

Création d'un plug-in pour WordPress

Table des matières

I.	Plug-in de base	3
1.	Étude rapide du plugin Hello Dolly	3
2.	Structure de fichier	5
3.	Création de plug-in simple helloworld.php	5
4.	Création du plug-in helloworld.php sous forme de Widget	5
5.	Ajouter un CSS à notre plug-in	7
II.	Utiliser une base de données	8
1.	Création d'une table lors de l'installation	8
2.	Stocker les données du widget dans la table	9
III.	Ajouter des champs de formulaire pour paramétrer le plug-in	10
1.	Ajouter un titre au plug-in	10
2.	Ajouter une page de paramètre au plug-in	10
	Ajouter des menus	10
	Créer des options	11

I. Plug-in de base

Les plug-in sont des extensions légères et flexibles. Ils sont utilisés pour de petites parties de page qui sont généralement moins complexes et peuvent être visibles à travers les différents composants.

Vous pouvez voir de nombreux exemples de plug-in dans une installation standard de WordPress : - menus - dernières actualités - formulaire de connexion - et bien d'autres.

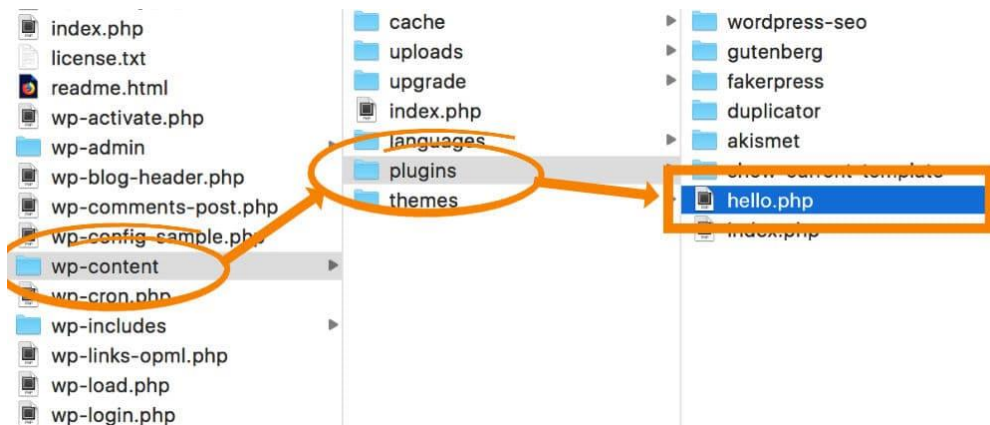
Ce didacticiel vous explique comment créer un simple plug-in Hello World. Grâce à lui, vous apprendrez la structure de fichiers de base d'un module. Cette structure de base peut ensuite être étendue afin de créer des modules plus élaborés.

1. Étude rapide du plugin Hello Dolly

Si vous avez envie de **créer votre premier plugin**, vous pouvez déjà commencer par étudier le fameux plugin **"Hello Dolly"** qui est embarqué avec toute installation WordPress.

En effet, ce plugin n'est pas là par hasard – *même si on a tendance à le supprimer automatiquement après l'installation de WordPress* – car il est l'exemple même qui nous montre le chemin pour **réaliser facilement sa propre extension**.

