

Créez vos formulaires facilement avec le package HTML_quickform de PEAR

par Jérôme CHABAUD ([Site perso](#))

Date de publication : 29 mai 2004

Dernière mise à jour :

Cet article expose le mode d'utilisation du package HTML_quickform de la bibliothèque PEAR

- I - Avant-propos
 - I-A - Pré-requis
 - I-B - Public
 - I-C - Pourquoi ce tutoriel ?
- II - Le package HTML_Quickform
 - II-A - Qu'est ce que le package HTML_Quickform ?
 - II-B - Pourquoi utiliser HTML_QuickForm ?
- III - Installation
- IV - Notre premier formulaire
 - IV-A - Petit cahier des charges
 - IV-B - Étape 1 : créer le formulaire
 - IV-C - Étape 2 : contrôler les saisies
 - IV-D - À vous de travailler !!
 - IV-E - Récupérer les données
- V - Présentation rapide des autres pseudo-éléments
 - V-A - Advcheckbox
 - V-B - Hierselect
 - V-C - File
 - V-D - Select
 - V-E - Header
- VI - Conclusion

I - Avant-propos

I-A - Pré-requis

Avoir déjà écrit le code HTML d'un formulaire, connaître les bases du langage PHP, avoir quelques (très petites) notions de programmation objet, avoir un serveur http avec php 4.3.0 d'installé. Pour l'utilisateur de Windows, EasyPHP1.7 ira très bien.

I-B - Public

Webmasters, développeurs php. Ce tutoriel est très détaillé et donc particulièrement **destiné aux programmeurs débutants**.

I-C - Pourquoi ce tutoriel ?

Parce que le package HTML_QuickForm est relativement complexe. Il est donc difficile, pour un débutant, de se plonger directement dans l'utilisation de ce package sans un guide assez détaillé.

II - Le package HTML_Quickform

II-A - Qu'est ce que le package HTML_Quickform ?

Le package HTML_quickform permet de réaliser des formulaires complexes sans avoir à s'occuper du code HTML. De plus ce package offre un certain nombre de fonctionnalités dont la validation et le filtrage des saisies et bien d'autres choses encore...

II-B - Pourquoi utiliser HTML_QuickForm ?

Il existe en effet d'autres classes générant des formulaires (oohform de phplib par exemple). L'avantage du package Pear est qu'il peut être utilisé par d'autres packages aux fonctionnalités très puissantes comme par exemple html_quickform_controler (qui permet de faire assez simplement des formulaires de "type assistant" sur plusieurs pages) et DB_DataObject_formBuilder qui permet de générer des formulaires directement à partir d'informations extraites d'une base de données sans avoir à écrire ni le code SQL ni le code HTML : nous verrons ceci lors d'un prochain tutoriel.

III - Installation

Pour effectuer l'installation de ce package je vous renvoie au tutoriel <http://php.developpez.com/cours/installationpear/>. N'oubliez pas que le package HTML_QuickForm a une dépendance envers HTML_common

IV - Notre premier formulaire

IV-A - Petit cahier des charges

Voici un petit cahier des charges auquel notre premier formulaire devra répondre :

- permettre la saisie d'un pseudo de 6 caractères mini et 10 maxi
- permettre la saisie d'un nom
- permettre la saisie d'une adresse Email valide
- permettre la saisie d'une date de naissance

Tous les champs sont obligatoires

Nous allons procéder par étapes : la première étape va créer le formulaire sans s'occuper des contraintes liées aux saisies. La deuxième va contrôler les saisies. La troisième va récupérer les données saisies.

IV-B - Étape 1 : créer le formulaire

Je vous propose de faire un copier/coller du code suivant, de l'essayer, puis nous allons le détailler .

