Tutoriel – SimpleTest

# SimpleTest qu’est-ce que c’est ?

SimpleTest est un framework PHP fournissant des outils de tests permettant notamment de tester les interfaces graphiques des pages web en simulant les entrées et sorties attendues par l’utilisateur.

Mais aussi de faire des tests unitaires qui permettent de tester du code PHP vérifiant le bon fonctionnement de la page internet.

On peut encore noter que SimpleTest offre la possibilité de regrouper des tests.

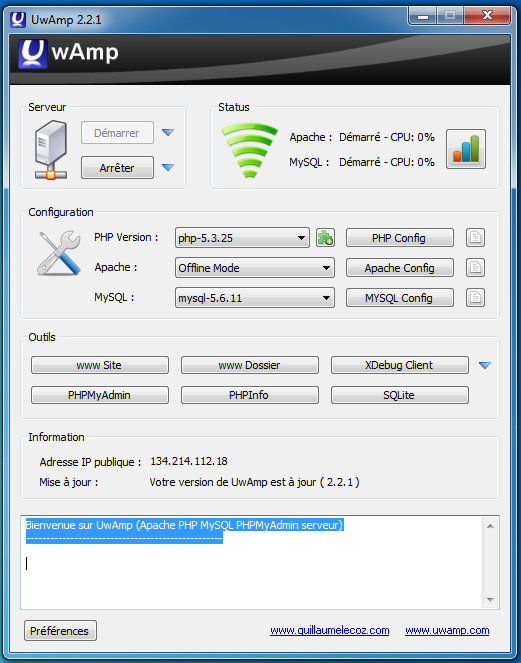
# Prérequis au tutoriel

Première prise en main :

* Dézipper tutoSimpleTest.zip
* Double-cliquer sur uWamp.exe

uWamp est un serveur web portable. Il comprend un serveur http pour fournir les pages web, une base données et un interpréteur PHP.

Si le statut est vert comme dans l’encadré. Alors, votre serveur est démarré et fonctionnel

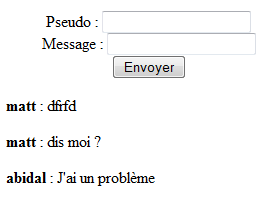


A partir de ce moment-là, il n’y a plus aucune installation à faire. Le framework SimpleTest ainsi que le programme PHP qui nous servira à découvrir ce framework sont situés dans le dossier « www » du dossier « UwAmp ».

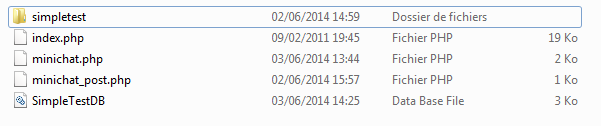
# C’est parti !

Le programme PHP qui va nous servir à nous approprier SimpleTest est un minichat constitué de deux fichiers PHP : *minichat.php* et *minichat\_post.php (Dans UwAmp\www)*.

*minichat.php*, une fois interprété par uWamp, produit l’interface graphique sous forme de page internet. Elle charge les messages en accédant à la base de données.



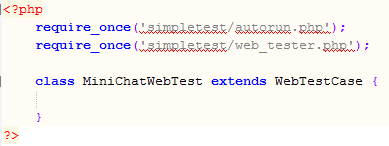
*minichat\_post.php,* quant à lui, sert de relais à *minichat.php* pour insérer les messages dans la base de données. Une fois l’insertion effectué, il redirige l’utilisateur vers *minichat.php* pour en avoir l’affichage.



## Les tests d’interface graphique

La classe du framework qui permet de tester la bonne réaction des interfaces graphiques est la classe WebTestCase. Pour l’utiliser, il nous faut créer nous même une classe qui hérite de WebTestCase. Ainsi, dans la classe fille, on aura accès aux méthodes implémentés par WebTestCase qui permettent de simuler un utilisateur.

*Code minimal pour implémenter les tests d’interfaces graphiques :*



Le test d’interface graphique que nous allons faire est basique mais idéal pour donner une bonne approche de ce que peut être ce genre de tests.

La totalité des fonctions disponibles avec la classe WebTestCase sont données dans les annexes.