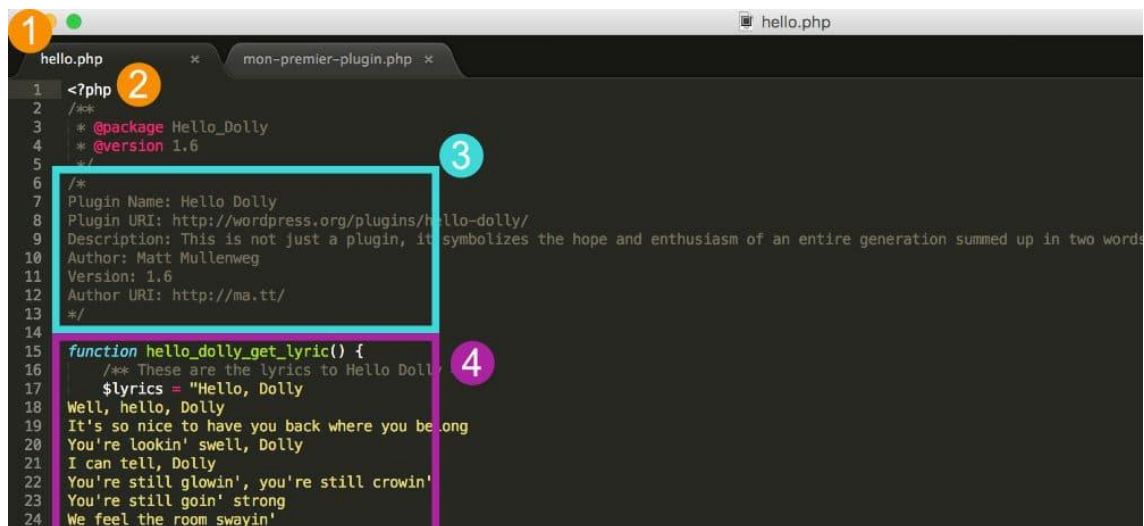
La première des choses qui saute aux yeux, lorsqu'on regarde le dossier **"PLUGINS"** situé dans le répertoire **"WP-CONTENT"**, c'est que **Hello Dolly** n'est pas placé dans un dossier, contrairement aux autres extensions.



Cela nous montre bien qu'un simple plugin peut être composé d'un seul **fichier.php** et n'a pas besoin d'autre chose comme une feuille de style.css ou des fichiers JavaScript pour fonctionner.

WordPress permet donc d'ajouter un simple fichier contenant une fonctionnalité directement dans le dossier **WP-CONTENT > PLUGINS**.

Mais attention, ce "simple fichier" doit suivre les guidelines de WordPress afin d'être exploitable, ouvrons le fichier **hello.php** pour comprendre et voir ce qu'il contient...



```
1 <?php
2 /**
3  * @package Hello_Dolly
4  * @version 1.6
5  */
6
7 Plugin Name: Hello Dolly
8 Plugin URI: http://wordpress.org/plugins/hello-dolly/
9 Description: This is not just a plugin, it symbolizes the hope and enthusiasm of an entire generation summed up in two words:
10 Author: Matt Mullenweg
11 Version: 1.6
12 Author URI: http://ma.tt/
13 */
14
15 function hello_dolly_get_lyric() {
16     /** These are the lyrics to Hello Dolly
17     $lyrics = "Hello, Dolly
18 Well, hello, Dolly
19 It's so nice to have you back where you belong
20 You're lookin' swell, Dolly
21 I can tell, Dolly
22 You're still glowin', you're still crowin'
23 You're still goin' strong
24 We feel the room swayin'
```

Comme cette capture d'écran ci-dessus nous le montre, le plugin **Hello Dolly**, comme tous les autres plug-ins, est composé de :

1. Un fichier PHP : **nom.php**, dans ce cas il s'agit de **hello.php**
2. Une ouverture de balise PHP (**<?php**). Pas besoin d'ajouter une balise de fermeture car ce n'est pas obligatoire, contrairement au HTML.
3. Un entête placé entre **/* et */** qui paraît être un commentaire anodin mais vous allez voir que ce n'est pas le cas.
4. Le code PHP qui permet d'ajouter une fonctionnalité à votre site.

C'est tout ! Et vous voyez que ce plugin est fonctionnel. Maintenant, penchons-nous sur cette fameuse entête qui est un élément obligatoire et primordial...