Script 1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>formulaire</title>
</head>
<body>
<?php
set_include_path(get_include_path() . ";c:\php\pear");
require_once "HTML/QuickForm.php";
$form = new HTML_QuickForm('frmTest', 'post');
$form->addElement('text', 'Pseudo', 'Votre pseudo : ');
$form->display();
?>
</body>
</html>
```

Analysons ligne par ligne ce petit script :

```
set_include_path(get_include_path() . ";c:\php\pear");
```

Permet d'ajouter le chemin d'accès à la bibliothèque PEAR à la liste des include path (voir le tutoriel <http://php.developpez.com/cours/installationpear/>). Si vous utilisez Linux il faut remplacer le point-virgule par deux points (:).

```
require_once "HTML/QuickForm.php";
```

On charge la bibliothèque quickform dans le script .

```
$form = new HTML_QuickForm('formulaire1', 'post');
```

On crée une **nouvelle instance de type HTML_QuickForm**. Le premier argument correspond au nom du formulaire, le second au type d'action utilisé pour transmettre ce formulaire (post ou get)

```
$form->addElement('text', 'Pseudo', 'Votre pseudo : ');
```

On **ajoute au formulaire un élément** de type text ayant pour nom Pseudo et qui sera précédé du texte Votre pseudo. Nous verrons par la suite qu'il existe de nombreux types d'élément dont certains n'existent pas en HTML standard.

```
$form->display();
```

Enfin, on affiche le formulaire à l'écran.

Maintenant que nous avons la base de notre script nous allons **l'enrichir pour arriver au résultat voulu**.

Pour permettre la saisie d'un nom et d'une adresse email il suffit d'ajouter deux lignes comme celles-ci :

```
$form->addElement('text', 'Nom', 'Votre nom : ');
$form->addElement('text', 'Email', 'Votre adresse email : ');
```

Pendant que l'on y est on va ajouter les boutons submit et reset :

```
$form->addElement('reset', 'bouton_clear', 'Effacer');
$form->addElement('submit', 'bouton_effacer', 'Envoyer');
```

Nous avons ici ajouté des éléments de **type reset et submit**. Il nous reste à ajouter la saisie de la date de naissance : ça va se compliquer un (tout petit) peu. Il existe un **pseudo-élément date**. Pourquoi pseudo ? Parce qu'il n'existe pas en HTML. Cet élément nécessite que l'on définisse précédemment quelques options comme le format de la date, le langage ...

```
$options = array(
    'language' => 'fr',
    'format'   => 'dMY',
    'minYear'  => 2001,
    'maxYear'  => 2005
);
```

Pour plus d'informations sur les options, je vous renvoie à la doc. Ensuite on **ajoute l'élément** de type date :

```
$form->addElement('date', 'date', 'votre date de naissance', $options);
```

Notre script complet est donc :

Script 2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>formulaire</title>
</head>
<body>
<?php
set_include_path(get_include_path() . ";c:\php\pear");
require_once "HTML/QuickForm.php";

$form = new HTML_QuickForm('frmTest', 'post');
$form->addElement('text', 'Pseudo', 'Votre pseudo : ');
$form->addElement('text', 'Nom', 'Votre nom : ');
```

Script 2

```
$form->addElement('text', 'Email', 'Votre adresse email : ');
$options = array(
    'language' => 'fr',
    'format'    => 'dMY',
    'minYear'  => 2001,
    'maxYear'  => 2005
);
$form->addElement('date', 'date', 'votre date de naissance : ', $options);

$form->addElement('reset', 'bouton_clear', 'Effacer');
$form->addElement('submit', 'bouton_effacer', 'Envoyer');

$form->display();
?>
</body>
</html>
```

Au passage, vous remarquerez que l'ordre dans lequel est fait l'ajout des éléments est celui dans lequel ils apparaissent à l'écran.

Bon jusqu'ici pas de quoi sauter au plafond !

IV-C - Étape 2 : contrôler les saisies

C'est l'étape la plus complexe mais aussi celle où **on peut gagner beaucoup de temps !!**

A chaque élément on peut affecter des règles. Le non respect de ces règles indique à l'utilisateur que sa saisie n'est pas correcte. La vérification de ces règles peut se faire côté client (en javascript) ou côté serveur (en php) mais ne vous inquiétez pas nous allons voir tout ça en détail.

