



DEVELOPPER UN PLUGIN POUR GLPI 0.84

TABLE DES MATIERES

Introduction	2
1. L'architecture d'un plugin	3
1.1 Les fichiers nécessaires	3
1.2 Le dossier ajax.....	3
1.3 Le dossier front.....	4
1.4 Le dossier inc	4
1.5 Les fichiers setup.php et hook.php.....	4
1.6 Le fichier config.php	5
2. La programmation	6
2.1 Le fichier setup.php.....	6
2.2 Le fichier hook.php	10
2.3 Le fichier config.php	11
2.4 Le fichier about.php	12
2.5 Le fichier example.php.....	13
Conclusion.....	13

INTRODUCTION

GLPI (gestionnaire libre de parc informatique) est une application web permettant la gestion de parc informatique et de gestion des services d'assistance distribués sous licence GPL. Le projet totalement communautaire a été lancé en 2003 par l'association Indepnet.

GLPI permet l'ajout de plugins, c'est-à-dire de fonctionnalités supplémentaires qui ne sont pas gérées nativement par GLPI. Le développement d'un plugin doit suivre des règles strictes pour garantir son bon fonctionnement. Ainsi, nous allons voir comment programmer pas-à-pas un plugin pour la version 0.84 de GLPI.

Pour suivre ce tutoriel, il est fortement conseillé d'avoir des notions (au moins basiques) de la programmation orientée objet et du PHP. Il est aussi conseillé d'encoder les fichiers php au format UTF-8.

Tous les fichiers développés tout au long de ce tutoriel sont disponibles à la fin.

1. L'ARCHITECTURE D'UN PLUGIN

Un plugin GLPI est composé d'un ensemble de fichiers regroupés dans un dossier. Le nom de ce dossier ne doit contenir que des caractères alphanumériques minuscules, sans espaces. Dans ce tutoriel, nous appellerons notre plugin *example*. Nous allons donc créer un dossier comme ci-dessous.



Une fois le plugin terminé, il suffira de déposer ce dossier dans le répertoire *plugins* de GLPI (celui-ci se trouve en général dans : `/var/www/glpi/plugins/`)

1.1 LES FICHIERS NECESSAIRES

Un plugin GLPI est composé au minimum de plusieurs fichiers php. Des fichiers css, javascript ou encore des scripts visual basics peuvent être utilisés pour ajouter encore plus de fonctionnalités. Nous allons cependant nous focaliser sur le développement d'un plugin minimal, qui pourra servir de base pour vos plugins.

Dans le dossier de notre plugin (le dossier *example* créé précédemment), nous devons trouver au minimum l'architecture suivante :



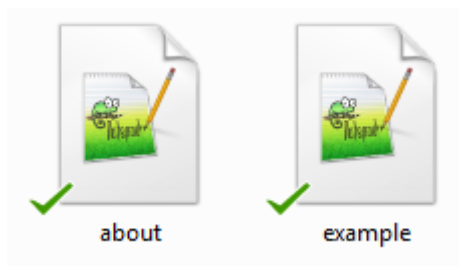
1.2 LE DOSSIER AJAX

Le dossier ajax se trouve à la racine de notre plugin. Il contient tout ce qui est en rapport avec l'Ajax. L'Ajax est un langage qui combine JavaScript, CSS, XML, DOM et le XMLHttpRequest afin d'améliorer la maniabilité et le confort d'utilisation des

applications web. Le plugin que nous allons développer étant très basic, nous n'aurons pas besoin d'utiliser de l'Ajax. Nous laisserons donc ce dossier vide.

1.3 LE DOSSIER FRONT

Par convention, ce dossier doit comprendre tous les formulaires et les pages à afficher. Pour notre plugin, nous n'allons pas utiliser de formulaires de saisies, mais nous allons créer 2 pages : une page d'accueil et une page d'informations. Dans le dossier front, il nous faut donc créer ces deux fichiers :



1.4 LE DOSSIER INC

Ce dossier contient tous les objets (c'est-à-dire les classes) du plugin. Il y a des conventions à respecter si nous voulons créer une classe. Le nom du fichier déclarant la classe doit être de la forme suivante :

`pluginNom_pluginNom_objet.class.php`

Par exemple pour déclarer une classe Profile pour notre plugin, il faut (dans le dossier inc) créer un fichier ayant pour nom `pluginExampleProfile.class.php`.