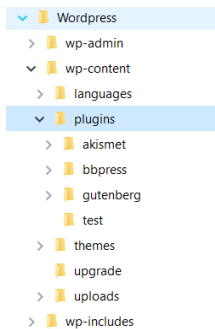
Si on regarde attentivement la capture d'écran ci-dessus, vous allez voir à quoi correspondent chaque ligne de l'entête d'un plugin.

Les **pastilles rouges** numérotent les lignes présentes dans le **fichier.php** et les **pastilles bleues** sont leur correspondance dans le back-office.

1. **la ligne *Plugin Name*:** => elle correspond au nom du plugin tel qu'il sera affiché dans la partie administrateur. On voit bien que celui-ci n'est pas obligatoirement le même que le nom du fichier, qui lui est nommé **hello.php**.
2. **la ligne *Plugin URI*:** => elle correspond à l'URL du plugin sur le repository (répertoire officiel), si ce n'est pas le cas, vous pouvez insérer ici l'URL de votre choix.
3. **la ligne *Description*:** => elle permet de décrire la fonctionnalité du plugin.
4. **la ligne *Author*:** => elle donne l'identité du créateur du plugin. C'est l'ancre qui apparaîtra sur le lien du site de l'auteur (pastille 6).
5. **la ligne *Version*:** => elle permet d'afficher le numéro de version du plugin. À chaque nouvelle version, ce numéro évoluera.
6. **la ligne *Author URI*:** => elle fait le lien avec le site de l'extension.

À noter : si vous regardez bien, il n'y a pas d'espace avant les "deux points", veillez à respecter cela.

2. Structure de fichier



Dans un souci de clarté, nous créerons toujours un dossier contenant notre plug-in. Par convention, nous nommerons le dossier ainsi que le premier fichier par le nom de notre plug-in.

3. Création de plug-in simple helloworld.php

Dans un premier temps, créer un dossier HelloWorld, puis à l'intérieur un fichier HelloWorld.php.

Insérer le code suivant

```
<?php
/*
Plugin Name: Hello World
Description: Ecrit Hello World en haut de la page
Author: Martine
Version: 1.0
*/
echo "<h1>Hello world</h1>";
```

Rafraîchir dans le module admin, la page extension. Le plug-in HelloWorld Apparaît. Activer le et rafraîchir le site.

4. Création du plug-in helloworld.php sous forme de Widget

Afin de bien scinder les parties, nous allons travailler avec des classes

Le fichier `helloworld.php` complet se présente ainsi :

```
<?php
/*
Plugin Name: Hello World
Description: écrit Hello World
Author: Martine
Version: 1.0
*/

class HelloWorld_Plugin{
    public function __construct(){
        include_once plugin_dir_path(__FILE__).'./helloClass.php';
        new HelloClass();
    }
}
new HelloWorld_Plugin();
```

Il inclut et instancie la classe.

Le fichier `helloclass.php` complet se présente ainsi :

```
<?php
//on inclu la definition du widget
include_once plugin_dir_path( __FILE__ ).'./helloworld.php';

class HelloClass{
    public function __construct(){
        // on déclare le widget
        add_action('widgets_init', function(){register_widget('HelloWidget');});
    }
}
```

Le fichier `helloworld.php` complet se présente ainsi :

```
<?php
class helloworld extends WP_Widget
{
    public function __construct()
    {
        parent::__construct('helloworld', 'Hello World', array('description' => 'Un plug-in
qui écrit Hello World'));
    }
    public function widget($args, $instance)
    { // formulaire afficher à l'écran pour l'utilisateur

        // on appel les méthodes standard au cas où un autre plug-in les aurait surchargées
        echo $args['before_widget'];
        echo $args['before_title'];
        echo apply_filters('widget_title', $instance['title']);
        echo $args['after_title'];
        // corps du widget
        ?>
        <h1>Hello World</h1>
        <?php
        echo $args['after_widget'];
    }
}
```

L'architecture est la suivante, le plug-in instancie la classe qui fait appel au widget.

