Ecriture d'un Plugin pour GLPI

A partir d'un besoin concret, nous allons explorer la structure des Plugins de GLPI et montrer comment étendre les fonctionnalités de ce logiciel de gestion de parc.

Mots-clés: GLPI, PHP, Plugin

1.Objectifs

1.1.Présentation de GLPI

GLPI signifie *Gestionnaire libre de parc informatique*. Il s'agit d'une application PHP+MySQL libre, développée par une équipe française (c'est assez rare pour être souligné). A travers son interface Web, GLPI permet, en autre, de gérer les ressources informatiques, les licences logicielles, les consommables, les fournisseurs, les réservations de matériel. GLPI assure aussi la gestion du "helpdesk", à travers la création et la gestion de tickets.

GLPI est disponible sur le site **http://glpi-project.org**. Sa version stable actuelle est la v0.71.5, mais la livraison de la version 0.72 semble imminente à l'heure où nous écrivons ces lignes.

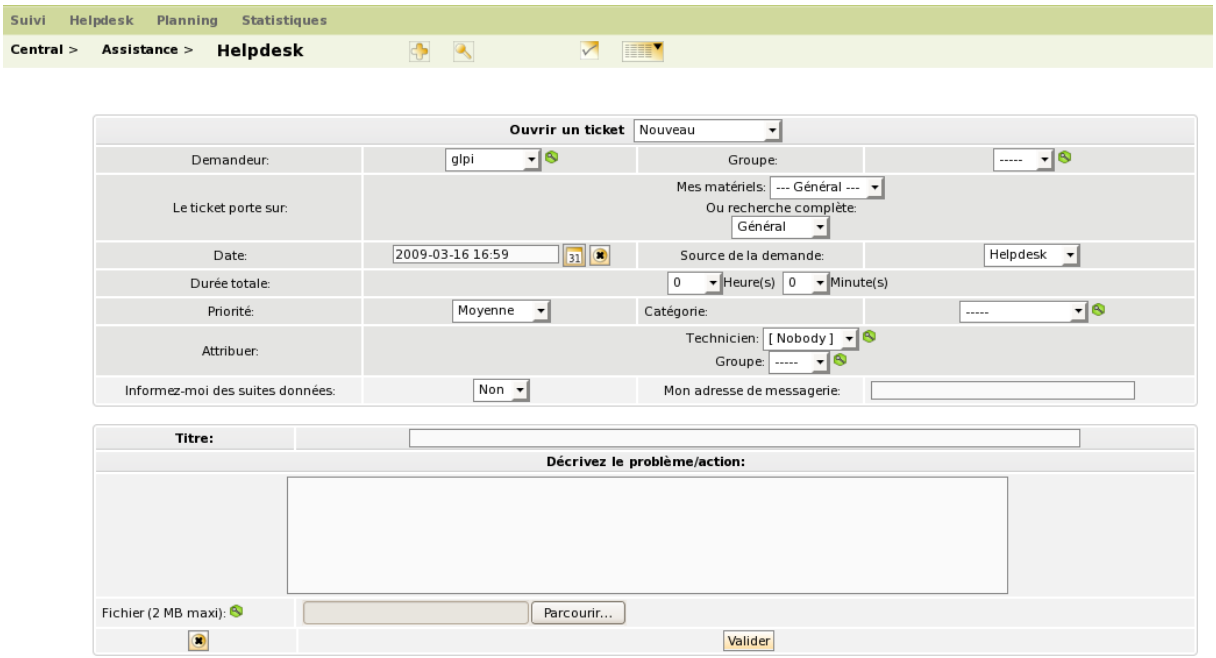
Souvent ce logiciel est couplé à une autre application nommée OCSng-Inventory (site **http://www.ocsinventory-ng.org**) qui est aussi principalement développée par des Français. Cette application a pour vocation d'établir un recensement des matériels et logiciels déployés. Grâce à son agent, déployé sur chaque poste utilisateur, le couple GLPI-OCS permet aussi le déploiement d'applications à distance. Cette association fait de ces outils une solution plus puissante que les alternatives comme Zentrack ou OTRS.

Un article de Linux-Magazine (numéro 91 de Février 2007) montre comment installer ces deux applications et comment intégrer OCS dans GLPI. Dans le présent article, nous nous intéressons uniquement à GLPI et, en particulier, nous allons montrer comment l'enrichir avec des informations personnalisées.

