nb1
 * @param double $nb2
 * @return double
 */
public function division($nb1, $nb2){
    if($nb2 != 0)
    {
        return $nb1 / $nb2;
    }
    else
    {
        throw new Exception('Division par zéro');
    }
}
}
```

Si vous relancez les tests, vous verrez qu'ils ont passé au vert. Sous la vue PHPUnit, vous avez un onglet intitulé « Code coverage » permettant de vous indiquer en pourcentage la partie de code que vous avez testé dans la classe calculatrice, le but étant d'obtenir 100%.



Element	Covered Lines (Vis)
▼ dvp-calc (2)	100% (38/38/115)
▼ application (1)	100% (8/8/25)
▼ calc (1)	100% (8/8/25)
▶ models (1)	100% (8/8/25)
▶ tests (1)	100% (30/30/90)

Vue code coverage

Nous devons encore nous occuper du contrôleur : il ne sera pas testé car il faudrait simuler un dispatch et je sortirais un peu trop du contexte de présentation de Zend Studio for Eclipse. Selon le design pattern MVC, les contrôleurs ne devraient contenir que quelques lignes et passer toute la logique à la couche modèle. C'est ce que nous allons faire grâce à notre classe Calculatrice. Voici la classe CalculatriceController :

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Calculatrice');

/**
 * Classe de lien entre la vue et le modèle Calculatrice
 *
 * @author Alain Sahli
 * @version 1.0
 */
class CalculatriceController extends Zend_Controller_Action {

    private $calc;

    /**
     * Action de contrôleur qui effectue une addition
     */
    public function additionAction(){
        $nb1 = $this->_request->getPost('nb1');
        $nb2 = $this->_request->getPost('nb2');

        $this->view->result = $this->calc->addition($nb1, $nb2);
        $this->view->nb1 = $nb1;
        $this->view->nb2 = $nb2;
    }

    /**
     * Action de contrôleur qui effectue une soustraction
     */
    public function soustractionAction(){
        $nb1 = $this->_request->getPost('nb1');
        $nb2 = $this->_request->getPost('nb2');
```

```

        $this->view->result = $this->calc->soustraction($nb1, $nb2);
        $this->view->nb1 = $nb1;
        $this->view->nb2 = $nb2;
    }

    /**
     * Action de contrôleur qui effectue une multiplication
     */
    public function multiplicationAction(){
        $nb1 = $this->_request->getPost('nb1');
        $nb2 = $this->_request->getPost('nb2');

        $this->view->result = $this->calc->multiplication($nb1, $nb2);
        $this->view->nb1 = $nb1;
        $this->view->nb2 = $nb2;
    }

    /**
     * Action de contrôleur qui effectue une division
     */
    public function divisionAction(){
        $nb1 = $this->_request->getPost('nb1');
        $nb2 = $this->_request->getPost('nb2');

        $this->view->result = $this->calc->division($nb1, $nb2);
        $this->view->nb1 = $nb1;
        $this->view->nb2 = $nb2;
    }

    /**
     * @see Zend_Controller_Action::preDispatch()
     */
    public function preDispatch() {
        parent::preDispatch();
        $this->calc = new Calculatrice();
    }
}

```



Vous pouvez "override" des méthodes de la classe parente en faisant un clique droit dans le code puis Source - Override/Implement Methods...

III-D - L'interface avec l'éditeur HTML WYSIWYG


Notre application est maintenant prête ! Mais il nous faut encore une interface utilisateur pour exécuter ces différentes opérations. Nous allons utiliser pour cela l'éditeur HTML WYSIWYG intégré dans Zend Studio for Eclipse.

Utilisez la vue MVC pour vous rendre dans le dossier Views/scripts/index, ensuite faites un clique droit sur le fichier index.phtml puis Open With - PHP/HTML WYSIWYG Editor. Vous obtenez ensuite une zone principale qui doit ressembler à ceci :

