

Objectif de l'extension

Dans ce chapitre, nous allons créer une extension "évoluée", autrement dit une extension ajoutant plusieurs fonctionnalités et faisant appel à différentes API de WordPress.

Ici, nous allons mettre au point une extension permettant de gérer une liste d'événements. Cette extension, nommée Simple Events, gère différentes fonctionnalités que nous détaillerons dans la section suivante.

C'est typiquement le genre d'extension qui pourra être utilisée dans le cadre d'une association par exemple. Une orientation qui tend plus vers le CMS que l'outil de blog...

Quelles fonctionnalités ?

Avant d'écrire ce code, il est impératif que vous vous posiez quelques questions. Ces questions vont permettre de regrouper les fonctionnalités de votre extension dans différents groupes, selon le type de fonctions sur lesquelles vous travaillerez, mais elles doivent également rappeler les différentes pages de l'extension.

Voici une liste non exhaustive des questions qui peuvent être importantes :

- Quelles sont les fonctionnalités ?
- Quelles sont les pages indispensables de l'interface d'administration ?
- Quels utilisateurs de WP vont pouvoir modifier les réglages de votre extension ?
- Quels utilisateurs de WP vont pouvoir utiliser l'extension ?
- Comment s'affichera le contenu de l'extension dans le thème ?
- Quelle stratégie vis-à-vis du référencement ?
- Où stocker les données ?
- Quelles fonctionnalités seront disponibles depuis la page de rédaction ? Etc.

Cette liste est un simple exemple. Avec l'expérience que vous allez acquérir, vous ajouterez probablement de nouvelles questions au fur et à mesure de vos développements.

Revenons à notre exemple Simple Events, et répondons à ces questions :

Quelles sont les fonctionnalités ?

L'extension doit pouvoir gérer l'ajout, l'édition et la suppression d'événements, avec la possibilité de classer les événements *via* des tags. L'extension contiendra également un widget pour afficher les derniers événements, et vous disposerez de deux vues côté thème, avec une

page pour les tags d'événements permettant de lister les extraits des événements et un fichier détaillé par événement. Soit quelque chose de semblable aux articles et aux tags de WordPress.

Quelles sont les pages indispensables de l'interface d'administration ?

Trois pages sont à prévoir : une page de gestion listant les événements, une page contenant le formulaire d'ajout et d'édition des événements et une page pour les réglages de l'extension.

Quels utilisateurs de WP vont pouvoir modifier les réglages de votre extension ?

Seul le rôle administrateur aura la possibilité de modifier les réglages de l'extension.

Quels utilisateurs de WP vont pouvoir utiliser l'extension ?

Les rôles administrateur et éditeur auront la possibilité d'utiliser l'extension, soit ajouter, effacer et modifier les événements.

Comment s'affichera le contenu de l'extension dans le thème ?

Pour afficher les événements dans le thème de l'utilisateur, vous disposez de différents moyens. Tout d'abord vous avez un widget qui affiche les derniers événements ajoutés ; il y a également deux templates dans le thème pour permettre l'affichage des événements et des catégories d'événements. Enfin, une fonction permet l'insertion du nuage de tags des événements dans le thème.

Quelle stratégie vis-à-vis du référencement ?

L'extension doit être capable de s'adapter à la forme des liens de WordPress. Si les permaliens sont actifs, l'extension doit générer de belles adresses, sinon elle produira des adresses classiques avec des paramètres.

Où stocker les données ?

Pour stocker les événements, il sera nécessaire de créer une base de données.

Quelles fonctionnalités seront disponibles depuis la page de rédaction ?

La page de rédaction offre deux fonctionnalités depuis le contenu des articles. Vous disposerez d'un marqueur dans le texte de l'article, pour afficher la liste des derniers événements ajoutés, et d'un shortcode pour afficher un lien avec une mise en page spécifique vers la vue détaillée d'un événement.

Regroupement des fonctionnalités

Une fois que vous avez répondu à ces différentes questions, vous devez regrouper les données par entité logique de WordPress. Une entité logique correspond en fait à un regroupement de fonctionnalités selon les bibliothèques de fonctions utilisées ; par exemple les accès en base de données seront gérés *via* l'API WPDB.

Administration

- ajout des menus dans WordPress ;
- création des nouvelles permissions et attributions aux rôles ;
- création du contenu des trois pages de l'extension.

Accès en base de données

- création de la base de données ;
- récupération d'événements depuis un tag donné ;
- ajout, édition, suppression d'un événement.

Réécriture d'adresse Internet – Rewriting

- initialisation du rewriting, construction de la base des URL ;
- génération des règles de rewriting.

Fonction du thème

- fonction du nuage de tags.

Widget

- création de l'option permettant d'activer ou non le widget ;
- création du widget.

Filtre du contenu des articles

- utilisation d'un marqueur ;
- utilisation des shortcodes pour afficher le lien d'un événement.

Taxinomie

- enregistrement de la taxinomie ;
- ajout du mot-clef dans WordPress ;
- contrôle de la requête de WordPress pour trouver le mot-clef ;
- redirection sur le bon template si on est sur la page des événements ou des tags d'événements ;
- création de la fonction de callback pour le compteur des tags.



Une fonction de callback est appelée de manière implicite par le logiciel. Par exemple, lors de l'ajout d'un tag à un article, le compteur du tag est automatiquement incrémenté. Pour y parvenir, il y a une fonction de callback qui ne fait que mettre à jour le compteur.

Architecture de l'extension

Fichier ou dossier ?

Pour architecturer une extension WordPress, deux cas de figure se présentent.

Le premier cas, c'est celui d'une extension simple comme Hello Dolly, que vous pouvez retrouver par défaut dans WordPress. Comme vous avez pu le constater, cette extension ne fait pas grand-chose, de ce fait placer toutes les fonctions dans un seul et même fichier est largement compréhensible.

Le second cas, à l'instar de notre exemple ici, est celui d'une extension évoluée, susceptible de contenir différents fichiers pour la traduction, un widget, une documentation ou encore des modèles de templates. Pour cette raison, il est nécessaire de créer un dossier pour ce type d'extension.

Un gros fichier ? Ou plusieurs petits fichiers ?

Dans le cadre d'une extension assez complexe, il y a deux façons de répartir le code. Soit vous placez tout le code dans un seul fichier PHP. Soit vous découpez l'extension par entité logique pour créer un fichier PHP par entité. Les deux techniques ont leurs avantages et leurs défauts.

Par exemple, si vous choisissez de placer toutes les fonctions dans un seul et même fichier, il sera difficile de trouver rapidement une fonction. Une extension un tant soit peu complexe atteindra très rapidement les 1 000 voire 2 000 lignes. D'un autre côté, même avec un grand nombre de fonctions, si ces dernières sont réparties dans différents fichiers thématiques, il sera assez facile de trouver celles qui vous intéressent.

Néanmoins, il n'est pas toujours possible de découper toutes les fonctions dans différents fichiers, pour la simple et bonne raison qu'un développeur peut travailler avec des classes PHP, et ces dernières ne peuvent pas être réparties sur plusieurs fichiers. Dans ce cas précis, la répartition pourra être la suivante : une classe cliente lancée tout le temps, et une classe admin ne démarrant que les fonctions spécifiques à l'interface d'administration seulement lorsque vous vous y trouvez.

Architecture de Simple Events

Ici, pour simplifier le code, l'extension sera développée *via* des fonctions classiques. De ce fait, l'extension sera architecturée de la façon suivante :

```
simple-events/  
  simple-events.php  
  languages/  
    event-fr_FR.mo  
    event-fr_FR.po  
  inc/  
    event.admin.php  
    event.db.php  
event.install.php  
event.rewriting.php  
event.taxonomy.php
```

```
event.template.php
event.thecontent.php
event.widget.php
theme/
  event-detail.php
  event-tags.php
```

Toute l'extension est contenue dans le dossier simple-events. Ce dernier possède trois dossiers et un fichier PHP.

Le fichier PHP simple-events.php contient l'en-tête spécifique aux extensions WordPress. C'est lui qui gère l'inclusion des autres bibliothèques, autrement dit c'est le cœur de l'extension.

Les trois dossiers sont languages, inc et thème. Ils contiennent respectivement les fichiers de traduction de l'extension, les bibliothèques de fonctions, qui sont au nombre de sept, nécessaires à l'extension, et enfin deux modèles de template à placer dans le thème.

Développement de l'extension

Les bases de l'extension

L'en-tête de WordPress

Comme dans toute extension, il est nécessaire d'avoir l'en-tête des extensions WordPress.

Nous débuterons le fichier simple-events.php avec ce code :

```
/*
Plugin Name: Simple Events
Plugin URI: http://page-web-du-plugin-event.net/
Description: Ce plugin permettra de gérer une liste d'événements à venir
Version: 1.1
Author: Amaury BALMER
Author URI: http://www.wordpress-fr.net
*/
```

Cet en-tête peut être suivi de la licence de l'extension.