1.2.Nos besoins complémentaires

1.2.1.Ajout de champs "ouverts"

Lorsque les techniciens du "helpdesk" sont sollicités par les utilisateurs, ils créent ce qu'on nomme un "ticket" avec l'interface de GLPI. Les informations qu'ils peuvent ou doivent saisir sont assez complètes (Status, Priorité, Demandeur, etc...), comme le montre l'écran ci-dessous :



(écran 1)

Cependant, les champs proposés ne sont pas toujours suffisants en regard des besoins "métier".

Par exemple, le formulaire ne permet pas de saisir des informations "ouvertes" comme un numéro de série de PDA, ou bien un numéro de ticket "externe" quand le problème est transmis à un fournisseur tiers qui nous retourne son propre numéro de ticket.

On pourrait dérouler ces exemples à l'infini, aussi tentons de les généraliser : notre premier besoin consiste en pouvoir ajouter des champs supplémentaires lors de la saisie d'un ticket.

1.2.2.Séparer la cause du problème de sa solution

Lors de la création du ticket, l'auteur décrit le problème dans l'emplacement prévu à cet effet. Puis, chaque intervenant qui essaye de résoudre le problème ajoute un nouveau "suivi" au ticket. Jusqu'à ce que quelqu'un résolve le problème. Dans ce cas, nous souhaiterions que la solution soit clairement identifiée et puisse être visualisée simplement ultérieurement.

1.2.3.Associer des copies d'écran aux tickets

L'écran de saisie du formulaire permet d'associer des fichiers à un ticket (voir copie d'écran ci-dessus). Cependant, si ces fichiers sont en fait des images (par exemple des copies d'écran contenant les messages d'erreurs fatidiques), on aimerait pouvoir visualiser ces images lors de la consultation du ticket.

2.Solution proposée : écriture d'un plug-in

Nous profitons des besoins décrits précédemment pour vous montrer comment écrire un *Plugin* (une extension) pour GLPI. En effet, GLPI offre la possibilité, par un mécanisme de "hooks" (crochets), d'enrichir ses traitements par du code supplémentaire.

Bien sûr, il y a quelques règles à suivre et quelques limitations. Des informations sur la structure des plugins sont disponibles à l'adresse suivante : **https://dev.indepnet.net/plugins/wiki/CreatePlugin** [GLPI1]. Une étude des sources permet en outre de compléter ces informations.

Notons qu'à l'heure où cet article est rédigé, le document [GLPI1] est incomplet. Nous devrons aussi nous référer au complément [GLPI4] (migration 0.71 vers 0.72). En effet, nous nous intéressons à la version 0.72 de GLPI qui devrait être disponible en version stable très rapidement (si ce n'est pas déjà le cas).

 Note : dans cet article, nous supposons que le code source de GLPI a été installé sous le répertoire **/opt/glpi** et nous nommerons **$ROOT** ce répertoire.

2.1.Les "hooks " de GLPI

Les scripts de GLPI utilisent un certain nombre de variables globales, dont un tableau associatif multi-dimensionnel, nommé **$PLUGIN_HOOKS**. A différentes étapes de la gestion des objets (comme les tickets, les fournisseurs, le matériel...), GLPI invoque la liste de tous les plugins qui ont enregistré une fonction pour cette étape.

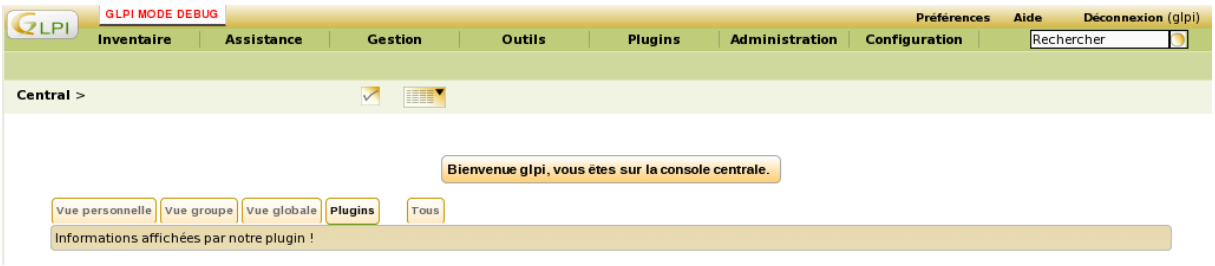
Les "hooks" peuvent être classés en plusieurs catégories.