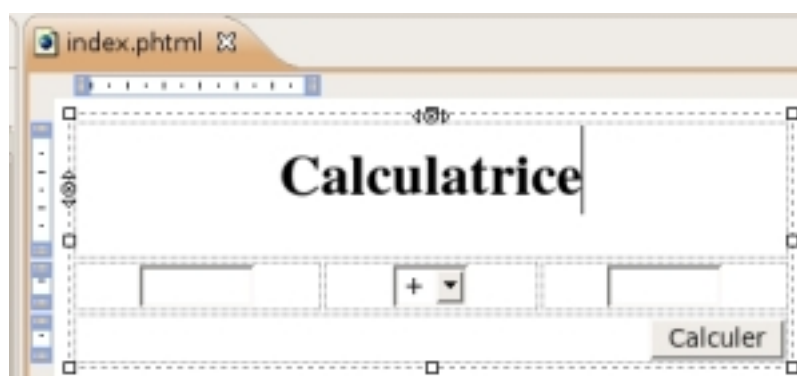


Mode design de l'éditeur HTML

Comme vous pouvez le voir sur cette image, vous avez à disposition trois onglets qui vous permettent de travailler en mode « Design », « Source » ou les deux, les habitués de Dreamweaver ne devraient pas être dépayés ! On vous propose également un onglet « Preview » afin de prévisualiser votre interface dans un navigateur.

 Vous pouvez configurer dans les options (*Window - Preferences - General - Web browser*) quel navigateur utiliser pour la prévisualisation !

Nous allons donc créer une interface pour effectuer les différents calculs. Passez en mode Design puis, à l'aide de la vue PHP/HTML Toolbox créez un tableau de 3 lignes et 3 colonnes. Fusionnez ensuite les trois premières cellules en les sélectionnant puis en faisant un clic droit et enfin Table - Merge Cells. Insérez le titre « Calculatrice » dans cette cellule fusionnée, sélectionnez-le et formatez-le en Heading 1 à l'aide de la barre d'outil en haut de l'éditeur. Utilisez la vue PHP/HTML Toolbox pour créer l'interface ci-dessous :



Interface HTML de la calculatrice

Ci-dessous vous trouvez le code HTML correspondant, notez bien le nom des champs ainsi que celui du formulaire.

Formulaire HTML

```
<form name="calculatrice" method="post" action="calculatrice/addition">
  <table width="400px" height="91px">
```

Formulaire HTML

```

<tbody>
  <tr>
    <td width="400" valign="top" align="center" rowspan="1" colspan="3">
      <h1>Calculatrice</h1>
    </td>
  </tr>
  <tr>
    <td width="400" valign="top" align="center">
      <input type="text" maxlength="5" size="5" name="nb1">
    </td>
    <td width="400" valign="top" align="center">
      <select name="operation">
        <option value="addition">+</option>
        <option value="soustraction">-</option>
        <option value="multiplication">*</option>
        <option value="soustraction">/</option>
      </select>
    </td>
    <td width="400" valign="top" align="center">
      <input type="text" maxlength="5" size="5" name="nb2">
    </td>
  </tr>
  <tr>
    <td width="400" valign="top" align="right" rowspan="1" colspan="3">
      <input type="submit" name="calculer" value="Calculer">
    </td>
  </tr>
</tbody>
</table>
</form>
    
```

 Si vous n'avez pas la vue PHP/HTML Toolbox, ajoutez-la en suivant les indications données dans le chapitre En avant avec Zend Studio for Eclipse.

La vue principale est maintenant créée, mais il nous faut encore créer des vues pour les différentes actions ainsi qu'une vue d'entête et de pied de page.

Pour commencer nous allons créer l'entête et le pied de page, créez donc deux nouvelles vues dans le dossiers views/scripts et nommez-les *header.phtml* et *footer.phtml*. Voici le code à placer dans ces deux vues :

header.phtml

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Calculatrice</title>
  </head>
  <body>
    
```

footer.phtml

```

  </body>
</html>
    
```

Nous allons créer un dossier calculatrice au même niveau que le dossier index, ensuite créez une nouvelle vue *addition.phtml* dans ce dossier. Ce fichier contiendra le résultat de l'addition :

addition.phtml

```

<?php echo $this->render('header.phtml'); ?>
<?php echo $this->escape($this->nb1); ?>&nbsp;+&nbsp;<?php echo $this->escape($this->nb2);
?>&nbsp;=&nbsp;<?php echo $this->escape($this->result); ?><br/>
    