Inclusion des différents fichiers

Pour permettre l'inclusion des fonctions contenues dans les différents fichiers, vous devez ajouter les appels PHP des fichiers après l'en-tête de WordPress :

```
require( dirname(__FILE__) . '/inc/event.taxonomy.php' );
require( dirname(__FILE__) . '/inc/event.admin.php' );
require( dirname(__FILE__) . '/inc/event.db.php' );
require( dirname(__FILE__) . '/inc/event.install.php' );
require( dirname(__FILE__) . '/inc/event.rewriting.php' );
require( dirname(__FILE__) . '/inc/event.template.php' );
require( dirname(__FILE__) . '/inc/event.thecontent.php' );
```

Remarquez que nous ne précisons pas le chemin complet des fichiers PHP, nous utilisons à la place la fonction `dirname()` qui retourne le chemin complet du dossier contenant le fichier que nous passons en paramètre. Ici nous passons en paramètre la constante `PHP__FILE__` ; cette constante a comme valeur le chemin complet du fichier où la constante est appelée.

Activation de l'extension

Lors de l'activation de l'extension dans WordPress, nous allons devoir effectuer deux actions : créer la table dans la base de données MySQL où nous stockerons les données, et initialiser des permissions spécifiques à notre extension.

Pour lancer une fonction PHP lors de l'activation, nous utiliserons la fonction `register_activation_hook()` qui appellera automatiquement la fonction `event_install()` lors de l'activation de l'extension.

Ajoutez cette ligne :

```
// Installation du plugin (fichier event.install.php)
register_activation_hook( __FILE__, 'event_install' );
```

Cette fonction doit être placée après l'ajout des fichiers PHP.

Création de la table SQL

Le schéma de la table

Dans cette extension, le schéma de la table sera assez simple. Pour chaque événement, il sera nécessaire de stocker les informations suivantes :

- l'ID de l'événement ;
- le nom de l'événement ;
- la description de l'événement ;
- la date de l'événement.

Dans cette démonstration, nous stockons peu d'informations, mais rien ne vous empêche d'en stocker plus.

Génération du code SQL via phpMyAdmin

Pour élaborer la requête SQL de création de table, il y a deux méthodes. La première consiste à écrire la requête SQL à la main ; pour cela il faut une bonne connaissance du langage SQL. La seconde méthode ne nécessite pas une bonne connaissance de SQL et peut se traduire par un gain de temps appréciable. Pour cela, il est très pratique de créer sa table directement dans l'outil phpMyAdmin et d'exporter ensuite la structure de la table pour obtenir le code SQL.

Nous rappelons que phpMyAdmin est un outil libre et gratuit, disponible chez la quasi-totalité des hébergeurs, permettant de gérer ses bases de données MySQL. Pour trouver l'emplacement de l'outil chez votre hébergeur, consultez les e-mails reçus lors de la création du

compte ou connectez-vous sur l'interface de votre hébergeur. Il propose en général un lien vers l'outil.

Pour générer le code, vous devez procéder de la façon suivante.

Dans un premier temps, connectez-vous à l'outil phpMyAdmin. Une fois connecté, si vous avez le choix entre différentes bases de données dans le menu de gauche, sélectionnez la table où est installé WordPress.

À cette étape, la partie droite du logiciel devrait présenter la liste des tables que contient la base de données. En bas de cette partie se trouve un formulaire Créer une nouvelle table sur la base ; il suffit d'entrer un nom temporaire pour votre table, ici tmp_event, et de spécifier le nombre de champs de table, ici 10.

Vous pouvez indiquer un nombre de champs supérieur à la réalité ; cela ne pose pas de problème d'avoir un formulaire pas complètement rempli. Une fois les deux données saisies, cliquez sur le bouton Exécuter.

Vous obtenez un nouveau formulaire permettant de spécifier les champs de la table (voir Figure 11.01). Pour cette extension, vous devez compléter les quatre premières lignes avec les quatre informations que nous avons déjà mentionnées dans ce chapitre.

Figure 11.01

Création de la table MySQL depuis l'outil phpMyAdmin.

Champ	Type	Taille/Valeurs ¹	Interclassement	Attributs	Null	Défaut ²	Extra	Commentaires
id	INT	10			not null		auto_increment	
name	VARCHAR	255			not null			
description	TEXT				not null			
date_event	DATETIME				not null			
	VARCHAR				not null			
	VARCHAR				not null			
	VARCHAR				not null			
	VARCHAR				not null			
	VARCHAR				not null			
	VARCHAR				not null			
	VARCHAR				not null			

Commentaires sur la table:

Moteur de stockage: MyISAM Interclassement: 2

Sauvegarder Ou Ajouter 1 champ(s) Exécuter

L'ID de l'événement

- Le nom du champ : id.
- Le type : int ; il s'agit ici d'un nombre entier.
- La taille : 10 ; cela signifie que vous pouvez avoir jusqu'à 9 999 999 999 événements.
- Extra : auto_increment ; cela veut dire que l'ID de l'événement sera automatiquement incrémenté de 1 à chaque insertion.
- Sélection du bouton primaire, ce qui veut dire que l'ID est la clé unique de chaque événement.

Le nom de l'événement

- Le nom du champ : name.
- Le type : VARCHAR ; il s'agit ici d'une chaîne de caractères.
- La taille : 255 ; cela veut dire que le nom peut avoir jusqu'à 255 caractères.

La description de l'événement

- Le nom du champ : `description`.
- Le type : `TEXT` ; il s'agit ici d'une longue chaîne de caractères, il n'y a pas de contrainte de taille.

La date de l'événement

- Le nom du champ : `date_event`.
- Le type : `DATETIME` ; il s'agit du format de date MySQL utilisé par WordPress.

Une fois ces données saisies, envoyez les données du formulaire *via* le bouton Sauvegarder. Vous arrivez alors sur une page contenant la notification "Table ``tmp_event`` a été créé(e)". En dessous de cette notification, vous pouvez lire le code SQL ayant permis la création de la table.

Vous pouvez récupérer le code SQL ici, ou bien, si vous effectuez une modification sur la structure de la table, il vous suffit de cliquer sur l'onglet Exporter. Dans la page d'exportation, décochez la case Données et validez ensuite le formulaire *via* le bouton Exécuter.

Vous obtenez alors le code SQL permettant la création de la table, qui devrait s'apparenter à :

```
CREATE TABLE IF NOT EXISTS `tmp_event` (  
  `id` int(10) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL,  
  `description` text NOT NULL,  
  `date_event` datetime NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Notez que le texte de la dernière ligne, contenu entre la fermeture de la parenthèse, peut être supprimé, ainsi que le point-virgule ; il s'agit d'informations annexes pouvant générer des problèmes de compatibilité selon les serveurs SQL.

Enfin n'oubliez pas d'effacer la table temporaire lorsque vous aurez terminé de schématiser votre base de données.

Initialisation du nom de la table SQL

Nous avons vu au Chapitre 2 qu'il était possible de définir le préfixe des tables de la base de données.

Par défaut, le préfixe est `wp_`. Mais rien n'empêche l'utilisateur de modifier le préfixe, cela pour des raisons de sécurité ou bien simplement en raison de contraintes techniques, s'il a par exemple deux blogs et une seule base de données.

Ainsi, il faudra que l'extension préfixe également le nom de sa table. Pour cela, vous allez utiliser la variable globale `$wpdb` qui vous permettra de récupérer le préfixe employé :

```
global $table_event, $wpdb;  
$table_event = $wpdb->prefix . "events";
```

Vous allez donc stocker le nom de la table avec le préfixe dans la variable `$table_event`.

Par exemple, si le préfixe de table est wp1_, le nom de la table sera wp1_events.

Exécution de la requête SQL de création dans WordPress

Lors de l'activation, la fonction `event_install()` contenue dans le fichier `event.install.php` est automatiquement appelée.

N'oubliez pas de lire les commentaires pour comprendre la logique du code.

```
// Création de la table lors de l'installation
function event_install() {
    global $wpdb, $table_event;

    // On vérifie si la table n'existe pas
    if( $wpdb->get_var("show tables like '{$table_event}'") != $table_event ) {

        // On construit la requête SQL de création de table
        $sql =
            "CREATE TABLE " . $table_event . " (
                `id` INT( 10 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
                `name` VARCHAR( 255 ) NOT NULL,
                `description` TEXT NOT NULL,
                `date_event` DATETIME NOT NULL
            );";

        // Exécution de la requête
        require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
        dbDelta($sql);

    }

    // Initialisation des permissions
    event_init_permission();
}
```

Nous retrouvons bien dans cette fonction le code SQL de création de la table. Vous remarquerez que nous avons remplacé le nom de la table par une variable PHP dynamique, utilisant le nom de la table avec le préfixe de table.

Faites attention de ne pas entourer le nom de la table avec des simples ou doubles quotes, cela fausserait la requête de création.

Enfin, la fonction `event_install()` appelle la fonction `event_init_permission()`, permettant ainsi l'initialisation des permissions dans WordPress. Pour des raisons de lisibilité, le code des permissions sera contenu dans une fonction tierce.

Initialisation des permissions

L'extension disposera de deux permissions différentes : une première permettant de gérer les données de l'extension et l'autre permettant de modifier les réglages de l'extension.

Ici la permission de gestion sera nommée `use_event` et elle sera attribuée aux rôles éditeur et administrateur, tandis que la permission `admin_event` sera attribuée uniquement à l'administrateur, lui permettant de modifier les réglages :