Le widget apparait à gauche de l'écran d'admin sous Apparence / Widget. Vous pouvez maintenant le placer sur votre site

5. Ajouter un CSS à notre plug-in

Dans le fichier `helloclass.php` ajouter l'action qui permet de charger le CSS en même temps que le CSS de Wordpress, dans le construct

```
add_action('wp_enqueue_scripts', array($this, 'persoCSS'), 15);
```

Puis ajouter la fonction qui permet de lier votre CSS.

```
function persoCSS()
{
    wp_enqueue_style('Hellocss', plugins_url('helloworld/design.css'));
}
```

II. Utiliser une base de données

1. Création d'une table lors de l'installation

Pour créer la table contenant les données du plug-in, nous devons exécuter une requête SQL spécifique. Dans WordPress, l'accès à la base de données se fait à l'aide de la classe `wpdb`, qui contient de nombreuses méthodes pour interagir avec elle. Notamment, une méthode `query()` est destinée à exécuter les requêtes qui lui sont passées en paramètre.

Une instance de la classe `wpdb` est créée au chargement de l'application et stockée dans une variable globale. Sa récupération se fait donc depuis n'importe quel endroit du code de façon très simple :

```
<?php
global $wpdb;
```

Afin de créer une table lors de l'installation, nous allons ajouter la fonction suivante dans `helloClass.php` :

```
public static function install()
{
    //méthode déclenchée à l'activation du plug-in
    global $wpdb;
    $wpdb->query("CREATE TABLE IF NOT EXISTS {$wpdb->prefix}helloworld_commentaire (id INT
    AUTO_INCREMENT PRIMARY KEY, comm VARCHAR(255) NOT NULL);");
}
```

La table contiendra un id et un commentaire.

Et créer l'accroche dans le construct de `HelloWorld.php`

```
register_activation_hook(__FILE__, array('helloClass', 'install'));
```

Si votre plug-in est déjà actif, désactiver le et activer le de nouveau. La table apparaît dans la base de données.

Nous devons penser à la désinstallation du plugin. En effet, si un utilisateur décide de supprimer le plugin de son installation WordPress, il faut aussi que la table que nous avons créée soit effacée de la base de données pour que celle-ci retrouve son état original.

De même que pour l'installation, créons une fonction qui va cette fois supprimer la table.

```
public static function uninstall()
{
    //méthode déclenchée à la suppression du module
    global $wpdb;
    $wpdb->query("DROP TABLE IF EXISTS {$wpdb->prefix}helloworld_commentaire;");
}
```

Et l'accroche correspondante dans le construct

```
register_deactivation_hook(__FILE__, array('helloClass', 'uninstall'));
```


2. Stocker les données du widget dans la table

Modifier le widget pour y mettre une zone de saisie de texte. Insérer ce code dans le corps du widget.

```
<form action="" method="post">
  <p>
    <label for="helloworld_comm">Votre commentaire :</label>
    <input id="helloworld_comm" name="helloworld_comm" type="texte"/>
  </p>
  <input type="submit"/>
</form>
```

Ajouter la fonction de sauvegarde dans la classe

```
public function save_comm()
{
    if (isset($_POST['helloworld_comm']) && !empty($_POST['helloworld_comm']))
    {
        global $wpdb;
        $comm = $_POST['helloworld_comm'];
        $row = $wpdb->get_row("SELECT * FROM {$wpdb->prefix}helloworld_commentaire WHERE
                               comm = '$comm'");

        if (is_null($row)) {
            $wpdb->insert("{ $wpdb->prefix }helloworld_commentaire", array('comm' =>
                                                                              $comm));
        }
    }
}
```

```
// on ajoute l'action de sauvegarde au chargement du widget
add_action('wp_loaded', array($this, 'save_comm'));
```

Les saisies faites par l'utilisateur sont stockées dans la base, si elle n'existe pas

III. Ajouter des champs de formulaire pour paramétrer le plug-in

1. Ajouter un titre au plug-in

Vous avez probablement remarqué que le widget n'a aucun paramètre disponible dans l'interface d'administration. Pourtant, il faudrait au minimum pouvoir définir un titre à afficher en haut du widget pour qu'il soit identifiable parmi les autres.