Nous voulons que le pseudo (dans un premier temps) soit renseigné donc nous allons ajouter (juste avant `$form->display()`) la ligne :

```
$form->addRule('Pseudo', 'Vous devez saisir un pseudo', 'required', '', 'client');
```

Le premier argument de la fonction `addRule()` est le nom de l'élément auquel s'applique la règle. Le deuxième est le texte qui apparaîtra comme message d'erreur. Le troisième est le nom de la règle (nous verrons plus loin les différentes règles disponibles). Dans notre cas on veut utiliser la règle **required** qui signifie que la saisie est obligatoire. Le quatrième argument représente les options de la règle : il n'y en a pas pour la règle **required**. Le cinquième argument spécifie si la validation doit s'effectuer côté client ou serveur (server) : essayons d'abord client.

Si vous exécutez le script, vous verrez apparaître à côté de *Votre nom* une petite étoile ainsi qu'un texte en bas du formulaire signifiant que les champs marqués par cette étoile sont obligatoires. Petit problème ce texte est en anglais. Nous allons donc le remplacer par un texte qui nous convient avec la ligne suivante :

```
$form->setRequiredNote('* = champs obligatoires');
```

Maintenant, essayons de valider notre formulaire (ne rien saisir dans le champ *pseudo*) : on clique sur *Envoyer*. Si votre navigateur supporte le javascript vous devriez voir apparaître une fenêtre vous indiquant que la saisie du pseudo est obligatoire. A nouveau les textes de cette fenêtre sont en anglais. Nous les redéfinissons par la ligne :

```
$form->setJsWarnings('Erreur de saisie ', 'Veuillez corriger');
```

IV-D - À vous de travailler !!

Je vous laisse ajouter les lignes qui rendent la saisie obligatoire des champs Nom, Email. Ca y est, alors on va pouvoir s'intéresser aux 6 caractères mini et 10 maxi du pseudo. Il faut utiliser la règle **rangelength**

```
$form->addRule('Pseudo', 'Votre pseudo doit avoir entre 6 caractères et 10 caractères',  
'rangelength', array(6,10), 'client');
```

Pour être sûr que l'adresse **Email** a une forme valide on utilisera la règle email :

```
$form->addRule('Email', 'Vous devez saisir une adresse email valide ', 'email', '', 'client');
```

Les différentes règles standard sont décrites dans la doc .

Sachez qu'il est possible de **définir ses propres règles** mais ceci dépasse le cadre de ce tutoriel. Vous trouverez des exemples de définition de règles dans le script rules-custom.php du répertoire docs contenu dans l'archive du package.

Et si le navigateur ne "comprend" pas le javascript ? Et bien on va utiliser la fonction **validate()**. Cette fonction renvoie true si toutes les règles sont respectées. On va donc ajouter la condition suivante à la fin de notre script :

```
if ($form->validate()) {  
    echo 'Toutes les règles sont respectées';  
}  
else {  
    $form->display();  
}
```

Désactiver JavaScript et envoyer les données saisies pour voir la validation côté serveur. Avouez que le **résultat est satisfaisant vu le faible nombre de lignes de code !!**

Notre formulaire a encore un autre petit problème : si l'utilisateur ne saisit que des espaces dans *les cases pseudo ou nom*, les règles seront considérées comme respectées. On va donc filtrer les saisies pour que les espaces de fin de ligne soit supprimés. Pour cela nous allons utiliser la méthode **applyFilter**.

```
$form->applyFilter('Nom','trim');  
$form->applyFilter('Pseudo','trim');
```

Le principe est le même que pour les règles : le premier argument est le nom de l'élément auquel s'applique le filtre et le deuxième est le nom du filtre. En standard il n'existe que le filtre **trim** qui supprime les espaces de fin de chaîne.