Le tableau suivant liste les "hooks" qui impactent l'interface Web :

Nom du "hook"	*Type*	*Description*
menu_entry	booléen	affiche une option dans le menu *Central* > *Plugins*
submenu_entry	script	page affichée quand on sélectionne le nom du plugin dans le menu *Central* > *Plugins*
helpdesk_menu_entry	booléen	affiche le nom du plugin dans l'onglet "Plugins" lors de l'affichage du suivi d'un ticket
headings	fonction	retourne les libellés à afficher dans les onglets (voir §4.3.1)
headings_action	fonction	retourne le nom des fonctions à invoquer quand on sélectionne les menus des onglets (voir §4.3.1)
config_page	script	formulaire de configuration du plugin (voir §4.2)
add_javascript	nom de fichier	nom de fichier javascript à inclure dans les pages (le nom est relatif au répertoire du plugin)
add_css	nom de fichier	nom de fichier CSS à inclure dans les pages (le nom est relatif au répertoire du plugin)
central_action	fonction	retourne le code HTML à afficher par la sélection de l'onglet Pugins du "central" (voir écran n°2 ci-dessous)
display_planning	fonction	définit l'affichage dans le planning

planning_populate	fonction	remplit un tableau des objets affichés dans le planning
reports	fonction	donne la possibilité au plugin de compléter les pages de rapports (menu *Central* > *Outils* > *Rapports*)
user_preferences	fonction	retourne le code HTML qui permet à l'utilisateur de modifier ses préférences concernant ce plugin
stats	fonction	donne la possibilité au plugin d'ajouter du HTML dans le tableau de présentation des statistiques (menu *Central* > *Assistance* > *Statistiques*)

L'écran ci-dessous montre l'affichage produit par le hook **central_action** :



(écran 2)

Le tableau suivant liste les "hooks" qui sont invoqués lors de la manipulation des objets internes de GLPI :

Nom du "hook"	*Type*	*Description*
pre_item_add, **item_add**	fonctions	fonctions invoquées avant et après l'ajout d'un nouvel objet
pre_item_delete, **item_delete**	fonctions	fonctions invoquées avant et après la destruction d'un objet (voir §4.4)
pre_item_update, **item_update**	fonctions	fonctions invoquées avant et après la modification d'un objet
pre_item_purge, **item_purge**	fonctions	fonctions invoquées avant et après la purge d'un objet
pre_item_restore, **item_restore**	fonctions	fonctions invoquées avant et après la restauration d'un objet
use_massive_action	booléen	indique si le plugin doit gérer les actions de modification, suppression, purge ou restauration invoquées à partir des formulaires affichant des listes d'objets
item_transfer	fonction	fonctions invoquées lors du transfert d'un item vers un autre contenant

rule_matched	fonction	fonctions appelées lors du traitement de chaque règle

Enfin, le tableau suivant liste les "hooks" qui impactent des services divers :

Nom du "hook"	*Type*	*Description*
cron	fréquence	donne la périodicité de l'appel de la fonction définissant la tâche planifiée du plugin codée dans la fonction dont le nom est **cron_plugin_nom_plugin()**
redirect_page	script	nom du script à invoquer pour les redirections
init_session	fonction	fonction invoquée lors de la création d'une session (quand on s'authentifie dans l'interface Web de GLPI)
change_profile	fonction	fonction invoquée quand l'utilisateur change d'identité dans l'interface
change_entity	fonction	fonction invoquée quand l'utilisateur change d'entité au sens GLPI
restrict_ldap_auth **retrieve_more_data_from_ldap**	fonction	fonctions invoquées lors de la recherche d'un utilisateur dans l'annuaire.