L'affichage du paramétrage du widget est complètement délégué à notre classe en surchargeant la méthode `form()`, qui prend en paramètre un tableau contenant les valeurs précédemment enregistrées.

Dans le formulaire que nous allons créer, il est important d'utiliser, pour la génération des attributs `id` et `name` de vos champs, deux méthodes définies par `WP_Widget` qui sont respectivement `get_field_id()` et `get_field_name()`. Ces deux méthodes vont générer un identifiant et un nom unique pour chaque champ qui utilisés par WordPress lors de la sauvegarde des valeurs, il est donc très important de les utiliser, sans quoi l'enregistrement des paramètres ne pourra pas fonctionner.

Voici donc comment nous allons définir la fonction `form()` dans le fichier `helloworldwidget.php`.

Ajouter le titre avec la fonction `form`

```
public function form($instance)
// formulaire de gestion des paramètres pour le module d'administration
{
    $title = isset($instance['title']) ? $instance['title'] : '';
    ?>
    <p>
    <label for="<?php echo $this->get_field_name( 'title' ); ?>"><?php _e( 'Title:' );
?></label>
    <input class="widefat" id="<?php echo $this->get_field_id( 'title' ); ?>" name="<?php
echo $this->get_field_name( 'title' ); ?>" type="text" value="<?php echo $title; ?>" />
    </p>
    <?php
}
```

2. Ajouter une page de paramètre au plug-in

Les fonctionnalités de notre plugin sont presque complètes ! Il nous reste cependant un point important à finaliser : le plugin doit pouvoir être configuré dans l'administration, notamment pour lui définir des options. Il nous faut donc ajouter un menu spécifique pour la gestion du module, dans lequel nous disposerons des paramètres éditables.

Ajouter des menus

Commençons par créer un élément de premier niveau qui apparaîtra dans le menu principal de l'administration, c'est-à-dire directement dans la colonne de gauche. Cet ajout se fait lors du chargement des menus de WordPress, un événement identifié par le déclenchement de l'action `admin_menu`. Dans le constructeur de la classe `HelloWorld_Plugin`, il nous faut donc brancher une fonction `add_admin_menu()` que l'on définira plus bas.

```
add_action('admin_menu', array($this, 'add_admin_menu'),20);
```

Puis créer les fonctions `add_admin_menu` et `menu_html`

La création d'un menu s'effectue avec la fonction `add_menu_page()`, qui peut prendre jusqu'à sept paramètres :

- le titre de la page sur laquelle nous serons redirigés ;
- le libellé du menu ;
- l'intitulé des droits que doit posséder l'utilisateur pour pouvoir accéder au menu. Si les droits sont insuffisants, le menu sera masqué ;
- la clé d'identifiant du menu qui doit être unique (mettre le nom du plugin est une bonne option) ;
- la fonction à appeler pour le rendu de la page pointée par le menu ;
- l'icône à utiliser pour le lien (vous pouvez laisser les valeurs par défaut) ;
- la position dans le menu (vous pouvez laisser les valeurs par défaut).

```
public function add_admin_menu()
{
    //on ajoute une page dans le menu administrateur
    add_menu_page('Hello World', 'Hello World plugin', 'manage_options',
        'helloworld', array($this, 'menu_html'));
}
public function menu_html() {
    echo '<h1>'.get_admin_page_title().</h1>';
    echo '<p>Bienvenue sur la page d\'accueil du plugin</p>';
}
```

Créer des options

Notre page est maintenant créée, il nous faut donc écrire le formulaire permettant de renseigner les différentes options de notre choix via la méthode `menu_html()`.