Notre script final sera donc le suivant :

Script 3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">  
<html>  
<head>  
<title>formulaire</title>  
</head>  
<body>  
<?php  
    set_include_path(get_include_path() . ":c:\php\pear");  
    require_once "HTML/QuickForm.php";  
  
    $form = new HTML_QuickForm('frmTest', 'post');
```

Script 3

```

$form->addElement('text', 'Pseudo', 'Votre pseudo : ');
$form->addElement('text', 'Nom', 'Votre nom : ');
$form->addElement('text', 'Email', 'Votre adresse email : ');
$options = array(
    'language' => 'fr',
    'format' => 'dMY',
    'minYear' => 2001,
    'maxYear' => 2005
);
$form->addElement('date', 'date', 'votre date de naissance : ', $options);
$form->addRule('Pseudo', 'Vous devez saisir un pseudo', 'required', '', 'client');
$form->addRule('Nom', 'Vous devez saisir un nom', 'required', '', 'client');
$form->addRule('Email', 'Vous devez saisir une adresse Email', 'required', '', 'client');
$form->addRule('Pseudo', 'Votre pseudo doit avoir entre 6 caractères et 10 caractères',
'ranglength', array(6,10), 'client');
$form->addRule('Email', 'Vous devez saisir une adresse email valide', 'email', '', 'client');
$form->applyFilter('Nom', 'trim') ;
$form->applyFilter('Pseudo', 'trim') ;
$form->setRequiredNote('<span style="color: #ff0000">*</span> = champs obligatoires');
$form->setJsWarnings('Erreur de saisie', 'Veuillez corriger');
$form->addElement('reset', 'bouton_clear', 'Effacer');
$form->addElement('submit', 'bouton_effacer', 'Envoyer');
if ($form->validate()) {
    echo "Toutes les règles sont respectées<br>";
}
else {
    $form->display();
}
?>
</body>
</html>
    
```

IV-E - Récupérer les données

C'est bien beau de saisir des valeurs, encore faut-il en faire quelque chose !! Il est bien sur possible d'utiliser les tableaux **\$_POST** ou **\$_GET** mais nous allons voir qu'il existe d'autres méthodes. Essayons le code suivant, à ajouter dans la boucle if (`$form->validate()`)

```

echo '---' . $_POST['Pseudo'] . '---<br>';
echo '---' . $form->exportValue('Pseudo') . '---';
    
```

Maintenant saisissez un pseudo comportant des espaces à la fin. Vous vous apercevez que la méthode **exportValue** renvoie la valeur **saisie filtrée**, c'est dire sans les espaces de fin. Cette différence peut vous paraître mineure mais elle est très importante. En effet nous verrons dans un autre tutoriel que l'on peut définir ses propres filtres donc supprimer des valeurs saisies les caractères indésirables comme par exemple < et >.

Il est également possible de **définir une fonction qui va traiter les valeurs saisies**. Pour cela il faut utiliser la méthode **process()** du formulaire. Exemple :

```

if ($form->validate()) {
    $form->process('traiteDonnees', false);
}
else {
    $form->display();
}
    
```

et bien sur d'écrire le code de la fonction `traiteDonnees()` qui pourrait être :

```

function traiteDonnees ($values) {
    
```

```
echo "<pre>";  
var_dump($values);  
echo "</pre>";  
}
```

V - Présentation rapide des autres pseudo-éléments

Dans notre premier exemple, nous avons vu l'utilisation des éléments de type text. Bien sur vous pouvez utiliser tous les autres éléments standard en HTML à savoir : button, checkbox, select, textarea, radio, reset, submit, image, textarea, password et file. Mais comme nous l'avons vu avec l'élément date de notre exemple, le package HTML_quickform propose aussi des **pseudo-éléments** qui offrent d'intéressantes possibilités. J'ai choisi de ne pas tous vous les présenter mais de détailler l'utilisation des éléments les plus intéressants (à mon avis).

V-A - Advcheckbox

Cet élément est une **amélioration de l'élément checkbox** standard. En effet une checkbox standard non seulement ne retourne pas de valeur si elle n'est pas cochée mais la variable elle-même n'existe pas. Avec un élément advcheckbox la variable existe même si la case n'est pas cochée.