### Le but du test :

Ce test va nous servir à vérifier que le formulaire de *minchat.php* nous renvoie bien sur *minichat\_post.php*.

### La manière dont nous allons procéder :

Nous allons créer un fichier miniChatWebTest.php dans lequel nous allons écrire le code minimal présenté plus haut. Au sein de notre classe MiniChatWebTest, nous allons créer une fonction testRedirectionFormulaire. Comme cette fonction a le nom qui commence par « test » elle sera exécuté automatiquement lors de l’appel de la page miniChatWebTest.php.



Pour réaliser notre test, nous allons avoir besoin d’un ensemble de fonctions fourni par SimpleTest :

* La méthode get($url) qui prend en paramètre l’url de la page que l’on veut charger (de type chaine de caractères). Il va falloir l’utiliser pour se connecter à minichat.php.
* La méthode assertResponse($codeHTTP) qui vérifie que le code HTTP (de type entier) retourné est le bon. Les codes qui nous intéressent pour ce tutoriel sont les codes 200 et 302. 200 signifie que le chargement d’une page s’est déroulé avec succès et 302 signifie que nous avons été redirigés vers une page de manière indirecte.
* La méthode setFieldByName($name, $valeure) qui prend en premier paramètre l’attribut name d’une balise input d’un formulaire et en deuxième paramètre la valeure qui l’on veut y mettre.
* Enfin, la méthode clickSubmit($etiquette) qui prend en paramètre le nom du bouton d’envoi tel qu’il apparait dans le navigateur et simule un click dessus.

L’usage de ces fonctions se fait en les préfixant par « $this-> ».

Exemple $this->get(‘http://localhost’).

**Il est nécessaire de configurer SimpleTest pour qu’il accepte les redirections en ajoutant la ligne de code $this->setMaximumRedirects(0) au début de la méthode de test.**

Pour exécuter le test, tapez <http://localhost/miniChatWebTest.php> dans votre navigateur. Si tout c’est bien passé, une bande verte apparait (rouge dans le cas contraire).

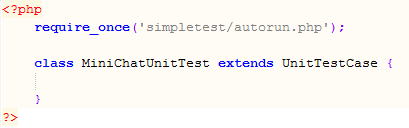
### Solution : (ATTENTION NE PAS REGARDER AVANT DE L’AVOIR FAIT)



## Les tests unitaires

La classe du framework qui permet de tester le bon fonctionnement du code PHP est la classe UnitTestCase. De même que pour les tests d’interfaces graphiques, pour utiliser les tests unitaires, il nous faut créer nous même une classe qui hérite de la classe UnitTestCase.

*Code minimal pour implémenter les tests unitaires :*



Pour s’entrainer aux tests unitaires, nous allons en réaliser deux. Un pour *minichat.php* et un pour *minichat\_post.php* Dans un cas normal d’utilisation de SimpleTest (le test des méthodes d’une classe), nous aurions rajouter require\_once(‘maClasse.php’) afin d’utiliser les méthodes de la classe à tester. Mais ici le programme est trop simple pour faire cela. C’est pourquoi nous allons simplement recopier les morceaux de code que l’on veut tester dans nos méthodes de tests.

Comme précédemment, nous allons créer un fichier miniChatUnitTest.php dans lequel nous allons écrire le code minimal ci-dessus. Au sein de notre classe MiniChatUnitTest, nous allons créer nos deux fonctions de test.

### Test de minichat\_post.php :

Voici le code que nous allons tester pour minichat\_post.php :