Notre plugin sera simple, par conséquent nous n'aurons pas besoin de classes. Le dossier inc sera donc vide.

1.5 LES FICHIERS SETUP.PHP ET HOOK.PHP

Par convention, ces deux fichiers se trouvent à la racine du dossier et sont nécessaires à l'installation ainsi qu'à la désinstallation du plugin. Nous analyserons leur contenu plus tard, mais il est préférable de commencer à les créer dès maintenant.

1.6 LE FICHIER CONFIG.PHP

Contrairement aux deux fichiers précédents, la présence de ce fichier n'est pas du tout obligatoire. Ce fichier permet de configurer le plugin. Notre plugin sera simple et ne nécessitera aucune configuration. Cependant nous verrons quand même comment faire cette page étant donné que la plupart des « vrais » plugins ont besoin d'être configurés.

2. LA PROGRAMMATION

Nous allons maintenant écrire les fonctions des fichiers créés précédemment. Il faut savoir que les noms des fonctions et des procédures que nous allons écrire doivent respecter des règles précises. Le respect de ces conventions permet au logiciel GLPI de faire appel à nos fonctions lorsqu'il y en a besoin.

2.1 LE FICHIER SETUP.PHP

Ce fichier sert à définir les informations générales du plugin avant de l'installer. Nous allons commencer par les fonctions les plus simples.

2.1.1 **plugin_version_example()**

Le prototype de cette fonction est le suivant :

```
function plugin_version_nomPlugin()
```

Il faut remplacer ici *nomPlugin*, par le nom du plugin que vous développez. Dans notre cas, cette fonction va donc s'appeler :

```
function plugin_version_example()
```



Par la suite de ce tutoriel, je me contenterai de donner le prototype général des fonctions. Il faudra donc adapter les noms des fonctions correctement.

Écrivons maintenant cette fonction, comme je l'ai dit précédemment, il n'y a aucune difficulté dans cette fonction. Contentez-vous d'adapter le code suivant :

```
function plugin_version_example()  
{  
    return array(  
        'name'           => 'Mon plugin',  
        'version'        => '0.84+1.0',  
        'author'         => 'Hakan SENER',  
        'license'        => 'GPLv2+',  
        'homepage'       => 'http://www.upmf-grenoble.fr/',  
        'minGlpVersion' => '0.84');  
}
```

L'argument 'minGlpVersion' correspond à la version de GLPI à partir de laquelle le plugin est compatible. Dans ce cas, notre plugin ne pourra pas être installé sur des versions antérieures à 0.84.

Voici le résultat obtenu :

Liste des plugins									
Nom	Version	Licence	Statut	Auteurs	Site Web	conforme CSRF			
FusionInventory	0.84+2.2	AGPLv3+	Activé	David DURIEUX & FusionInventory team		Oui	Désactiver	Désinstaller	
Mon plugin	0.84+1.0	GPLv2+	Non installé	Hakan SENER		Oui	Installer	Désinstaller	
Monitoring	0.84+1.0	AGPLv3+	Activé	David DURIEUX		Oui	Désactiver	Désinstaller	
Services Web	1.4.2	GPLv2+	Activé	Remi Collet, Nelly Mahu-Lasson, Walid Nouh		Oui	Désactiver	Désinstaller	

[Voir le catalogue des plugins](#)

2.1.2 plugin_example_check_prerequisites()

Le prototype de cette fonction est le suivant :

```
function plugin_nomPlugin_check_prerequisites()
```

Cette fonction sert à ajouter d'autre prérequis nécessitant permettant l'installation du plugin. La fonction renvoie « true » si l'installation du plugin est autorisée. Elle peut être utile pour nécessiter par exemple la présence d'un plugin sans quoi notre plugin ne fonctionnerait pas. Dans notre cas, nous allons utiliser cette fonction pour limiter l'installation du plugin sur la version 0.84 de GLPI seulement. Voici le code de cette fonction :

```
function plugin_example_check_prerequisites()
{
    if (version_compare(GLPI_VERSION, '0.84', 'lt') ||
version_compare(GLPI_VERSION, '0.85', 'gt'))
    {
        echo "This plugin requires GLPI 0.84.X";
        return false;
    }
    return true;
}
```

Ainsi, notre plugin ne pourra être installé que sur la version 0.84 de GLPI.