2.2.Autres variables globales accessibles par les plug-ins

Outre la variable globale **$PLUGIN_HOOKS**, GLPI définit un certain nombre d'autres variables globales utilisables par les Plugins. Le tableau ci-dessous en liste les principales :

Nom de la variable	*Description*
$DB	instance de la classe **DB** permettant d'accéder à la base
$CFG_GLPI	paramètres globaux de GLPI modifiables par le menu "Configuration" de l'interface Web
$LANG	tableau des libellés affichables
$NEEDED_ITEMS	tableau des classes d'objets utilisés par un script
GLPI_ROOT	(constante globale) correspondant au répertoire racine de l'interface Web

2.3.Messages et localisation

Pour chaque langue supportée par notre plugin, il doit exister un fichier situé sous le répertoire **$ROOT/plugins/nom_plugin/locales/**. Le nom des fichiers reprend la norme i18n. On trouvera

donc les fichiers **fr_FR.php** pour les libellés en Français, **en_GB.php** pour les libellés en anglais (libellés par défaut), etc...

Ces fichiers contiennent des lignes qui initialisent le tableau **$LANG['nom_plugin']**, par exemple :

```
$LANG['nom_plugin']['mot_clé'][indice] = "libellé";
```

Les valeurs de 'mot_clé' et 'indice' sont choisies arbitrairement par le plugin.

2.4.API de support fournie par GLPI

2.4.1.Classes usuelles - accès aux tables de la base

Parmi les classes définies dans les fichiers **$ROOT/inc/*class.php**, la plus utilisée est probablement **CommonDBTM**, définie dans **$ROOT/inc/commondbtm.class.php**. Cette classe implémente les fonctions de connexion, d'accès et de manipulation de la base.

Pour chaque nouvel objet qu'on soit stocker, il suffira donc de dériver cette classe pour profiter des méthodes **add()**, **update()**, **delete()**, etc...

Notons aussi la classe **glpi_phpmailer** située dans **$ROOT/inc/mailing.class.php** et qui permet d'émettre des mails.

2.4.2.Fonctions globales

Le contenu du répertoire **$ROOT/inc/** montre un certain nombre de fichiers dont le nom est de la forme ***.function.php**. Ces fichiers contiennent des fonctions globales réutilisables dans le code de notre plugin. Par exemple le fichier **display.function.php** contient les fonctions **commonHeader()** et **commonFooter()** qui affichent l'entête et le pied des pages de l'interface Web.

3.Architecture de notre plug-in

Ca y est, nous avons suffisamment présenté la partie théorique, retroussons nos manches et implémentons notre plugin !

Il faut tout d'abord le baptiser. Nous décidons qu'il s'appellera "**vignette**", même s'il affichera plus que les vignettes représentant des copies d'écran...

3.1.Arborescence des fichiers et répertoires

Conformément à la documentation [GLPI1] et [GLPI4], nous devons créer le répertoire **$ROOT/plugins/vignette** sous lequel on créera les sous-répertoires et les fichiers (même vides) suivants :

Répertoire ou fichier	*Description*
./inc/	ensemble de scripts inclus par d'autres scripts du plugin

./front/	scripts invoqués directement par l'interface Web
./pics/	images propres au plugin
./docs/	contient les fichiers **Changelog.txt**, **Lisezmoi.txt** et **Roadmap.txt**
./locales/	contient les fichiers de messages (voir §2.3)
./index.php	soit un fichier vide, soit un fichier qui contient le code HTML affiché si le plugin est listé dans le menu "Plugins" du menu principal
./setup.php	(obligatoire) définit les fonctions obligatoires
./hook.php	définit les hooks du plugin

3.2.Création du squelette de notre plugin

Nous avons toutes les informations pour démarrer :

```
$ cd /opt/glpi/plugins
$ mkdir vignette
```

3.2.1.Stockage en base de données

Comme stipulé dans le document [GLPI1], les informations complémentaires sur les tickets et qui sont gérées par notre plugin, doivent être stockées dans des tables dont le nom est préfixé par **glpi_plugin_vignette_**.