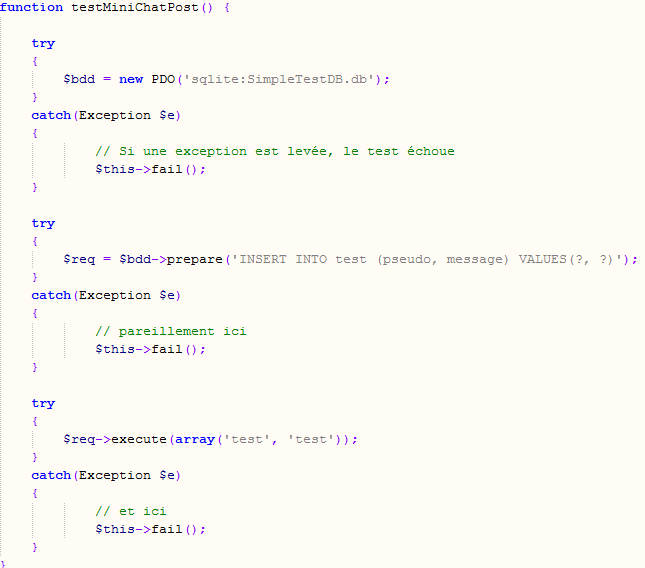


Le but de ce test étant de vérifier le bon dialogue avec la base de données, nous allons placer chaque ligne dans un try catch et déclencher un mauvais résultat du test si une exception est levée.

On peut forcer un mauvais résultat grâce à la méthode fail().

Pour exécuter le test, tapez <http://localhost/miniChatUnitTest.php> dans votre navigateur.

### Solution du test de minichat\_post.php :



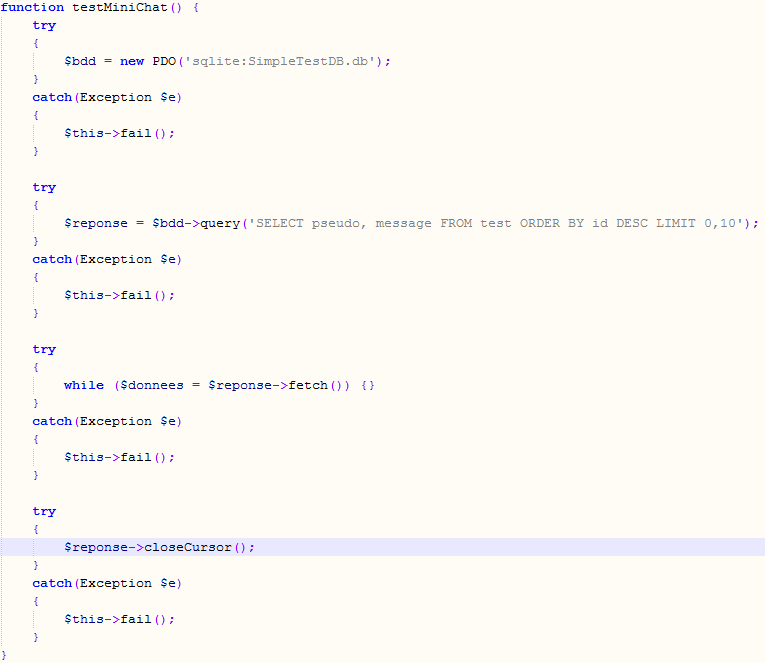
### Test de minichat.php :

Voici le code que nous allons tester pour minichat.php :



Comme tout à l’heure, le but de ce test est de vérifier le bon dialogue avec la base de données. Nous utiliserons la même technique du try catch.

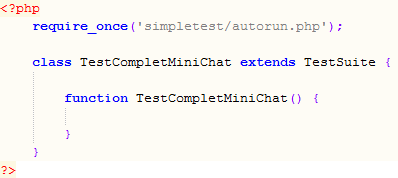
### Solution du test de minichat.php :



## Les groupes de tests

Les groupes de tests sont une fonctionnalité très pratique de SimpleTest car ils permettent d’exécuter plusieurs tests d’un coup. Cela permet notamment de les regrouper par logique métier.

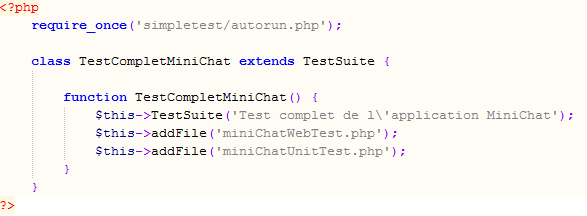
*Code minimal pour implémenter les groupes de tests :*



Pour achever ce tutoriel, nous allons réunir tous nos tests dans un seul groupe de tests.

Les seules méthodes dont nous allons avoir besoin sont TestSuite($nom) qui prend en paramètre le nom que l’on donne au groupe de test (pas obligatoire mais recommander) et addFile($chemin) qui prend en paramètre le chemin d’un fichier de test à inclure dans le groupe de tests.