2.1.3 plugin_example_check_config()

Le prototype de cette fonction est le suivant :

```
function plugin_example_check_config($verbose=false)
```


Cette fonction permet de vérifier que la configuration du plugin est correcte. Dans tous les cas, la fonction ressemble à ça :

```
function plugin_example_check_config($verbose=false)
{
    if (true) // Configuration OK
    {
        return true;
    }

    if ($verbose)
    {
        echo 'Installed / not configured';
    }
    return false;
}
```

2.1.4 plugin_init_example()

Le prototype de cette fonction est le suivant :

```
function plugin_init_nomPlugin()
```

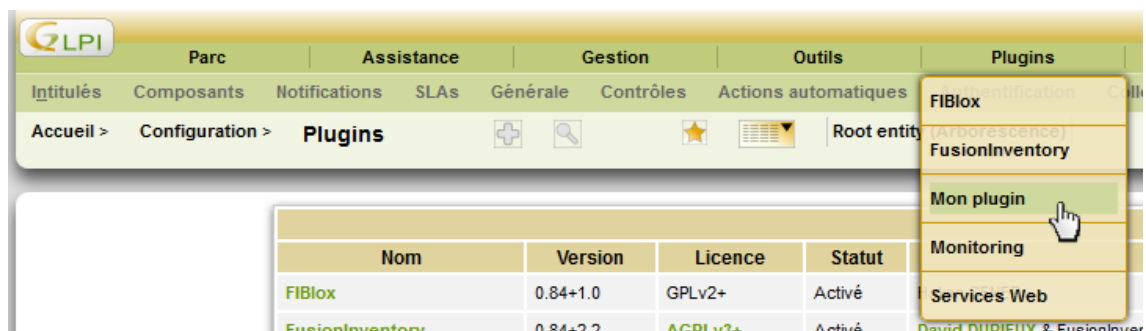
Cette fonction a pour but de mettre en place la structure des pages du plugin. Pour cela on utilise des « hooks ».

Un « hook » ou crochet, permet de s'accrocher à des fonctionnalités déjà existantes d'un système. GLPI met à disposition des hooks pour à peu près tout. Il existe des hooks pour faire un onglet, donner un titre à une page, ajouter du css, ou encore du javascript.

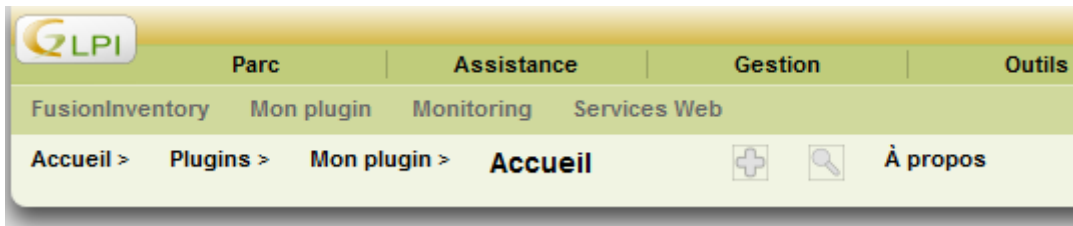
À l'aide de ces hooks, nous allons ajouter une entrée dans le menu plugin de GLPI :

```
$PLUGIN_HOOKS['menu_entry']['example'] = 'front/example.php';
$PLUGIN_HOOKS["helpdesk_menu_entry"]["example"] = true;
```

Voici le résultat obtenu :



Nous allons maintenant définir la structure de nos futures pages. Voici le résultat que nous voulons avoir pour la page d'accueil :



Pour déclarer le titre de notre page (ici « Accueil »), on utilise les hooks suivants :