```
// Ajout des permissions
function event_init_permission() {
    if ( function_exists('get_role') ) {

        // Je récupère l'objet "Rôle administrateur"
        $role = get_role('administrator');

        // Si la permission "use_event" n'existe pas, on l'ajoute.
        if( $role != null && !$role->has_cap('use_event') ) {
            $role->add_cap('use_event');
        }

        // Pareil pour la permission "admin_event"
        if( $role != null && !$role->has_cap('admin_event') ) {
            $role->add_cap('admin_event');
        }

        // On supprime la variable de notre fonction.
        unset($role);

        // On procède de la même façon pour le rôle "Editeur" sauf qu'on lui ajoute
        // uniquement la permission "user_event"
        $role = get_role('editor');
        if( $role != null && !$role->has_cap('use_event') ) {
            $role->add_cap('use_event');
        }

        // On supprime la variable de notre fonction.
        unset($role);
    }
}
```

Ces permissions vont permettre de sécuriser les actions de l'extension et les menus. WordPress n'affichera que les menus dont l'utilisateur possède la permission. De ce fait, un éditeur ne verra pas le menu Réglages de l'extension.

Initialisation de l'extension

Pour lancer les différentes fonctionnalités de l'extension, il faut initialiser les différents filtres et actions. Pour cela, vous allez regrouper ces appels dans une seule et unique fonction `event_init()` que vous placerez dans le fichier `simple-events.php`.

C'est la seule fonction que contiendra ce fichier et elle sera placée à la fin du fichier.

```
// Fonction d'initialisation du plugin
add_action('plugins_loaded', 'event_init');
function event_init() {
    //...
}
```

Vous ajouterez du code à l'intérieur de la fonction au fur et à mesure du développement. Notez que la fonction sera lancée lors de l'action `plugins_loaded`, pas avant !

Accès en base de données

Pour les différentes fonctionnalités, vous devez créer plusieurs fonctions effectuant des manipulations sur la base de données.

Vous aurez besoin d'une fonction pour :

- récupérer un événement ;
- récupérer plusieurs événements ;
- ajouter un événement ;
- mettre à jour un événement ;
- effacer un événement ;
- récupérer plusieurs événements appartenant à un même tag.

Ainsi, vous allez créer une fonction pour chaque manipulation. Toutes ces fonctions sont contenues dans le fichier `event.db.php`.

Récupérer un événement

Cette fonction permet de récupérer les données d'un événement depuis son ID. Comme vous récupérez toutes les données d'un élément, vous utiliserez la méthode `get_row()` de la classe WPDB :

```
function get_event( $event_id = 0 ) {
    global $wpdb, $table_event;

    // On sécurise
    $event_id = (int) $event_id;

    return $wpdb->get_row(" SELECT * FROM {$table_event} WHERE id
        = '{$event_id}' ");
}
```

Cette fonction peut prendre comme paramètre l'ID de l'événement ; cela veut dire qu'on attend un nombre entier. Dans un tel cas, un moyen très simple de sécuriser la requête SQL est de "caster" (convertir explicitement) la variable au type `integer`. Si la variable n'est pas un entier mais une chaîne de caractères, la variable `$event_id`, une fois castée, sera égale à 0.

Récupérer plusieurs événements

Cette fonction permet de récupérer une liste d'événements. Ils sont automatiquement ordonnés au sens inverse de la date, autrement dit le plus récent en premier, et la fonction prend comme seul paramètre le nombre de résultats souhaité. Par défaut, vous pourrez récupérer jusqu'à vingt événements :

```
function get_events( $limit = 20 ) {
    global $wpdb, $table_event;

    // On sécurise
    $limit = (int) $limit;

    return $wpdb->get_results( "SELECT * FROM {$table_event} ORDER BY
    date_event DESC LIMIT {$limit} " );
}
```

Ajouter un événement

Cette fonction permet d'ajouter un événement dans la base de données. Elle prend trois paramètres : le nom, la description et la date de l'événement. L'ID n'est pas précisé, car ce dernier sera automatiquement ajouté lors de l'insertion dans la base de données. D'ailleurs, si l'insertion réussit, la fonction retournera l'ID de l'événement ajouté.

Pour cela, vous utilisez la variable de classe de `$wpdb->insert_id`. Si l'insertion ne fonctionne pas pour une raison ou pour une autre, la fonction retournera `false`. Notez l'utilisation de la méthode `insert()` qui permet l'insertion en base de données sans saisir la requête SQL :

```
function add_event( $name = '', $description = '', $date_event = '' ) {
    global $wpdb, $table_event;

    $wpdb->insert( $table_event, array('name' => $name, 'description'
    => $description, 'date_event' => $date_event ) );
    return $wpdb->insert_id;
}
```

Mettre à jour un événement

Cette fonction permet de mettre à jour un événement. Pour cela, vous passez en paramètre l'ID de l'événement à modifier, le nouveau nom, la nouvelle description et la nouvelle date.

Ici vous utilisez la méthode `update()` de la classe WPDB, ce qui permet de mettre à jour une ligne de la base de données sans écrire une ligne de SQL :

```
function update_event( $event_id, $name = '', $description = '', $date_event = '' ) {
    global $wpdb, $table_event;

    // On sécurise
    $event_id = (int) $event_id;

    $state = $wpdb->update( $table_event, array('name' => $name, 'description'
    => $description, 'date_event' => $date_event ), array('id' => $event_id ) );
    return $state;
}
```

Effacer un événement

Cette fonction permet de supprimer un événement depuis son ID. Elle retournera le nombre de lignes affectées ; si la valeur retournée vaut 1, alors l'événement a été effacé, si c'est 0, cela veut dire qu'aucun événement n'a été supprimé :

```
function delete_event( $event_id = 0 ) {
    global $wpdb, $table_event;

    // On sécurise
    $event_id = (int) $event_id;

    $state = $wpdb->query(" DELETE FROM {$table_event} WHERE id =
'{$event_id}'");
    return $state;
}
```

Récupérer plusieurs événements appartenant à un même tag

Cette fonction sera détaillée ultérieurement, car elle fait appel à des variables que nous n'avons pas encore vues ni expliquées.

Partie – Administration

Menu de WordPress (jusqu'à la version 2.6)

Pour les besoins de l'extension, nous devons ajouter un menu dans l'onglet Écrire, un autre dans l'onglet Gérer et un dernier dans l'onglet Réglages.

Pour ajouter des menus dans WordPress, vous allez lier une fonction à l'action admin_menu, et cette fonction insérera les menus (voir Figure 11.02).

Figure 11.02

Les menus de l'extension
Simple Events.



Donc, dans un premier temps, vous devez ajouter une action dans la fonction d'initialisation event_init(), mais qui sera lancée uniquement dans l'interface d'administration :

```
// Initialisation Admin
if ( is_admin() ) {
    // Ajout du menu utilisateur
    add_action('admin_menu', 'event_add_menu');
}
```

Et la fonction event_add_menu(), que vous exécuterez lors de l'affichage de l'interface d'administration, ajoute les trois menus :

```
// Menu administration
function event_add_menu() {
    // Menu réglages - Admin
    add_options_page( 'Événements', 'Événements', 'admin_event',
'events_options', 'event_page_options');
```

```
// Menu gérer
add_management_page( 'Gérer les événements', 'Gérer les événements',
    'use_event', 'events', 'event_page' );

// Menu écrire
add_submenu_page( 'post-new.php', 'Ajouter un événement', 'Ajouter un
    événement', 'use_event', 'event_form', 'event_page_form' );
}
```

Cette fonction est placée dans le fichier `event.admin.php`.

Les deux premières fonctions appelées nécessitent la permission `use_event` que les rôles administrateur et éditeur possèdent, tandis que la dernière fait appel à la permission `admin_event` que seul le rôle administrateur possède.

Afficher des notifications dans WordPress

Lorsque vous effectuez une action dans WordPress, ajout/suppression d'un article, enregistrement des options, etc., vous pouvez toujours visionner une notification de l'action réalisée (voir Figure 11.03), qui se présente sous la forme d'un texte sur fond jaune avec une bordure.

Figure 11.03

Notification dans l'interface d'administration WordPress.



Par exemple, lorsque vous effacez un article dans WordPress, vous obtenez la notification suivante : "Article effacé avec succès."