Nous allons donc créer la table **glpi_plugin_vignette_extinfo** (comme "Extended Information") qui contiendra toutes les informations complémentaires associées à un ticket. Pour chaque ticket, nous ajouterons les champs suivants :

Nom du champ	*Description*
vignette_extticket	contient la valeur du ticket externe fourni éventuellement par un fournisseur externe
vignette_solution	contient la solution qui a permis de clore le ticket
id_tracking	identifiant (ID) du ticket auquel sont rattachées ces informations. C'est une référence au champ **ID** de la table **glpi_tracking**
ID	la classe **CommonDBTM** impose que chaque table possède un champ de type auto-incrément qui soit une clé primaire

Comme nous l'avons dit plus haut, les auteurs de GLPI ont eu la bonne idée de proposer une classe de base, nommée **CommonDBTM** qui encapsule les accès au SGBD. La gestion de notre nouvelle table **glpi_plugin_vignette_extinfo** est assurée par la classe **plugin_Vignette** définie dans le fichier **$ROOT/plugins/vignette/inc/plugin_vignette_classes.php** :

```
<?php
if (!defined('GLPI_ROOT')) {
        die("Sorry. You can't access directly to this file");
}

class plugin_Vignette extends commonDBTM
{
        public function plugin_Vignette()
        {
                $this->table = "glpi_plugin_vignette_extinfo";
                $this->type = PLUGIN_VIGNETTE_TYPE;
        }
};
?>
```

Note : ne vous précipitez pas pour créer la table **glpi_plugin_vignette_extinfo** dans la base de GLPI, avec un "CREATE TABLE" que vous feriez avec un client SQL. Nous verrons au §3.3.1, un mécanisme d'installation et désinstallation proposé par GLPI et qui réalise cette création.

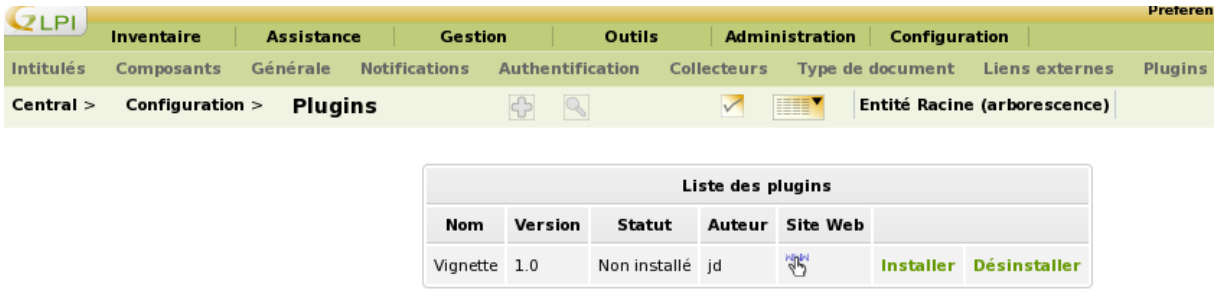
Puis nous créons un fichier **setup.php** qui doit contenir au minimum six fonctions obligatoires.

3.2.2.Fonction plugin_version_vignette()

La première fonction définit simplement le nom et la version du plugin. Le nom de l'auteur et l'adresse de son site sont facultatifs :

```
function plugin_version_vignette()
{
        return array( 'name'    => "Vignette",
                      'minGlpiVersion' => '0.71',
                      'version' => '1.0',
                      'author' => 'jd',
                      'homepage' => 'http://www.maje.biz');
}
```

Nous avons indiqué le numéro de version de GLPI minimum attendu. Le menu *Central* > *Configuration* > *Plugins* affiche la liste des plugins disponible ainsi que les informations retournées par cette fonction :



(écran 3)

3.2.3.Fonction plugin_init_vignette()

C'est la fonction maitresse. Elle est aussi obligatoire. Elle définit la liste des "hooks" auxquels notre plugin souhaite être relié.

```
function plugin_init_vignette()
{
  global $PLUGIN_HOOKS;

  $PLUGIN_HOOKS['config_page']['vignette'] =
                          'front/plugin_vignette_config.php';
  $PLUGIN_HOOKS['helpdesk_menu_entry']['vignette'] = false;
  $PLUGIN_HOOKS['menu_entry']['vignette'] = false;
  $PLUGIN_HOOKS['item_delete']['vignette'] = 'plugin_vignette_item_delete';
  $PLUGIN_HOOKS['headings']['vignette'] = 'plugin_vignette_get_headings';
  $PLUGIN_HOOKS['headings_action']['vignette'] =
                           'plugin_vignette_headings_action';

  registerPluginType('vignette',"PLUGIN_VIGNETTE_TYPE",14588,
                      array ('classname' => "plugin_Vignette",
                             'tablename' => "glpi_plugin_vignette",
                             'typename' => "vignette",
                             'formpage' => NULL,
                             'searchpage' => NULL,
                             'template_ables' => false,
                             'deleted_tables' => false,
                             'recursive_type' => false,
                             'specif_entities_table' => false,
                             'reservation_type' => false));
}
```