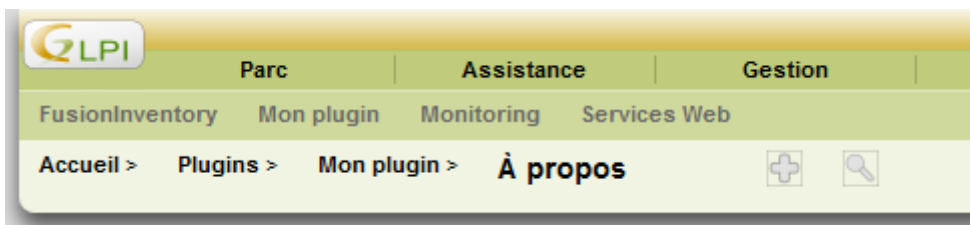
```
$PLUGIN_HOOKS['submenu_entry']['example']['options']['home']['title']
= 'Accueil';
$PLUGIN_HOOKS['submenu_entry']['example']['options']['home']['page']
= '/plugins/example/front/example.php';
```

Notez qu'ici on précise l'entrée 'home'. Nous aurions très bien pu choisir un autre nom pour cette entrée. En effet, nous verrons plus tard qu'il faut rappeler cette entrée sur la page qui reçoit ces hooks. Il suffira alors d'indiquer correctement le nom de l'entrée qu'on a déclaré préalablement (ici 'home').

Enfin, nous allons déclarer le sous-menu « À propos » du plugin. Pour cela rien de plus simple, il suffit de déclarer un nouveau hook comme cela :

```
$PLUGIN_HOOKS['submenu_entry']['example']['options']['home']['links']['
'À propos'] = '/plugins/example/front/about.php';
```

La déclaration des hooks de la page d'accueil est terminée. Nous allons maintenant nous intéresser à la page « À propos ». Voici ce à quoi va ressembler cette page :



Comme vous pouvez le voir, il n'y a aucun sous-menu. Simplement le titre de cette page. Normalement vous devriez être capable de déclarer ces hooks correctement.

```
$PLUGIN_HOOKS['submenu_entry']['example']['options']['about']['title']
= 'À propos';
```

```
$PLUGIN_HOOKS['submenu_entry']['example']['options']['about']['page']  
= '/plugins/example/front/about.php';
```

Il ne nous reste plus que quelques hooks à déclarer :

```
// adresse de la page de configuration  
$PLUGIN_HOOKS['config_page']['example'] = 'config.php';  
// nécessaire pour la sécurité  
$PLUGIN_HOOKS['csrf_compliant']['example'] = true;  
// fonction permettant de vérifier les droits du profile connecté  
$PLUGIN_HOOKS['change_profile']['example'] =  
'plugin_change_profile_example';
```

2.2 LE FICHIER HOOK.PHP

Dans ce fichier on retrouve entre autre le corps des fonctions enregistrées dans les hooks. Nous n'avons enregistrées qu'une fonction : la fonction `plugin_change_profile_example`. Cette fonction a pour but d'indiquer les droits de l'utilisateur connecté à GLPI. Ainsi on pourra réserver certaines parties du plugin aux utilisateurs SuperAdmin. Voici le corps de cette fonction :

```
function plugin_change_profile_example()  
{  
    // For example : same right of computer  
    if (Session::haveRight('computer','w'))  
    {  
        $_SESSION["glpi_plugin_example_profile"] = array('example'  
=> 'w');  
    }  
    else if (Session::haveRight('computer','r'))  
    {  
        $_SESSION["glpi_plugin_example_profile"] = array('example'  
=> 'r');  
    }  
    else  
    {  
        unset($_SESSION["glpi_plugin_example_profile"]);  
    }  
}
```

On retrouve ensuite dans ce fichier deux fonctions très importantes : les fonctions d'installation et de désinstallation du plugin.