```
$form->addElement('advcheckbox',
    'pomme', // nom de l'element
    'Votre fruit préféré', // texte avant l'élément
    'pomme', // texte apres l'élément
    '', // attributs
    array('sans pomme', 'avec pomme')); // valeur retournée case non cochée et case cochée
```

Le dernier paramètre est optionnel et peut être de différentes formes. S'il n'est pas renseigné, la valeur de la variable est 0 si la case n'est pas cochée ou 1 si elle l'est. Si ce dernier paramètre est une chaîne de caractères, la valeur de la variable sera une chaîne vide si la case n'est pas cochée et la valeur du paramètre si elle l'est. Enfin, comme dans l'exemple ci-dessus, ce paramètre peut être un tableau de 2 valeurs dont la première correspond à la case non cochée et la deuxième à la case cochée.

V-B - Hierselect

Ce pseudo-élément crée en fait 2 éléments de type select. **Les choix possibles dans le deuxième select dépendent de la valeur du premier select.** Exemple :

```
$marque = array();
$modelle = array();

$marque[0] = "Renault";
$marque[1] = "Peugeot";
$marque[2] = "Citroen";

$modelle[0][0] = "Scenic";
$modelle[0][1] = "Laguna";
$modelle[0][2] = "Velsatis";
$modelle[1][3] = "407";
$modelle[1][4] = "607";
$modelle[2][5] = "Xsara";
$modelle[2][6] = "Picasso";
$modelle[2][7] = "C5";

$sel =& $form->addElement('hierselect', 'voiture', 'Voiture : ');
$sel->setMainOptions($marque);
$sel->setSecOptions($modelle);
```

 **Attention:** cet élément fait appel à des fonctionnalités javascript.

Pour récupérer la marque et le modèle sélectionnés, il suffit de faire dans la fonction `traiteDonnees()` :

```
echo "marque =" . $marque[$values['voiture'][0]];
echo "modele =" . $modele[$values['voiture'][0]][$values['voiture'][1]];
```

V-C - File

Ce type d'élément existe en standard en HTML mais ce qui est intéressant avec quickform c'est le fait que l'on puisse ajouter des règles sur la taille maximum du fichier et sur son type ;

Exemple :

```
$file =& $form->addElement('file', 'filename', 'File:');
$form->addRule('filename', 'Vous devez choisir un fichier', 'uploadedfile');
$form->addRule('filename', 'Le fichier choisi est trop gros', 'maxfilesize', 524288);
$form->addRule('filename', 'Vous devez choisir un JPEG', 'mimetype', array('image/jpeg',
'image/jpeg') );
```

Ce code mérite quelques explications.

```
$file =& $form->addElement('file', 'filename', 'File:');
```

Permet de récupérer la variable retournée par la fonction `addElement`. Cette variable sera utilisée plus loin lors de l'upload.

```
$form->addRule('filename', 'Vous devez choisir un fichier', 'uploadedfile');
$form->addRule('filename', 'Le fichier choisi est trop gros', 'maxfilesize', 524288);
$form->addRule('filename', 'Vous devez choisir un JPEG', 'mimetype', array('image/jpeg',
'image/jpeg') );
```

On ajoute **trois règles** qui sont spécifiques à l'élément file. La règle **uploadedfile** oblige l'utilisateur à choisir un fichier. La règle **maxfilesize** fixe la taille maximum du fichier (en octet). La règle **mimetype** spécifie le type de fichier qui doit être choisi (dans notre cas un fichier de type image jpeg).

Pour uploader le fichier il faut faire (une fois les données validées) :