L'appel à la fonction **registerPluginType()** permet de définir une nouvelle catégorie d'objets qui sera gérée par le plugin. En effet, nous allons stocker de nouvelles informations dans une nouvelle table de la base (voir §3.3). Cette table sera assimilée à une classe d'objets.

Les paramètres fournis sont :

- le nom du plugin (**vignette**)
- une chaine définissant sont type (**PLUGIN_VIGNETTE_TYPE**)
- la valeur du type (voir [GLPI3] pour le choix des valeurs)
- un tableau associatif contenant :
  - le nom de la table où sont stockées les informations gérées par le plugin (**tablename**)
  - le nom de la classe qui dérive de commonDBTM et gère la table en base (**classname**)
  - le nom du nouveau type des objets (**typename**)
  - des paramètres d'affichage de formulaire ou de gestion des objets

3.2.4.Fonction plugin_vignette_check_prerequisites()

Cette fonction obligatoire, sans paramètre, sera appelée par GLPI afin de déterminer si le plugin est installable. Cette fonction a toute latitude pour vérifier si les prérequis sont satisfaits. Ici, nous vérifions seulement que la version de GLPI est au moins la v0.72, nous pourrions assi vérifier que la version courante de PHP dispose bien des fonctions de traitement d'images dont nous aurons besoin plus loin :

```
function plugin_vignette_check_prerequisites()
{
        if (GLPI_VERSION >= 0.72)
                return true;

        echo "A besoin de la version 0.72";
        return false;
}
```

3.2.5.Fonction plugin_vignette_check_config()

Dans le même ordre d'idée, cette fonction vérifie si la configuration de GLPI permet d'utiliser le plugin. Dans notre exemple, nous ne testons rien de particulier et retournons la valeur "true" :

```
function plugin_vignette_check_config()
{
        return true;
}
```

3.2.6.Fonction plugin_vignette_install()

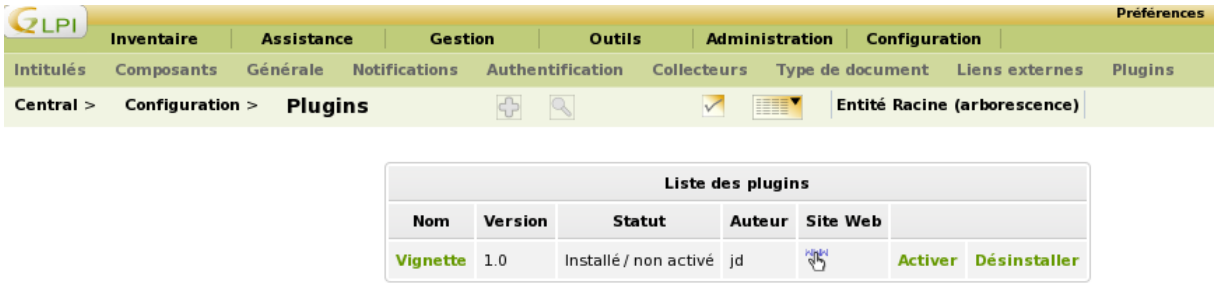
Avant d'exposer le code de cette fonction, nous devons expliquer comment GLPI détermine si un plugin doit être installé ou désinstallé (voir écran précédent dans §3.2.2) : pour afficher la liste des plugins disponibles, GLPI balaye le répertoire **$ROOT/plugins**. Pour chaque fichier **setup.php** trouvé, il invoque la fonction **plugin_*_version()** et regarde dans la table **glpi_plugins** si ce plugin y est déjà inscrit. Dans le cas positif, il y trouve aussi le statut du plugin, dans le cas négatif, il ajoute une nouvelle ligne à la table et donne au plugin le status "Non installé".