C'est dans ces fonctions qu'on crée par exemple, les tables sql que va utiliser le plugin. Dans notre cas, nous n'utiliserons aucune table particulière donc ces fonctions seront vides.

```
// procédure d'installation du plugin
// c'est ici qu'on crée notamment les tables
function plugin_example_install()
{
    return true;
}

// procédure de désinstallation du plugin
// c'est ici qu'on supprime entre autre les tables créées
function plugin_example_uninstall()
{
    return true;
}
```

2.3 LE FICHIER CONFIG.PHP

Ce fichier sert à configurer le plugin. Notre plugin n'aura pas de configuration particulière, mais nous allons limiter l'accès à cette page aux SuperUtilisateur. Pour cela nous allons vérifier si l'utilisateur a le droit d'écriture :

```
// On vérifie qu'on a les droits d'écriture
Session::checkRight("config", "w");
```

Ensuite, il faut inclure l'entête et le pied de page comme dans chaque page à afficher du plugin :

```
Html::header("Mon plugin",$_SERVER['PHP_SELF'], "config", "plugins");
echo 'Page de configuration du plugin';
Html::footer();
```

Notre page de configuration est terminée, voici le code complet :

```
include ('../../inc/includes.php');

// On vérifie qu'on a les droits d'écriture
Session::checkRight("config", "w");

// Pour être disponible même quand le plugin n'est pas activé
Plugin::load('example');

Html::header("Mon plugin",$_SERVER['PHP_SELF'], "config", "plugins");
echo 'Page de configuration du plugin';
Html::footer();
```

2.4 LE FICHIER ABOUT.PHP

Dans chaque formulaire qui sera affiché, il faut systématiquement écrire les trois lignes suivantes :

```
//pour utiliser les fonctionnalités de GLPI
include ("../../inc/includes.php");

//pour utiliser la structure définie dans le fichier setup.php
Html::header('Mon
plugin',$_SERVER['PHP_SELF'], "plugins", "example", "about");

//corps de la page

//fin de la page
Html::footer();
```

On remarque que dans l'appelle de la fonction `Html::header`, on indique les hooks qui seront utilisés pour cette page. Ici on utilise tous les hooks « about » du plugin « example ». Si on change donc la déclaration des hooks dans le fichier `setup.php`, on changera donc la structure de la page qui charge ces hooks.

Pour terminer, il ne nous manque plus qu'à écrire le corps de la page. Pour cela on utilise du php « classique ». Nous allons nous contenter d'afficher simplement un message comme ceci :

```
//corps de la page
echo 'À propos du plugin<br />';
```

Notre page de configuration est maintenant terminée, comme vous pouvez le constater il y a très peu de code au final car on utilise les outils qui nous sont mis à disposition. Voici le code complet :

```
//pour utiliser les fonctionnalités de GLPI
include ("../../inc/includes.php");

//pour utiliser la structure défini dans le fichier setup.php
Html::header('Mon
plugin',$_SERVER['PHP_SELF'], "plugins", "example", "about");

//corps de la page
echo 'À propos du plugin<br />';

//fin de la page
Html::footer();
```

2.5 LE FICHER EXAMPLE.PHP

Cette page sera notre page d'accueil du plugin. Comme nous allons faire une page simple, le code sera très similaire à celui de la page about.php :

```
include ('../../../../../inc/includes.php');

Html::header('Mon plugin',
$_SERVER['PHP_SELF'], "plugins", "example", "home");

echo "page d'accueil du plugin<br />";

Html::footer();
```

Ici on charge les hooks « home » du plugin exemple.

CONCLUSION

Nous avons terminé notre plugin. Même si notre plugin est simple, vous aurez remarqué qu'au final il y a juste quelques règles à assimiler et moins de code que ce qu'on aurait pu imaginer.

Pour aller plus loin, je mets à disposition des liens utiles qui m'ont servis à développer mon propre plugin.

- Forum de GLPI
<http://www.glpi-project.org/forum/viewforum.php?id=8>
- Les différents hooks
https://code.google.com/p/qbplugin-rapidminer/wiki/GLPI_Crear_Pluging
- BDD et SQL
<http://www.glpi-project.org/forum/viewtopic.php?pid=175574#p175574>
- Plugin exemple plus complet
https://forge.indepnet.net/projects/plugins/wiki/Fr_CreatePlugin084
- Fichiers et plugin développés au cours de cette documentation
<https://github.com/senerh/D-veloppeur-un-plugin-GLPI-0.84>