```

addition.phtml

```
<a href="/dvp-calc/html/">Retour</a>
<?php echo $this->render('footer.phtml'); ?>
```

Ce code n'est pas très compliqué et vous pouvez faire la même chose pour les 3 autres actions (soustraction, multiplication, division).

Vous aurez certainement remarqué l'inclusion de l'entête et du pied de page, ajoutez ces deux lignes également dans la vue index.phtml afin de générer de l'HTML correct.

III-E - Support du code Javascript

Notre interface est maintenant prête, mais nous devons encore taper quelques lignes de Javascript pour initialiser l'action du formulaire en fonction de l'opération arithmétique sélectionnée. Pour faire cela, rendez vous dans le dossier html / scripts et créez un nouveau fichier Javascript nommé calc-selector.js. Vous remarquerez au passage que le code Javascript est supporté par Zend Studio for Eclipse et qu'il dispose d'une fonctionnalité d'autocomplétion. Tapez le code suivant afin de modifier l'action de notre formulaire en fonction de l'opérateur arithmétique sélectionné.

calc-selector.js

```
function setAction(operator)
{
    switch(operator)
    {
        case 'addition':
            document.calculatrice.action = 'calculatrice/addition';
            break;
        case 'soustraction':
            document.calculatrice.action = 'calculatrice/soustraction';
            break;
        case 'multiplication':
            document.calculatrice.action = 'calculatrice/multiplication';
            break;
        case 'division':
            document.calculatrice.action = 'calculatrice/division';
            break;
    }
}
```

Il faut maintenant modifier notre vue index.phtml pour que ce script soit appelé lorsqu'on sélectionne une opération dans la liste déroulante. Pour cela, ajoutez un événement onchange dans la balise select :

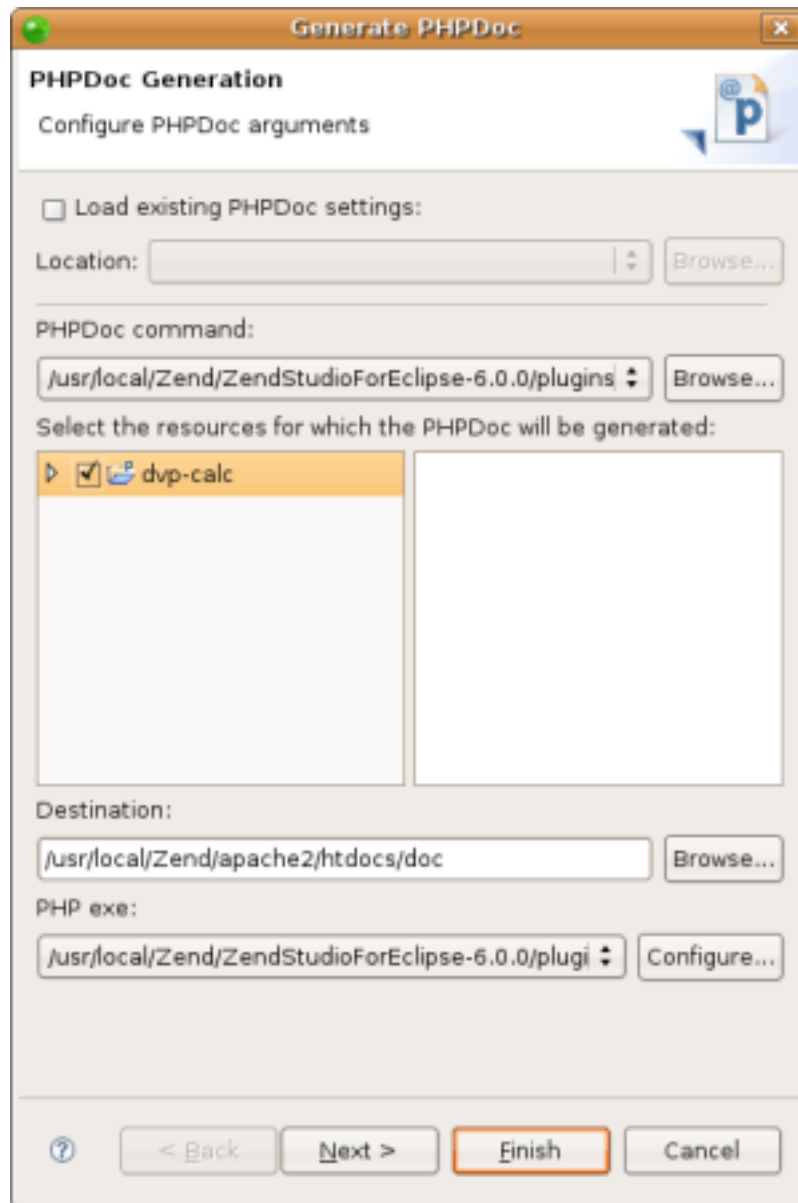
```
<select onchange="setAction(this.options[this.selectedIndex].value);" name="operation">
```

Voilà, notre code est maintenant terminé et il fonctionne correctement !

III-F - Génération de documentation