```
function display_wordpress_alert( $texte, $status = 'updated' ) {
    if ( $texte ) {
        ?>
        <div id="message" class="<?php echo ($status != '') ? $status : 'updated';
        ?> fade">
            <p><strong><?php echo $texte; ?></strong></p>
        </div>
        <?php
    }
}
```

La fonction prend deux paramètres : le message de la notification et le statut de la notification. Par défaut, c'est le statut `updated` de mise à jour ; il est généralement utilisé pour

l'ajout et l'édition dans WordPress. Vous pouvez également passer le statut `error` qui met une alerte sur fond rouge, lors d'une erreur par exemple.

Écrire – Formulaire d'ajout et d'édition

Dans cette page, l'extension doit pouvoir afficher un formulaire d'ajout, et ce formulaire doit également servir à l'édition des événements (voir Figure 11.04).

Cela signifie que nous allons passer des données GET en paramètre de cette page, et si nous trouvons le paramètre d'édition d'un événement, nous allons récupérer les données de l'événement et préremplir le formulaire. Selon que nous serons dans un ajout ou une édition, nous utiliserons la fonction de base de données correspondante.

Figure 11.04

Rendu du formulaire d'ajout et d'édition.

La fonction utilisée pour la page est `event_page_form()` ; elle est également contenue dans le fichier `event.admin.php`.

Cette fonction peut être découpée en trois parties :

- le traitement PHP d'ajout et d'édition lors de l'envoi du formulaire ;
- l'affichage des données dans le formulaire, s'il y a édition ;
- la structure HTML.

La première partie (lisez les commentaires dans le code) est la suivante :

```
function event_page_form() {
    // On regarde s'il y a un ID d'événement dans l'URL, si oui c'est une édition,
    // sinon un ajout.
    $edit = ( isset($_GET['event_id']) && ( (int) $_GET['event_id'] != 0 ) ) ?
        true : false;

    // On teste les données du formulaire, et on vérifie que le champ "Titre"
    // est bien complété
    if ( ( isset($_POST['save_event']) && !empty($_POST['name']) ) ) {
```

```

// Sécurité, on teste les permissions.
if ( current_user_can('use_event') && wp_verify_nonce($_POST['event_nonce'],
'add_or_update') ) {

    // On récupère et on nettoie les données POST
    $name = stripslashes($_POST['name']);
    $description = stripslashes($_POST['description']);
    $date_event = stripslashes($_POST['date_event']);

    if ( $edit == true ) { // on édite ?
        $event_id = (int) $_GET['event_id'];
        update_event( $event_id, $name, $description, $date_event );
        display_wordpress_alert('Événement mis à jour !');
    } else { // On ajoute alors...
        $event_id = (int) add_event( $name, $description, $date_event );
        display_wordpress_alert('Événement ajouté !');
    }
    if ( $event_id ) {
        // On récupère les tags.
        $tags = explode( ' ', $_POST['tags'] );
        // On nettoie les tags.
        $clean_tags = array();
        foreach ( (array) $tags as $tag ) {
            $clean_tags[] = trim($tag);
        }
        // On ajoute les tags à l'événement
        global $event_base_tags;
        wp_set_object_terms( $event_id, $clean_tags, $event_base_tags,
false );
    }
}
}
}

```

La deuxième partie – la récupération des données dans le cadre d’une édition – se présente ainsi :

```

if ( $edit == true ) { // Si on édite un événement
    // On récupère l'ID de l'événement
    $event_id = (int) $_GET['event_id'];
    // On récupère les données de l'événement
    $event = get_event( $event_id );

    if ( $event ) { // Si l'événement existe
        $name = stripslashes($event->name); // Le nom
        $description = stripslashes($event->description); // La description
        $date_event = stripslashes($event->date_event); // La date de
l'événement
        global $event_base_tags;
        $tags_array = wp_get_object_terms( $event_id, $event_base_tags,
array('fields' => 'names') );
        $tags = implode( ' ', $tags_array ); // La liste de tags séparés par
une virgule
    }
} else { // Si c'est un ajout, alors les variables sont nulles.
    $name = $description = $date_event = $tags = '';
}

```



```
    }
    ?>
```

Nous reviendrons en détail sur la récupération et l'enregistrement des tags dans la section consacrée à la taxinomie.

La troisième partie correspond au code HTML :

```
<div class="wrap">
    <h2>Ajout (ou édition) d'un événement</h2>

    <form id="addlink" method="post">
        <div id="poststuff">
            <div class="submitbox" id="submitlink">
                <div id="previewview"></div>
                <div class="inside">
                    <p>
                        <label for="date_event">Date</label>
                        <br />
                        <input type="text" size="10" name="date_event" id="date_event"
                            value="<?php echo $date_event; ?>" />
                        <br />
                        <small>Format : YYYY-MM-DD</small>
                    </p>
                </div>

                <p class="submit">
                    <input type="hidden" name="event_nonce" value="<?php echo
                        wp_create_nonce('add_or_update'); ?>" />
                    <input type="submit" class="button button-highlighted"
                        name="save_event" value="Enregistrer" tabindex="4" />
                </p>
            </div>

            <div id="post-body">
                <div id="namediv" class="stuffbox">
                    <h3><label for="name">Nom</label></h3>
                    <div class="inside">
                        <input type="text" name="name" size="30" tabindex="1"
                            value="<?php echo$name; ?>" id="name" /><br />
                        Exemple : Festival de musique
                    </div>
                </div>

                <div id="descriptiondiv" class="stuffbox">
                    <h3><label for="description">Description</label></h3>
                    <div class="inside">
                        <textarea name="description" id="description" style="width:99%;
                            height:100px;"><?php echo $description; ?></textarea><br />
                        Cette description sera affichée sur la page détaillée de
                        l'événement.
                    </div>
                </div>
            </div>
        </div>
    </form>
</div>
```

```

<div id="tagsdiv" class="stuffbox">
  <h3><label for="tags">Tags</label></h3>
  <div class="inside">
    <input type="text" name="tags" size="30" tabindex="1"
      value="<?php echo$tags; ?>" id="tags" /><br />
    Exemple : Musique, Cinéma, Livre
  </div>
</div>
</div>
</div>
</form>
</div>
<?php
}

```

Pour créer le code HTML et faire en sorte qu'il s'intègre bien à l'interface d'administration, vous devez vous baser sur le code HTML des pages existantes de WordPress. Pour la page du formulaire, vous reprendrez la structure HTML de la page du formulaire d'ajout de liens, que vous adapterez à la situation.

Gérer – Liste des événements

La page de gestion des événements se présente sous la même forme que celle des articles : vous retrouvez un tableau avec les événements et vous avez la possibilité d'éditer et supprimer les événements (voir Figure 11.05).

Figure 11.05

Rendu de la page de gestion des événements.

ID	Nom	Tags	Description	Date événement	Actions
1	Evenement 1		Ceci est un super événement ! Et oui !	2009-12-09 00:00:00	Editer Effacer
2	Evene			0000-00-00 00:00:00	Editer Effacer

Vous pouvez distinguer deux parties dans cette fonction : une première permettant la suppression des événements, et le listing des événements à proprement parler.

```

function event_page() {
    // On regarde s'il y a le paramètre GET d'effacement dans l'URL.
    if ( isset($_GET['event_id']) && $_GET['action'] == 'delete' ) {

        // On vérifie les permissions
        if ( current_user_can('use_event') && wp_verify_nonce($_GET['_wpnonce'],
            'delete_event') ) {

            delete_event( $_GET['event_id'] ); // On efface l'événement.
            display_wordpress_alert( 'Événement supprimé.' );
        }
    }
}

```

Notez que nous contrôlons manuellement la présence du nonce avant d'effectuer la suppression de l'événement.

Cette extension ne gère pas la pagination des événements. Sachez toutefois qu'il est possible de faire de la pagination facilement avec la fonction `paginate_links()` :

```
// On récupère la totalité des événements, on aurait pu faire une pagination.
$sevents = get_events( 999999 );
?>
<div class="wrap">
    <h2>Gérer une liste d'événements</h2>
    <table class="widefat">
        <thead>
            <tr>
                <th scope="col">ID</th>
                <th scope="col">Nom</th>
                <th scope="col">Tags</th>
                <th scope="col">Description</th>
                <th scope="col">Date événement</th>
                <th scope="col" colspan="2">Actions</th>
            </tr>
        </thead>
        <tbody>
            <?php
                // On parcourt le tableau d'événements
                foreach ( (array) $sevents as $event ) :
                    // Pour chaque événement, on récupère les tags associés.
                    $tags_array = wp_get_object_terms( $event->id, 'event_tag',
                        array('fields' => 'names') );
                    $tags = implode( ' ', $tags_array );
                    <tr id="post-<?php echo $event->id; ?>" class="alternate"
                        valign="top">
                        <td><?php echo $event->id; ?></td>
                        <td><?php echo stripslashes($event->name); ?></td>
                        <td><?php echo stripslashes($tags); ?></td>
                        <td><?php echo stripslashes($event->description); ?></td>
                        <td><?php echo $event->date_event; ?></td>
                        <td><a href="<?php echo get_option('siteurl'); ?>/wp-admin/
                            post-new.php?page=event_form&event_id=<?php echo $event-
                                >id; ?>">Editer</a></td>
                        <td><a href="<?php echo wp_nonce_url( get_option('siteurl')
                            . '/wp-admin/edit.php?page=events&action=delete&
                                event_id='.$event->id, 'delete_event'); ?>">Effacer</a></td>
                    </tr>
                <?php
                    endforeach;
                ?>
            </tbody>
        </table>
    </div>
<?php
}
```

Ici encore, le code HTML se base sur celui de WordPress.

Réglages – Options des événements

Dans cette extension, la page des options se veut assez sommaire (voir Figure 11.06). En effet, le seul paramètre qu'il est possible de modifier est l'activation ou non du widget. Les réglages seront stockés sur la table des options de WordPress ; pour cela vous utiliserez l'API des options.

Figure 11.06

Rendu de la page de réglages de l'extension.



Là aussi le code se répartit en deux : une partie traitement PHP, dans un premier temps, avec le test des données POST et la mise à jour de l'option si nécessaire, et une partie récupération de l'option et affichage du code HTML dans un second temps :