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+-------+---------------------+
| ID | directory | name     | version | state | author| homepage            |
+----+-----------+----------+---------+-------+-------+---------------------+
|  5 | vignette  | Vignette | 1.0     |     2 | jd    | http://www.maje.biz |
+----+-----------+----------+---------+-------+-------+---------------------+
```

La fonction d'installation doit créer la nouvelle table qui va nous permettre de stocker nos informations complémentaires :

```
function plugin_vignette_install()
{
        $DB = new DB;

        $query = "CREATE TABLE `glpi_plugin_vignette_extinfo` (
                        `ID` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
                        `id_tracking` int(11) NOT NULL,
                        `vignette_extticket` char(32) NOT NULL default '',
                        `vignette_solution` text NOT NULL default ''
                ) ENGINE=MyISAM DEFAULT CHARSET=latin1;";

        $DB->query($query) or die($DB->error());

        return true;  // ne pas oublier !!!
}
```

Si on choisit l'option "Installer" dans l'écran de gestion des plugins, la fonction précédente est invoquée, puis la liste des plugins est à nouveau réaffichée et indique bien un nouveau statut pour notre plugin :

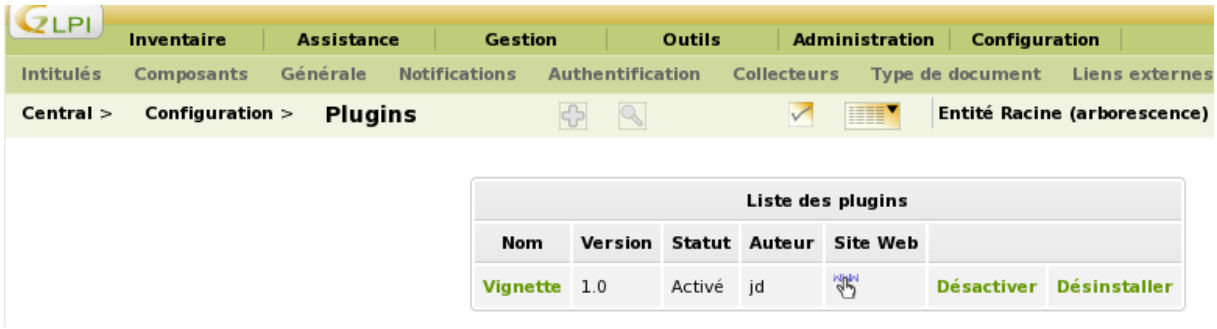


(écran 4)

Le statut a aussi changé dans la base :

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+-------+---------------------+
| ID | directory | name     | version | state | author| homepage            |
+----+-----------+----------+---------+-------+-------+---------------------+
|  5 | vignette  | Vignette | 1.0     |     4 | jd    | http://www.maje.biz |
+----+-----------+----------+---------+-------+-------+---------------------+
```

L'écran 4 précédent montre que le plugin doit encore être "activé". En sélectionnant cette option, GLPI va inqué successivement les fonctions **plugin_vignette_check_prerequisites()** et **plugin_vignette_check_config()** décrites au §3.2.4 et §3.2.5. Si ces deux fonctions retournent la valeur **true**, alors le plugin est activé !



(écran 5)

Effectivement, dans la base, son statut a aussi changé :

```
mysql> select * from glpi_plugins;
+----+-----------+----------+---------+-------+-------+---------------------+
| ID | directory | name     | version | state | author| homepage            |
+----+-----------+----------+---------+-------+-------+---------------------+
|  5 | vignette  | Vignette | 1.0     |     1 | jd    | http://www.maje.biz |
+----+-----------+----------+---------+-------+-------+---------------------+
```

3.2.7.Fonction plugin_vignette_uninstall()

Rien de paticulier : la fonction est appelée si on sélectionne l'option "Désinstaller" dans l'écran de gestion des plugins (écran 5). Cette fonction est la réciproque de la fonction d'installation : elle doit supprimer la table précédemment créée :

```
function plugin_vignette_uninstall()
```

```
{
        $DB = new DB;

        $query = "DROP TABLE `glpi_plugin_vignette_extinfo";
        $DB->query($query) or die($DB->error());

        return true; // ne pas oublier !!!
}
```

4.Affichage des pages spécifiques au plugin

4.1.Modèles des pages affichées par notre plugin

Les scripts qui créent du contenu affichable doivent adopter la structure suivante :