```
if ($file->isUploadedFile()) {
    $file->moveUploadedFile('c:/tmp');
}
else {
    echo "Fichier non téléchargé";
}
```

Ainsi le fichier sera copié dans le répertoire `c:\tmp` du serveur. **Exemple complet** d'utilisation de l'élément file :

Script 4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>formulaire</title>
</head>
<body>
<?php
```

Script 4

```

set_include_path(get_include_path() . ";c:\php\pear");
require_once "HTML/QuickForm.php";

$form = new HTML_QuickForm('frmTest', 'post');
$form->addElement('hidden', 'path', 'c:/tmp');
$file =& $form->addElement('file', 'filename', 'File:');
$form->addRule('filename', 'Vous devez choisir un fichier', 'uploadedfile');
$form->addRule('filename', 'Le fichier choisi est trop gros', 'maxfilesize', 524288);
$form->addRule('filename', 'Vous devez choisir un JPEG', 'mimetype', array('image/jpeg',
'image/jpeg') );
$form->addElement('reset', 'bouton_clear', 'Effacer');
    $form->addElement('submit', 'bouton_effacer', 'Envoyer');
if ($form->validate()) {
    if ($file->isUploadedFile()) {
        $file->moveUploadedFile('c:/tmp');
    }
    else {
        echo "Fichier non téléchargé";
    }
}
else {
    $form->display();
}
?>
</body>
</html>
    
```

V-D - Select

Comme l'élément file, cet élément existe en standard en HTML. Mais il propose une fonction intéressante qui est de générer les options de choix directement à partir d'une table d'une base de données.

Attention: pour être opérationnelle, cette fonction nécessite que le **package DB** soit installé.

Supposons que nous ayons une table nommée chanteur dans une base de données nommée mabase. Pour vous faciliter la tâche, voici le code mysql permettant de créer cette table :

```

CREATE TABLE `chanteur` (
  `id` tinyint(4) NOT NULL default '0',
  `chanteur` varchar(100) NOT NULL default ''
) TYPE=MyISAM;

#
# Contenu de la table `chanteur`
#

INSERT INTO `test` VALUES (0, 'Brel');
INSERT INTO `test` VALUES (1, 'Brassens');
INSERT INTO `test` VALUES (2, 'MC Solaar');
INSERT INTO `test` VALUES (3, 'Ferré');
    
```

Nous allons créer un élément de type select dont les options de choix sont le **contenu de la table chanteur** :

```

$user = 'root';
$password = '';
$mysqlServer = 'localhost';
$dataBaseName = 'mabase';
$dsn = 'mysql://'. $user . ':' . $password . '@'. $mysqlServer . '/' . $dataBaseName;
$form = new HTML_QuickForm('frmTest', 'post');
$chanteurSelect = & $form->addElement('select', 'chanteur', 'Chanteur : ');
$chanteurSelect->load($result) ;
    
```

```
$chanteurSelect->loadQuery($dsn, 'select * from test', 'chanteur', 'id');
```

Bien sûr, vous changerez vos noms de serveur, d'utilisateur et votre mot de passe.

Et c'est tout ! Sympa non ?

V-E - Header

J'ai gardé le plus facile pour la fin. Le pseudo-élément **header** permet d'ajouter un entête au formulaire. En général cet élément est le premier ajouté.

Exemple :

```
$form->addElement('header', 'MonEnTete', 'Vos coordonnées');
```

VI - Conclusion

Voilà pour cette première partie. Ce tutoriel n'avait d'autres intentions que de vous mettre l'eau à la bouche et vous donner envie de découvrir toutes les possibilités qu'offre ce package. Par exemple il est possible de grouper des éléments pour qu'ils apparaissent sur une même ligne, d'ajouter des éléments dynamiquement (d'après des données extraites d'une base de données par exemple), de spécifier des valeurs par défaut, et de coupler l'utilisation de HTML_quickform avec des moteurs de templates. Alors bon courage ...