```
function event_page_options() {
    // Les données viennent du formulaire ?
    if ( isset($_POST['update_event_options']) ) {
        // On vérifie la permission ?
        if ( current_user_can('admin_event') && wp_verify_nonce($_POST
        ['event_nonce'], 'update_options') ) {
            // On teste si la case est cochée
            if ( isset($_POST['event_widgets']) ) {
                $event_widgets = 1; // Si oui, la valeur est 1
                display_wordpress_alert( 'Widget activé ' );
            } else {
                $event_widgets = 0; // Sinon, la valeur est 0
                display_wordpress_alert( 'Widget désactivé ' );
            }
            // On essaie de mettre à jour l'option, si ça ne fonctionne pas,
            c'est qu'elle n'existe pas. Donc on l'ajoute !
            if ( !update_option( 'event_widgets', $event_widgets ) ) {
                add_option( 'event_widgets', $event_widgets );
            }
        }
    }
    // On récupère la valeur actuelle de l'option
    $event_widgets = (int) get_option( 'event_widgets' );
    ?>
    <div class="wrap">
        <h2>Options des événements</h2>

        <form method="post">
            <table class="form-table">
                <tr valign="top">
                    <th scope="row">Widgets</th>
                    <td>
                        <input type="checkbox" id="event_widgets" name="event_widgets"
                        value="1" <?php echo ( $event_widgets == 1 ) ? 'checked'

```

```

        ="checked" : '' ; ?> />
        <label for="event_widgets">Activer le widget "Événement à
        venir</label>
    </td>
</tr>
</table>
<p class="submit">
    <input type="hidden" name="event_nonce" value="<?php echo
    wp_create_nonce('update_options'); ?>" />
    <input type="submit" name="update_event_options" value="Mettre à
    jour les options" />
</p>
</form>
</div>
<?php
}

```

Partie – Taxinomie et réécriture d'URL

Enregistrement de la taxinomie

Une des fonctionnalités de l'extension est la possibilité de classer les événements dans des tags. Ici le but n'est pas d'utiliser les tags des articles avec les événements mais de créer une nouvelle taxinomie, les tags d'événements.

Les articles possèdent la taxinomie `post_tag`, et les événements auront leur propre taxonomie `event_tag`.

L'identifiant de la taxinomie sera également utilisé lors de la construction des URL, pour afficher les pages de tags d'événements, et ces pages permettront d'afficher la liste des événements tagués avec.

Pour simplifier la modification de l'identifiant de la taxinomie, vous stockerez sa valeur dans une variable globale déclarée au tout début de l'extension, juste après la construction du nom de la table de la base de données des événements.

Donc, dans le fichier `simple-events.php`, vous ajouterez la variable globale :

```
$event_base_tags = 'event_tag';
```

Une fois cette variable définie, il sera utile de déclarer la nouvelle taxinomie. Cette déclaration ne doit pas être effectuée avant l'action `init`. Vous ajouterez donc une action lors de cet événement dans la fonction `event_init()` :

```
// Taxinomie
add_action( 'init', 'event_register_taxonomy');
```

La fonction appelée par l'action sera `event_register_taxonomy()` ; elle sera placée dans le fichier `event.taxonomy.php` :

```
function event_register_taxonomy() {
    global $event_base_tags;
```

```
// Enregistre la taxinomie
register_taxonomy( $event_base_tags, 'event', array('hierarchical' =>
false, 'update_count_callback' => '_update_event_count', 'rewrite' =>
false, 'query_var' => false ));
}
```

Lors de la déclaration de la taxinomie, vous désactiverez le rewriting automatique, ainsi que l'ajout du mot-clef dans la classe WP_Query.

Fonction de callback de la taxinomie

Au moment de l'enregistrement de la taxinomie, nous avons passé en trois paramètres un tableau PHP contenant différentes informations, comme le nom de la fonction de callback. Cette fonction sera appelée implicitement par WordPress pour maintenir à jour le compteur de chaque tag d'événement.

Le code suivant sera placé dans le fichier event.taxonomy.php :

```
function _update_event_count( $tags ) {
    global $wpdb, $table_event;
    foreach ( (array) $tags as $tag ) {
        $count = $wpdb->get_var( $wpdb->prepare( "SELECT COUNT(*) FROM
$wpdb->term_relationships, $table_event WHERE $table_event.id =
$wpdb->term_relationships.object_id AND term_taxonomy_id = %d", $tag ) );
        $wpdb->update( $wpdb->term_taxonomy, compact( 'count' ),
            array( 'term_taxonomy_id' => $tag ) );
    }
    return true;
}
```

La fonction prend comme paramètre le tableau de tags modifiés. Pour chaque tag, la fonction va récupérer le nombre d'événements classés, et avec ce nombre la table contenant le compteur du tag sera mise à jour. La requête SQL pour récupérer le nombre d'événements est un peu complexe, elle effectue une jointure entre deux tables SQL.

Enregistrement et récupération des tags

Vous avez vu dans la page du formulaire d'événement deux fonctions liées à l'enregistrement et à la récupération des tags.

```
wp_set_object_terms( $event_id, $clean_tags, $event_base_tags, false );
```

Cette première fonction permet de lier des tags à un événement. Pour cela, il est nécessaire de préciser la taxinomie, ce que vous faites avec la variable globale \$event_base_tags.

Il en est de même avec la fonction :

```
wp_get_object_terms( $event_id, $event_base_tags, array('fields' => 'names') );
```

Notez que le dernier paramètre permet de spécifier les champs que vous souhaitez récupérer ; dans le cadre d'une édition, les noms de tags suffisent largement.

Récupération d'une liste d'événements depuis un tag donné

Il était difficilement concevable de présenter la fonction `get_events_by_tag()` sans parler auparavant de la création de la taxinomie dans cette extension.

Pour récupérer une liste d'événements depuis un tag donné, vous allez utiliser plusieurs fonctions de l'API de taxinomie de WordPress.

Le code est commenté pour expliquer le déroulement du processus :

```
function get_events_by_tag( $tag_slug = '' ) {
    global $event_base_tags, $wpdb, $table_event;
    // On récupère le tag depuis l'identifiant
    $tag = get_term_by('slug', $tag_slug, $event_base_tags);
    // On récupère les ID des événements depuis l'ID du tag
    $events_id = get_objects_in_term( $tag->term_id, $event_base_tags );
    // On vérifie qu'il y a des événements.
    if ( empty($events_id) ) {
        return false;
    }
    // On récupère les événements dont l'ID est présent dans le tableau récupéré
    // ci-dessus
    return $wpdb->get_results( "SELECT * FROM {$table_event} WHERE id IN
    (".implode(',',$events_id).") ORDER BY date_event DESC" );
}
```

Ajout des mots-clefs dans *WP_Query*

Nous avons vu avec les tags qu'il était nécessaire d'avoir un premier mot-clef pour la construction des URL et l'identifiant de la taxinomie. Nous aurons besoin d'un deuxième mot-clef pour la construction des URL de la vue détaillée des événements.

Vous ajouterez donc dans un premier temps une nouvelle variable globale pour le mot-clef de la vue détaillée :

```
$event_base = 'event'; // Mot-clef utilisé dans l'URL
```

Placez-la juste après la déclaration du nom de table dans le fichier `simple-events.php`.

Ensuite, vous insérerez les deux mots-clefs dans la classe `WP_Query` de WordPress ; cela permettra à WordPress de récupérer les données de ces mots-clefs.

Pour cela, vous placerez un filtre sur le hook `query_vars` :

```
add_filter('query_vars', 'event_add_query_var');
```

La fonction `event_add_query_var()` va recevoir en paramètre le tableau de mots-clefs de WordPress, et vous allez y ajouter les deux nouveaux mots-clefs :

```
function event_add_query_var( $wp_query_var ) {
    global $event_base, $event_base_tags;
    // Ajout du mot-clef événement
```

```

$wp_query_var[] = $event_base;
// Ajout du mot-clef tags des événements
$wp_query_var[] = $event_base_tags;
return ($wp_query_var);
}

```

Cette fonction sera elle aussi placée dans le fichier `event.taxonomy.php`.

Redirection sur le template

Maintenant que vous avez ajouté les deux mots-clefs dans WordPress, vous allez créer une fonction permettant de scanner le traitement `WP_Query`. Et dans le cas où l'un des deux mots-clefs possède une valeur, cela signifie que vous êtes soit dans un tag d'événement, soit dans un événement.

Si c'est le cas, vous chargerez une fonction lors du hook `template_redirect` pour rediriger sur le bon template !

Vous allez commencer par mettre en place une fonction qui scanne la requête *via* le hook `parse_query`. Cette ligne de code est à placer dans la fonction `event_init()` comme tous les ajouts d'actions :

```
add_action('parse_query', 'event_parse_query');
```

La fonction appelée `event_parse_query()` scanne les deux mots-clefs et les stocke éventuellement dans des variables globales accessibles depuis toute l'extension :