Vous avez peut-être remarqué que dans tous mes commentaires j'ai utilisé une certaine structure ainsi que des tags commençant par @. J'ai fait ceci car Zend Studio for Eclipse dispose d'une fonction de génération de documentation basée sur phpDocumentor. Grâce à ces commentaires, nous sommes maintenant en mesure de générer une documentation de projet.

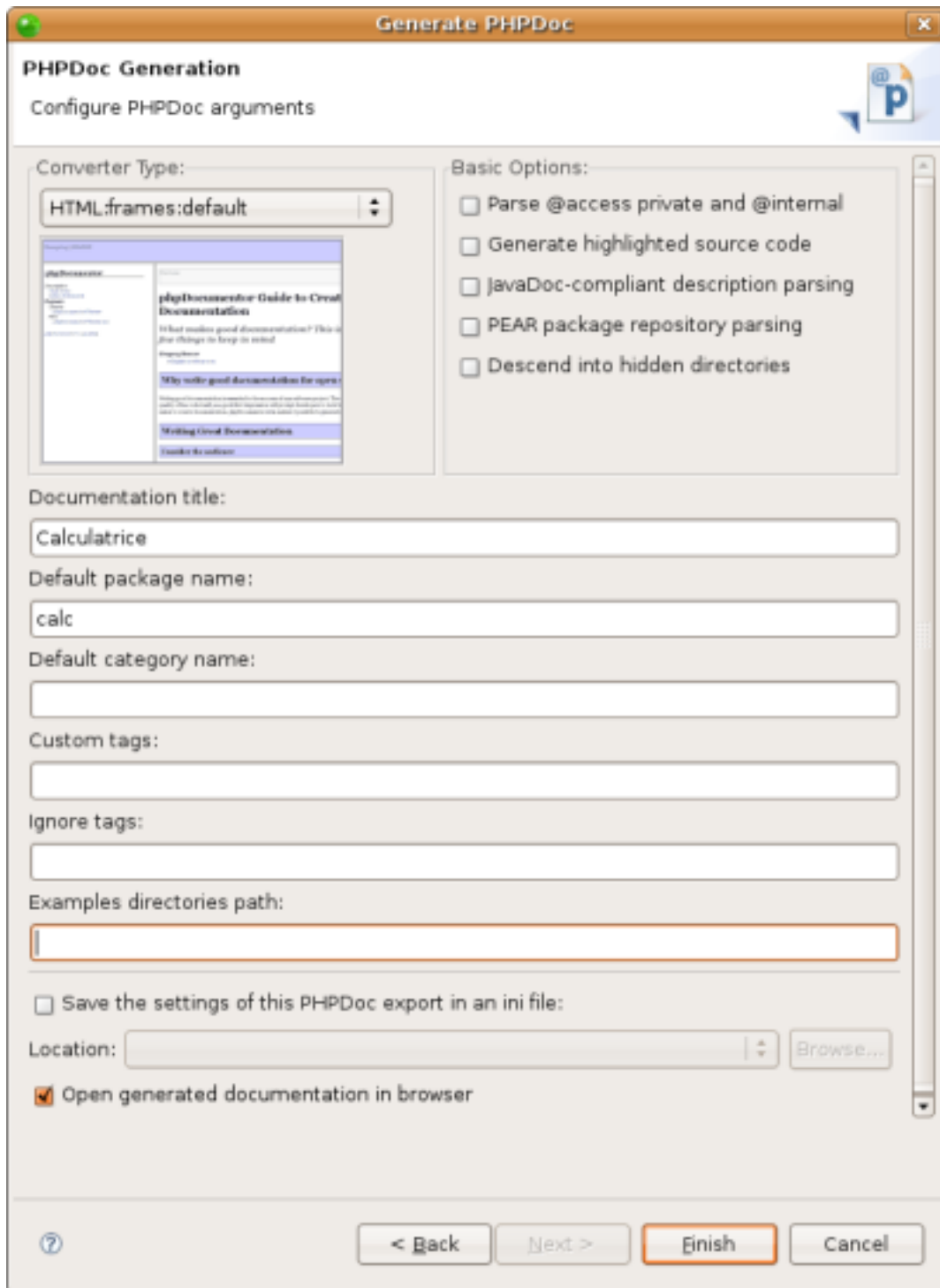
Rendez-vous dans le menu Project - Generate PHPDoc la fenêtre, ci-dessous va apparaître :



Génération de la documentation - étape 1

Remplissez les champs comme indiqué ci-dessus puis cliquez sur suivant.

Sur la fenêtre suivante, vous pouvez définir le titre de votre documentation et d'autres options propres à PHPDocumentor, veuillez vous référer à la  [documentation de PHPDocumentor](#) pour en savoir plus sur ces options.



Génération de la documentation - étape2

En cliquant sur finish, la documentation est générée et le résultat est directement ouvert dans votre navigateur.

III-G - Débogage

Un des grands atouts de Zend Studio for Eclipse est que vous pouvez faire du débogage. Il y a deux types de débogage, soit vous déboggez en « local » ou alors via une URL distante. Le débogage local fonctionne sans

problèmes car l'exécutable Zend Debugger est automatiquement installé avec le logiciel. Par contre, si vous voulez débogger une page distante, vous devrez installer la Zend Platform, qui elle a besoin de Zend Core. Ces produits sont gratuits pour une utilisation personnelle, donc très pratiques pour faire du développement.

III-G-1 - Débogage distant

Prenez par exemple le fichier index.php et insérez un « breakpoint » à la ligne 16, pour ce faire double cliquez dans la marge à la ligne 16 ou alors laissez le curseur à la ligne 16 et cliquez sur le menu Run - Toggle Breakpoint. Vous devez obtenir un petit point dans la marge comme ceci :




Une fois ce breakpoint ajouté, vous pouvez cliquer sur le menu Run - Debug URL et insérez l'URL qui pointe sur la racine de votre projet. Décochez la case « Break at First Line » ainsi le debugger arrivera directement à notre breakpoint. Une fois que vous aurez cliqué sur ok vous allez passer en perspective PHP Debug. Ensuite vous pouvez avancer dans le code en utilisant les touches F5 et F6. En pressant F5, vous entrez dans la méthode et en pressant F6, vous passez outre. Vous avez également une vue à disposition intitulée Variables qui vous permet de voir la valeur de vos variables ainsi que celle des variables globales. Grâce à cette vue, vous avez même la possibilité de changer les valeurs des variables !