Le fonctionnement des options

Pour conserver un maximum de souplesse, WordPress inscrit un grand nombre de valeurs de configuration dans la base de données, dont un certain nombre sont éditables au travers du menu « Réglages » de l'administration. Ce fonctionnement permet d'avoir des attributs facilement modifiables par l'administrateur pour ne pas avoir à modifier le code PHP lorsqu'il désire modifier des paramètres comme le titre du site, le format des dates, la taille des médias...

Ces différents paramètres sont appelés des options et sont stockés dans la base de données dans la table `wp_options`, l'identifiant de l'option étant rangé dans la colonne `option_name` et sa valeur dans `option_value`. Ainsi, rien ne nous empêche de définir nos propres valeurs de configuration, notamment lorsque l'on veut pouvoir paramétrer un plugin. Il suffit pour cela d'ajouter une ligne à la table avec l'identifiant et la valeur désirés.

Pour récupérer la valeur d'une option, il faut utiliser la fonction `get_option()` en lui indiquant l'identifiant de l'option à récupérer. Si l'option n'existe pas, la fonction retourne `false` ou bien la valeur du second paramètre s'il a été fourni.

L'ajout d'une option dans la base de données se fait avec la méthode `add_option()`, à laquelle il faut envoyer l'identifiant ainsi que la valeur de l'option. Toutefois, cette fonction ne permet pas de mettre à jour une option déjà existante, il faut alors utiliser `update_option()`, dont les paramètres sont identiques. Notez qu'en utilisant `update_option()`, l'option sera créée si elle n'existe pas encore. Elle est donc utilisée dans la majorité des cas car elle convient aux deux usages.

Le formulaire

Lorsque l'on crée un formulaire destiné à enregistrer des données dans la table des options, celui-ci doit appeler le fichier `wp-admin/options.php` lors de la soumission des données. Commencez donc à compléter la méthode `Helloclass::menu_html()` comme ceci :

```
public function menu_html()
{
    echo '<h1>'.get_admin_page_title().'</h1>';
    ?>
    <form method="post" action="options.php">
        <label>Couleur</label>
        <input type="text" name="helloworld_couleur" value="<?php echo
                                get_option("helloworld_couleur")?>" />
        <?php submit_button(); ?>
    </form>
    <?php
}
```

Le formulaire doit maintenant s'afficher correctement, mais si vous l'envoyez, vous verrez qu'aucune valeur n'est sauvegardée dans la base de données (le champ apparaît vide lorsque vous revenez sur la page). Ceci est dû au fait que nous n'avons pas encore autorisé WordPress à enregistrer la valeur de l'option.

Enregistrer les options modifiables

La première chose à rajouter à l'intérieur du formulaire est un appel à la fonction `settings_fields()`, qui affiche un certain nombre de champs cachés permettant notamment à l'application de savoir à quel groupe d'option les champs que vous allez rajouter appartiennent.

```
<?php settings_fields('helloworld_settings') ?>
```

Il nous faut ensuite enregistrer le champ `helloworld_couleur` dans le groupe d'options. Ceci doit impérativement se faire lorsque le système d'administration est initialisé, c'est-à-dire au déclenchement de l'événement `admin_init`. Ajoutons donc une fonction `register_settings` dans la classe `Helloclass`, ainsi que l'enregistrement de l'action dans son constructeur.

```
add_action('admin_init', array($this, 'register_settings'));
```

```
public function register_settings()
{
    register_setting('helloworld_settings', 'helloworld_couleur');
}
```

Vous pouvez maintenant définir une valeur pour la couleur, et vérifier que celle-ci est bien sauvegardée. (Présente au rechargement)

En modifiant légèrement le widget, on peut appliquer cette couleur à la phrase d'entrée du plug-in, en utilisant la fonction `get_option`

```
// corps du widget
$couleur = get_option('helloworld_couleur', 'white');
?>
<div id="test" style="color:<?php echo $couleur;?>">Hello World est un plug-in qui enregistre
les commentaires en base de données</div>
```