```

function event_parse_query() {
    global $event_base, $event_base_tags, $current_event, $current_tags_event;

    // On vérifie la présence d'un événement
    $current_event = stripslashes(get_query_var($event_base));
    if ( get_magic_quotes_gpc() ) {
        $current_event = stripslashes($current_event);
    }

    // On vérifie la présence d'un tag d'événement
    $current_tags_event = stripslashes(get_query_var($event_base_tags));
    if ( get_magic_quotes_gpc() ) {
        $current_tags_event = stripslashes($current_tags_event);
    }

    // Si l'un des 2 est complété, alors on ajoute la fonction de redirection
    et on retire les flags de WordPress.
    if ( !empty($current_event) || !empty($current_tags_event) ) {
        // Remove all WP flags
        global $wp_query;
        $wp_query->init_query_flags();

        // Redirect to specific template
        add_action('template_redirect', 'event_template_redirect');
    }
}

```


Vous réinitialisez les flags de WordPress pour éviter l'exécution d'une requête SQL sur les articles, et ainsi améliorer les performances. Vous ajoutez dans la fonction une action sur le hook `template_redirect` ; la fonction liée permettra de charger le bon template de votre thème :

```
function event_template_redirect() {
    global $current_event, $current_tags_event;

    if ( !empty($current_event) || !empty($current_tags_event) ) {
        $template = '';
        // File
        if ( !empty($current_event) ) {
            $tpl_file = 'event-detail.php';
        } else {
            $tpl_file = 'event-tags.php';
        }
        if ( is_file(TEMPLATEPATH . '/' . $tpl_file) ) {
            $template = TEMPLATEPATH . '/' . $tpl_file;
        } else {
            wp_die('The template file <code>'.$tpl_file.'</code> is required for this plugin');
        }
        if ($template) {
            load_template($template);
            exit();
        }
    }
    return false;
}
```

La fonction vérifie dans un premier temps que vous disposez bel et bien d'un ID d'événement ou d'un tag ; selon le cas, vous définissez le template à lancer. Vous vérifiez que le fichier existe bien dans le thème actif ; si c'est le cas, vous le chargez, sinon vous affichez un message d'erreur informant que le template est requis dans le thème.

Rewriting – Initialisation des URL

Pour simplifier la construction des URL dans l'extension et permettre la gestion des différents formats d'adresses de WordPress, les débuts des adresses de tags et d'événements seront automatiquement initialisés. Il suffira ensuite d'ajouter le slug ou l'ID à la fin de l'adresse pour qu'elle soit fonctionnelle.

Vous allez ajouter une action lors du hook `init` de WordPress. Le code doit être placé dans le fichier `simple-events.php`, plus précisément dans la fonction `event_init()` :

```
// Rewriting
add_action('init', 'event_init_rewrite');
```

La fonction `event_init_rewrite()` va générer les deux débuts d'adresses et les stocker dans des variables globales. Pour savoir dans quel cas vous vous situez, vous testez si les permaliens sont actifs ou non.

La fonction suivante sera placée dans le fichier `event.rewrite.php` :

```
function event_init_rewrite() {
    global $wp_rewrite, $event_base_url, $event_base, $event_base_tags,
        $event_tags_base_url, $event_home;

    // La base de l'adresse
    $event_home = get_settings('home') . '/';

    // On teste si les permaliens sont actifs
    if (isset($wp_rewrite) && $wp_rewrite->using_permalinks()) {
        // Si oui, on construit la forme optimisée
        // http://exemple.fr/event/4
        $is_rewriteon = true; // using rewrite rules
        $base_url .= ( substr($wp_rewrite->front, 0, 1) == '/' ) ? substr
            ($wp_rewrite->front, 1, strlen($wp_rewrite->front)) : $wp_rewrite->front;
        $base_url .= $wp_rewrite->root; // set to "index.php/" if using that style

        $event_base_url = $base_url . $event_base . '/';
        $event_tags_base_url = $base_url . $event_base_tags . '/';
    } else {
        // Sinon on construit la méthode classique
        // http://exemple.fr/?event=4
        $event_base_url .= '?' . $event_base . '=';
        $event_tags_base_url = '?' . $event_base_tags . '=';
    }
    // On stocke les 2 adresses dans des variables globales
    $event_base_url = $event_home . $event_base_url;
    $event_tags_base_url = $event_home . $event_tags_base_url;

    // Si les permaliens sont activés, on ajoute les règles de réécriture.
    if ($is_rewriteon === true) {
        add_filter('search_rewrite_rules', 'event_create_rewrite_rules');

        // flush rules if requested
        $wp_rewrite->flush_rules();
    }
}
```

Grâce à cette fonction, si les permaliens sont désactivés, vous obtenez les bases d'adresses suivantes :

```
http://exemple.fr/?event=
http://exemple.fr/?event_tag=
```

S'ils sont actifs, vous avez :

```
http://exemple.fr/event/
http://exemple.fr/event_tag/
```

Pour générer les adresses des événements et des tags d'événements, il suffira d'ajouter l'ID ou le slug à la suite de ces bases !

Rewriting – Génération des règles dans WordPress

Dans la fonction précédente, si les permaliens sont actifs, vous ajoutez un filtre au hook `search_rewrite_rules` ; la fonction associée permettra de générer les règles de réécriture.

```
function event_create_rewrite_rules( $rewrite ) {
    global $wp_rewrite, $event_base, $event_base_tags;

    // Rewriting des événements
    $wp_rewrite->add_rewrite_tag('%' . $event_base . '%', '(.)', $event_base . '=');
    $event_rewrite = $wp_rewrite->generate_rewrite_rules($wp_rewrite->root .
        $event_base . "/%$event_base%/");

    // Rewriting des tags d'événement
    $wp_rewrite->add_rewrite_tag('%' . $event_base_tags . '%', '(.)',
        $event_base_tags . '=');
    $tags_event_rewrite = $wp_rewrite->generate_rewrite_rules($wp_rewrite->root .
        $event_base_tags . "/%$event_base_tags%/");

    // On retourne l'addition des différents tableaux
    return ( $rewrite + $event_rewrite + $tags_event_rewrite );
}
```

Cette fonction prend comme paramètre le tableau des règles de rewriting ; son objectif est d'ajouter les règles des événements et des tags d'événements.

Pour y parvenir, il faut dans un premier temps ajouter le mot-clef de l'URL aux règles de réécriture et ensuite générer les règles *via* la méthode `generate_rewrite_rules()`.

Ce traitement doit être effectué deux fois, et lorsque les deux règles supplémentaires sont générées, il faut additionner les différents tableaux et les retourner, vu que vous travaillez sur un filtre.

Partie – Widget

Lancement du widget

Par défaut, le fichier PHP du widget `event.widget.php` n'est pas chargé par l'extension.

Les fonctions propres aux widgets doivent être chargées lors de l'événement `plugins_loaded`. Vous chargerez le fichier PHP depuis la fonction `event_init()` uniquement si l'option de l'extension est activée.

Ajoutez ceci dans la fonction `event_init()` :

```
// Initialisation widget
$event_widgets = (int) get_option( 'event_widgets' );
if ( $event_widgets === 1 ) {
    @include( dirname(__FILE__) . '/inc/event.widget.php' );
}
```

Création du widget

Un widget dans WordPress se décompose en deux parties : une partie cliente qui affiche le widget sur le thème et une partie admin qui gère les options et l’affichage dans l’interface d’administration. Le code des widgets est exclusivement situé dans le fichier `event.widget.php` (voir Figure 11.07).

Figure 11.07

Rendu du widget côté thème.



La première chose à faire, c’est de vérifier la présence des fonctions spécifiques aux widgets :

```
// On vérifie que l'installation de WordPress supporte les widgets
if ( !function_exists('wp_register_sidebar_widget') ||
!function_exists('wp_register_widget_control') ) {
    return false;
}
```

Si les fonctions n’existent pas, vous ne lancez pas la suite du code.

Vous trouvez ainsi le code du widget côté client ; celui-ci permet de lister les X derniers événements, la quantité étant paramétrable dans les options du widget, tout comme le titre :

```
function widget_event($args) {
    extract($args);
    // On récupère les options du widget
    $options = get_option('widget_event');

    // Si les valeurs sont vides, on met une valeur par défaut
    $title = empty($options['title']) ? 'Événement à venir' : $options['title'];
    $quantity = empty($options['quantity']) ? 20 : $options['quantity'];

    // On imprime les variables nécessaires au bon fonctionnement des widgets
    echo $before_widget;
    echo $before_title . $title . $after_title;

    // On récupère les événements
    $events = get_events( $quantity );
    if ( $events ) { // S'il y a des événements
        global $event_base_url; // Base des URL des événements
        echo '<ul>';
        // On parcourt la liste des événements et on affiche le lien
        foreach( $events as $event ) {
            echo '<li><a href="'. $event_base_url . $event->id . ">". $event->name . '</a></li>';
        }
        echo '</ul>';
    }
}
```