III-G-2 - Débogage local

Le débogage local est moins puissant que le débogage distant car il n'utilise pas le serveur Web. Par conséquent, les variables globales comme \$_POST, \$_GET, etc. ne sont pas initialisées, mais vous pourrez les définir une fois la perspective PHP Debug ouverte. Zend Studio for Eclipse utilise simplement les configurations locales et ne vous permet pas de débogger votre code dans un environnement de production. C'est néanmoins très pratique pour corriger rapidement quelques erreurs de syntaxe ou des problèmes de valeurs nulles...

Pour débogger en local, faites simplement un clique droit sur le fichier que vous voulez débogger puis cliquez sur Debug As - PHP Script. La perspective PHP Debug va s'ouvrir et vous pourrez débogger de manière similaire au débogage distant.

 *Si vous avez défini des include paths dans votre php.ini, n'oubliez pas de les ajouter dans Zend Studio for Eclipse. Vous pouvez définir les include paths en utilisant la vue PHP Explorer puis en faisant un clique droit sur Include Paths et enfin Configure Include Path.*

IV - Conclusion

Cet éditeur offre un confort non négligeable, les fonctionnalités sont très puissantes et fonctionnent vraiment bien. Il est très intéressant de voir que cet environnement se rapproche beaucoup de Java. Je pense notamment à l'interface dédiée aux tests unitaires qui est strictement identique à celle de Junit.

Zend Studio for Eclipse répond certainement aux projets de grande envergure, et grâce à lui les développeurs PHP n'ont plus rien à envier aux éditeurs comme Visual Studio ou Eclipse pour Java. De plus, l'intégration de vues pour le Zend Framework apporte un réel confort lors du développement de projets basés sur celui-ci.

Pour conclure, ce que j'apprécie particulièrement dans ce nouvel éditeur c'est l'intégration des tests unitaires ainsi que les vues propres au Zend Framework.

IV-A - Remerciements

Je remercie Zend pour la mise à disposition d'une licence de Zend Studio for Eclipse, ainsi que Yogui pour la correction du tutoriel.