### Solution :



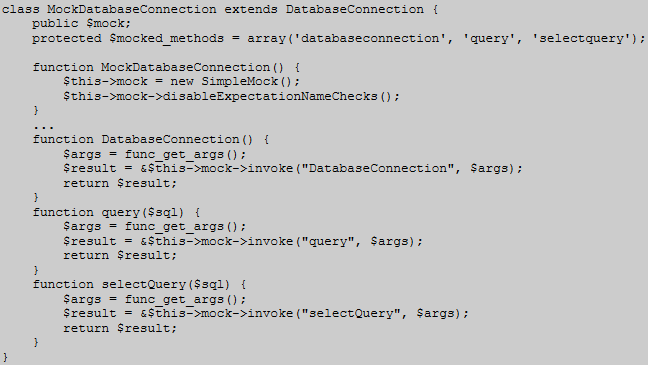
# Pour aller plus loin.... les Mocks

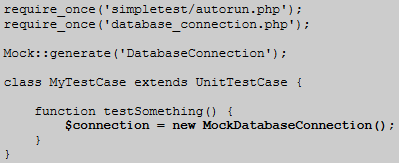
Les Mocks agissent comme des proxys au niveau du code. Ils servent à simuler le bon fonctionnement d’une classe en permettant de rentrer des jeux de tests. Ils sont souvent utilisés sur des classes de type DAO (Data Access Object) qui communiquent avec la base de données. Cela permet au testeur de pouvoir concentrer ses tests uniquement sur une classe sans se soucier du bon fonctionnement de ses dépendances.

Avec SimpleTest, la création d’un Mock se fait en créant une classe qui hérite de la classe à « mocker » et en en redéfinissant toutes les méthodes qui retourneront des données écrites « en dur ».

Enfin, il faut implémenter le ou les Mock(s) dans le test.

*Exemple d’implémentation d’un mock :*

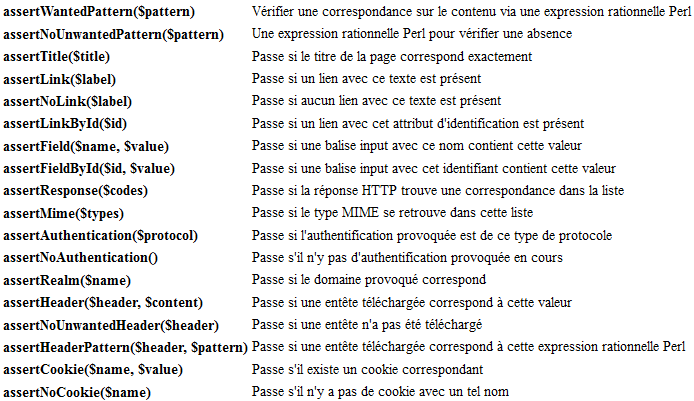




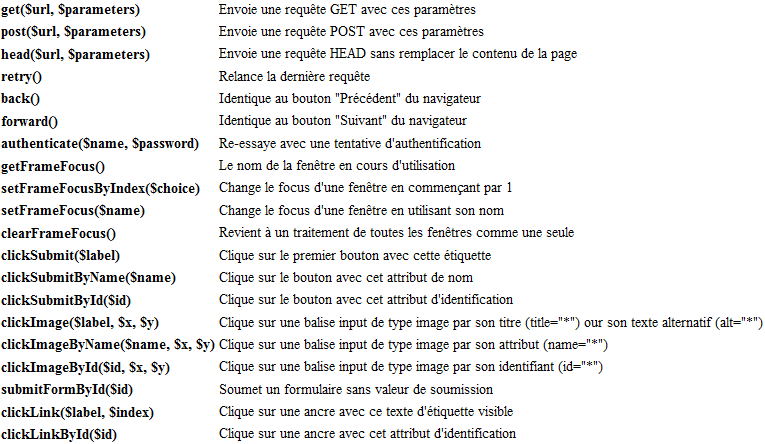
# Annexes

*Fonctionnalités de WebTestCase :*

Assertions :



Navigation :



*Assertions de UnitTestCase :*