```
    } else { // Sinon
        echo 'Aucun événement à venir' ;
    }
    echo $after_widget;
}
```

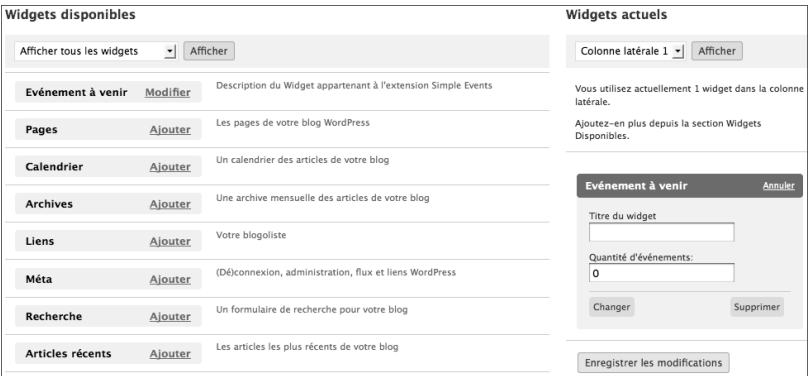
Vous retrouvez ensuite la fonction spécifique à l'administration du widget :

```
function widget_event_control() {
    // On récupère les options du widget
    $options = get_option('widget_event');
    // Ceci gère la mise à jour des widgets
    if ( $_POST['event-submit'] ) {
        $options['title'] = strip_tags(stripslashes($_POST['event-title']));
        $options['quantity'] = (int) $_POST['event-quantity'];
        update_option('widget_event', $options);
    }
    // Formatage des options. Le titre ne peut pas contenir de HTML.
    $title = htmlspecialchars($options['title'], ENT_QUOTES);
    $quantity = (int) $options['quantity'];
    ?>
    <div>
        <p><label for="event-title">
            Titre du widget
            <input type="text" id="event-title" name="event-title"
            value="<?php echo $title; ?>" />
        </label></p>
        <p><label for="event-quantity">
            Quantité d'événements:
            <input type="text" id="event-quantity" name="event-quantity"
            value="<?php echo $quantity; ?>" />
        </label></p>
        <input type="hidden" name="event-submit" id="event-submit" value="1" />
    </div>
    <?php
}
```

Cette fonction permet d'afficher un formulaire contenant les réglages du widget (voir Figure 11.08). Les réglages sont enregistrés dans une option de WordPress.

Figure 11.08

Rendu du widget côté administration.



Vous devez ensuite enregistrer les deux fonctions de widgets de WordPress :

```
// On enregistre le widget pour la partie visiteur
wp_register_sidebar_widget('events', 'Événement à venir', 'widget_event',
array('description' => 'Description du Widget appartenant à l\'extension Simple
Events' ));

// Et la partie administration...
wp_register_widget_control('events', 'Événement à venir', 'widget_event_control' );
```

Ces fonctions sont détaillées au Chapitre 10 consacré aux API de WordPress.

Partie – Filtre du contenu des articles

Il existe deux méthodes pour filtrer le contenu des articles : une méthode évoluée qui fonctionne *via* les shortcodes, mais également une ancienne méthode qui consiste à filtrer manuellement le contenu des articles pour y remplacer le marqueur de son choix.

Marqueur personnalisé

Ce premier marqueur permet d'insérer la liste complète des événements que contient l'extension. Le marqueur à inclure dans le contenu des articles sera `###events###` :

```
function content_list_events( $content = '' ) {
    // On teste la présence du marqueur dans l'article
    if ( strpos($content, '###events###') !== false ) { // S'il existe
        // On récupère l'ensemble des événements
        $events = get_events( 99999 );
        if ( $events ) { // S'il y a des événements
            global $event_base_url;
            $output = '<ul>';
            // On fait une boucle sur les événements pour afficher une liste HTML
            non ordonnée
            foreach( (array) $events as $event ) {
                $output .= '<li><a href="'. $event_base_url.$event->id.' ">'. $event-
                    >name.'</a></li>';
            }
            $output .= '</ul>';
        } else {
            // S'il n'y a aucun événement, on le note.
            $output = 'Aucun événement à venir';
        }
        // On remplace le marqueur par le contenu de la variable $output.
        $content = str_replace( '###events###', $output, $content);
    }
    return $content; // On renvoie le contenu
}
```

Il faut ensuite ajouter cette fonction au filtre `the_content`. Vous insérerez donc le code suivant dans la fonction `event_init()` :

```
// Ajouter un marqueur pour lister les événements
add_filter('the_content', 'content_list_events');
```

Shortcode

La nouvelle API des shortcodes ajoutée dans WordPress 2.5 permet l'insertion facile de balise du type BBcode. Ici, nous souhaitons créer de manière très simple des liens vers les événements avec une mise en page personnalisée.

L'objectif est de remplacer le BBcode du type `[event id='123']` par un lien vers l'événement. Pour cela, vous allez initialiser le shortcode dans la fonction `event_init()` avec le code suivant :

```
// Short code
add_shortcode('event', 'shortcode_event');
```

Cette fonction appellera la fonction `shortcode_event()` chaque fois qu'elle trouvera un BBcode event :

```
function shortcode_event( $atts ) {
    // Récupération des valeurs des attributs
    extract(shortcode_atts(array(
        'id' => ''
    ), $atts));

    // Vérification de l'id
    $id = intval($id);

    // Si l'ID est valide
    if ( $id != 0 ) {
        global $event_base_url;
        // On récupère l'événement
        $event = get_event( $id );
        if ( $event ) {
            // On retourne le lien vers l'événement.
            return '<blockquote><p><a href="'. $event_base_url. $event->id. '">'
                . $event->name. '</a> ('.mysql2date( 'j F Y', $event->date_event).')
                </p></blockquote>';
        }
    }
    return '';
}
```

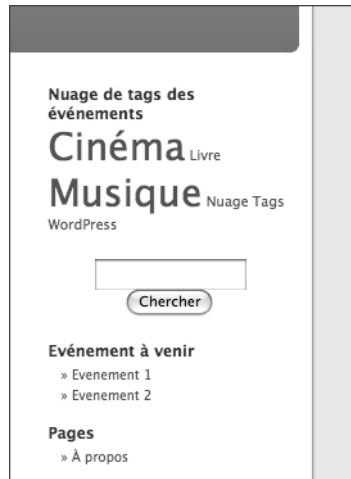
Grâce au shortcode, il n'est plus important de réfléchir à la méthode de remplacement : vous vous concentrez uniquement sur le contenu que vous allez afficher à la place.

Partie – Nuage de tags

Pour créer le nuage de tags des événements, nous n'allons pas réinventer la roue. Pour gagner du temps, tout en proposant un nuage de tags avec différents paramètres, nous allons dupliquer la fonction de nuage de tags de WordPress (voir Figure 11.09).

Figure 11.09

Rendu des nuages de tags.



Plus précisément, vous allez copier et renommer respectivement les deux fonctions `wp_tag_cloud()` et `wp_generate_tag_cloud()` en `event_tag_cloud()` et `event_generate_tag_cloud()` dans le fichier `event.template.php`.

Par rapport à ces fonctions, vous allez modifier trois choses.

D'abord vous allez changer la source des tags ; ici il n'est pas question de récupérer les tags des articles, mais ceux des événements. Pour cela, vous remplacerez dans la fonction `event_tag_cloud()` :

```
$tags = get_tags( array_merge($args, array('orderby' => 'count', 'order'
=> 'DESC')) ); // Always query top tags
```

par

```
global $event_base_tags;
$tags = get_terms( $event_base_tags, array_merge($args, array('orderby'
=> 'count', 'order' => 'DESC')) ); // Always query top tags
```

Vous remplacerez aussi l'appel de la fonction `wp_generate_tag_cloud()` par `event_generate_tag_cloud()`, soit :

```
$return = wp_generate_tag_cloud( $tags, $args ); // Here's where those top tags
get sorted according to $args
```

par

```
$return = event_generate_tag_cloud( $tags, $args ); // Here's where those top tags
get sorted according to $args
```

Enfin, dans la fonction `event_generate_tag_cloud()`, vous modifierez la méthode pour créer les liens. Au lieu d'employer la fonction `get_tag_link()` que propose WordPress pour les tags des articles, vous allez utiliser le slug du tag et la base de l'adresse *via* une variable globale.

Vous remplacerez donc :

```
$tag_links[$tag->name] = get_tag_link( $tag->term_id );
```

par

```
global $event_tags_base_url;
$tag_links[$tag->name] = $event_tags_base_url . $tag->slug;
```

Grâce à cette méthodologie, vous avez reproduit le fonctionnement du nuage de tags de WordPress en quelques clics pour les tags d'événements. Ces fonctions étant relativement complexes d'un point de vue algorithmique, il n'y a pas un grand intérêt à expliquer leur fonctionnement ici.

Au final, vous obtenez le code suivant, à placer dans le fichier event.template.php :

```
function event_tag_cloud( $args = '' ) {
    $defaults = array(
        'smallest' => 8, 'largest' => 22, 'unit' => 'pt', 'number' => 45,
        'format' => 'flat', 'orderby' => 'name', 'order' => 'ASC',
        'exclude' => '', 'include' => ''
    );
    $args = wp_parse_args( $args, $defaults );

    global $event_base_tags;
    $tags = get_terms( $event_base_tags, array_merge($args, array('orderby'
    => 'count', 'order' => 'DESC')) ); // Always query top tags

    if ( empty($tags) )
        return;

    $return = event_generate_tag_cloud( $tags, $args ); // Here's where those
    top tags get sorted according to $args

    if ( is_wp_error( $return ) )
        return false;

    $return = apply_filters( 'event_tag_cloud', $return, $args );

    if ( 'array' == $args['format'] )
        return $return;

    echo $return;
}

// $tags = prefetched tag array ( get_tags() )
// $args['format'] = 'flat' => whitespace separated, 'list' => UL, 'array'
=> array()
// $args['orderby'] = 'name', 'count'
function event_generate_tag_cloud( $tags, $args = '' ) {
    global $wp_rewrite;
    $defaults = array(
        'smallest' => 8, 'largest' => 22, 'unit' => 'pt', 'number' => 45,
        'format' => 'flat', 'orderby' => 'name', 'order' => 'ASC'
```

```

);
$args = wp_parse_args( $args, $defaults );
extract($args);

if ( !$tags )
    return;
$counts = $tag_links = array();

global $event_tags_base_url;

foreach ( (array) $tags as $tag ) {
    $counts[$tag->name] = $tag->count;
    $tag_links[$tag->name] = $event_tags_base_url . $tag->slug;
    if ( is_wp_error( $tag_links[$tag->name] ) )
        return $tag_links[$tag->name];
    $tag_ids[$tag->name] = $tag->term_id;
}

$min_count = min($counts);
$spread = max($counts) - $min_count;
if ( $spread <= 0 )
    $spread = 1;
$font_spread = $largest - $smallest;
if ( $font_spread <= 0 )
    $font_spread = 1;
$font_step = $font_spread / $spread;

// SQL cannot save you; this is a second (potentially different) sort on a
subset of data.
if ( 'name' == $orderby )
    uksort($counts, 'strnatcasecmp');
else
    asort($counts);

if ( 'DESC' == $order )
    $counts = array_reverse( $counts, true );
elseif ( 'RAND' == $order ) {
    $keys = array_rand( $counts, count($counts) );
    foreach ( $keys as $key )
        $temp[$key] = $counts[$key];
    $counts = $temp;
    unset($temp);
}

$a = array();

$rel = ( is_object($wp_rewrite) && $wp_rewrite->using_permalinks() ) ?
    ' rel="tag" ' : '';

foreach ( $counts as $tag => $count ) {
    $tag_id = $tag_ids[$tag];
    $tag_link = clean_url($tag_links[$tag]);
    $a[] = "<a href='$tag_link' class='tag-link-$tag_id' title='" .
        attribute_escape( sprintf( __gettext( '%d topic', '%d topics', $count ),
        $count ) ) . "'$rel style='font-size: " .
        ( $smallest + ( ( $count - $min_count ) * $font_step ) )

```

```

        . "$unit; '$tag'>$tag</a>";
    }

    switch ( $format ) :
    case 'array' :
        $return =& $a;
        break;
    case 'list' :
        $return = "<ul class='event-tag-cloud'>\n\t<li>";
        $return .= join("</li>\n\t<li>", $a);
        $return .= "</li>\n</ul>\n";
        break;
    default :
        $return = join("\n", $a);
        break;
    endswitch;

    return apply_filters( 'event_generate_tag_cloud', $return, $tags, $args );
}

```

Partie – Internationalisation

Une extension qui n'est pas internationalisée ne peut pas avoir de succès ; l'internationalisation est indispensable de nos jours. Deux méthodes sont disponibles pour charger la traduction de votre extension. Ces méthodes sont à placer dans la fonction `event_init()` de votre extension.

La première méthode est complètement générique, bien qu'un peu complexe :

```

// Localization
$locale = get_locale(); // fr_FR
if ( !empty( $locale ) ) {
    $mofile = dirname(__FILE__) . '/languages/event-' . $locale . '.mo';
    // event-fr_FR.mo
    load_textdomain('event', $mofile);
}

```

Ici, vous récupérez vous-même le code de la langue utilisée et vous chargez le fichier MO en précisant son chemin absolu. L'intérêt de cette méthode, c'est une parfaite compatibilité quels que soient l'emplacement de l'extension et la version de WordPress employée.

L'autre méthode, plus simple mais moins souple et implémentée plus récemment dans WordPress, consiste à utiliser la fonction `load_plugin_textdomain()`.

Vous obtenez alors :

```
load_plugin_textdomain('event', false, 'simple-events/languages');
```

Partie – Templates des thèmes

L'extension contient deux templates dans le dossier theme. Ces templates sont des exemples à utiliser par les webmasters lors de l'installation de l'extension sur leur blog. Ils doivent de leur côté adapter le code HTML pour correspondre à leur thème.

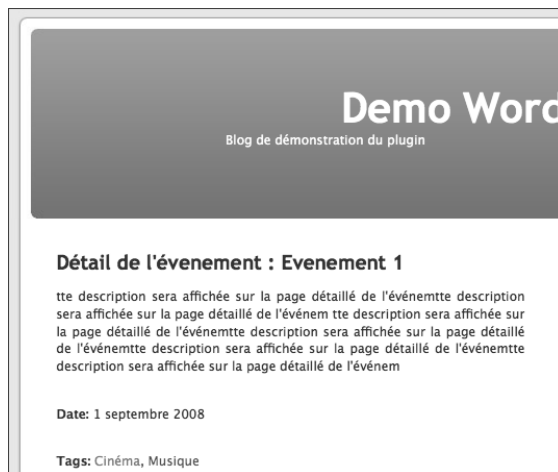
Ici le code HTML des templates est celui du thème par défaut de WordPress.

Template des événements

Ce template est le fichier PHP event-detail.php. Il permet d'afficher le détail de chaque événement (voir Figure 11.10).

Figure 11.10

Page de détails des événements depuis le thème utilisateur.



Le template peut être décomposé en deux parties. Une première partie qui effectue des traitements PHP permettant de récupérer les données de l'événement et de les formater :

```
<?php
global $current_event, $event_tags_base_url;
// On vérifie que le paramètre est un nombre
$current_event = intval($current_event);
// Nous récupérons les données de l'événement.
$event = get_event( $current_event );
if ( $event ) :
    // On récupère et formate les valeurs de l'événement
    $name = $event->name;
    $description = $event->description;
    $date_event = mysql2date( 'j F Y ', $event->date_event );
    $tags_array = wp_get_object_terms( $event->id, 'event_tag' );
    $tag_urls = array();
    foreach ( $tags_array as $tag ) {
        $tag_urls[] = '<a href="' . $event_tags_base_url . $tag->slug .
            '">'. $tag->name. '</a>';
    }
    $tags = implode( ' ', $tag_urls );
?>
```

Et une seconde partie composée du code HTML :

```
<h2>Détail de l'événement : <?php echo $name; ?></h2>
<div class="entry">
  <p><?php echo $description; ?></p>
  <p><br />
  <strong>Date:</strong> <?php echo $date_event; ?>
</p>
  <p><br />
  <strong>Tags:</strong> <?php echo $tags; ?>
</p>
</div>
```

Notez que le reste du code HTML et les autres fonctions PHP sont propres au thème. Pour plus de détails, consultez la partie de ce livre consacrée au développement de thème.

Template des tags d'événements

Ce template est le fichier PHP event-tags.php ; il permet d'afficher la liste des événements pour un tag donné (voir Figure 11.11).

Figure 11.11

Page de tags des événements listant les événements.



```
<?php
global $current_tags_event, $event_base_url;
$events = get_events_by_tag( $current_tags_event );
?>
<h2 class="pagetitle">Événements tagués avec : <?php echo $current_tags_event; ?>
</h2>
<?php
if ( $events ) :
  foreach( $events as $event ) :
    ?>
    <div class="post">
      <h2><a href="<?php echo $event_base_url.$event->id; ?>"><?php echo
        $event->name; ?></a></h2>
```

```
        <p><strong>Date:</strong> <?php echo mysql2date( 'j F Y ', $event-  
            >date_event ); ?></p>  
    </div>  
    <?php  
        endforeach;  
    else:  
    ?>  
    <p>Aucun événement pour ce tag.</p>  
    <?php  
        endif;  
    ?>
```

Le traitement est relativement basique : vous récupérez dans un premier temps la liste des événements depuis le slug du tag. Vous effectuez ensuite une boucle sur les événements en affichant le titre, le lien et la date.

Conclusion

Débuter le développement d'extensions sous WordPress est quelque chose de relativement accessible. Ici il n'est pas question de développement orienté objet, juste de fonctions PHP couplées à un mécanisme d'actions et de filtres.

Bien entendu, il est nécessaire de bien connaître le PHP pour parvenir à vos fins... Il n'empêche que, avec la base d'extensions existantes, vous pouvez facilement, même en tant que développeur occasionnel, modifier une extension pour vos besoins. Et c'est au fur et à mesure que vous modifierez des extensions et que vous vous familiariserez avec WordPress que vous serez capable de vous lancer dans le développement d'une extension de A à Z !