1) définir la constante **GLPI_ROOT** dont la valeur est le chemin racine de GLPI (ici, c'est **/opt/glpi**)
2) si on veut utiliser des objets de GLPI, comme **TRACKING_TYPE**, **COMPUTER_TYPE**, etc..., il faut les nommer dans le tableau **$NEEDED_ITEMS** qui doit être inclus avant l'inclusion du fichier **$ROOT/config/define.php**. Cele permettra un chargement automatique des fichiers où sont déclarées ces classes d'objets
3) même si l'interface Web de GLPI affiche les menus et les options autorisées en fonction des privilèges attachés au profil de l'utilisateur employé pour la connexion, il est recommandé de rajouter des tests vérifiant l'adéquation entre le profil courant et l'action requise. En effet, un utilisateur un peu plus "malin" que les autres pourrait appeler directement un script PHP s'il en connait le nom. Pour cela, on dispose des fonctions **checkRight()** et **haveRight()**. **checkRight()** affiche un message d'erreur si le profil ne dispose pas du privilège, puis termine le script tandis que **haveRight()** retourne uniquement un booléen
4) l'affichage du menu principal de GLPI est réalisé en appelant la fonction **commonHeader()**
5) le bas de page est affiché en appelant la fonction **commonFooter()**
6) on utilise les feuilles de styles situés sous **./css/**

Ce qui donne le squelette suivant :

```
<?php
$NEEDED_ITEMS=array(liste d'objets);
if (!defined('GLPI_ROOT')) {
        define('GLPI_ROOT', '/opt/glpi');
}
include (GLPI_ROOT."/inc/includes.php");

/* vérification éventuelle des permissions: */
//checkRight("config", "w");
...
/* Les paramètres sont : le titre de la page, l'URL,
 * la rubrique d'affichage, le nom de la page */
commonHeader(liste de paramètres);
...
commonFooter();
?>
```

4.2.Page de configuration du plugin

Comme indiqué dans la fonction **plugin_init_vignette()** du §3.2.3, le hook **config_page** permet de donner le nom d'un script qui sera appelé quand on clique sur le nom du plugin (ici : Vignette) dans l'écran n°5. Par convention, lorsqu'une page est construite directement par une action de l'interface Web, son script est placé dans le répertoire **./front** du plugin.

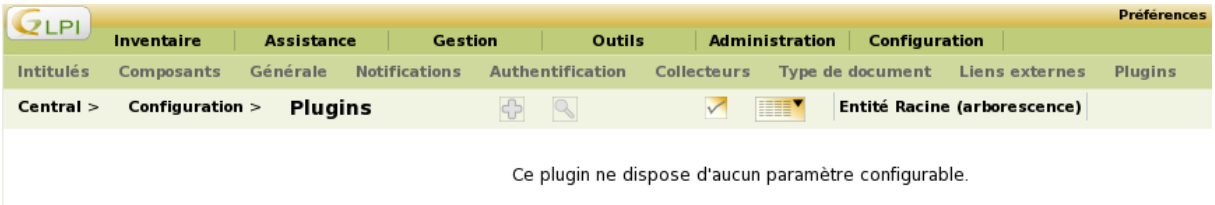
Le script **front/plugin_vignette_config.php** est très simple car notre plugin ne dispose pas de paramètres configurables. Nous aurions aussi pu ne pas définir ce hook dans la fonction **setup.php**...

```
<?php
$NEEDED_ITEMS=array("setup");
if (!defined('GLPI_ROOT')) {
        define('GLPI_ROOT', '/opt/glpi');
}
include (GLPI_ROOT."/inc/includes.php");

checkRight("config", "w");

commonHeader($LANG["common"][12],$_SERVER['PHP_SELF'],"config","plugins");
echo "Ce plugin ne dispose d'aucun paramètre configurable";
commonFooter();
?>
```

Comme indiqué au §4.1, à propos des privilèges et de la fonction **checkRight()**, si le profil de l'utilisateur ne lui permet pas de modifier la configuration de GLPI, le "beau" message suivant est affiché :

(écran 6)

dans le cas contraire, si le profil autorise la configuration de GLPI, la page suivante s'affiche :



(écran 7)

4.3.Affichage des champs personnalisés

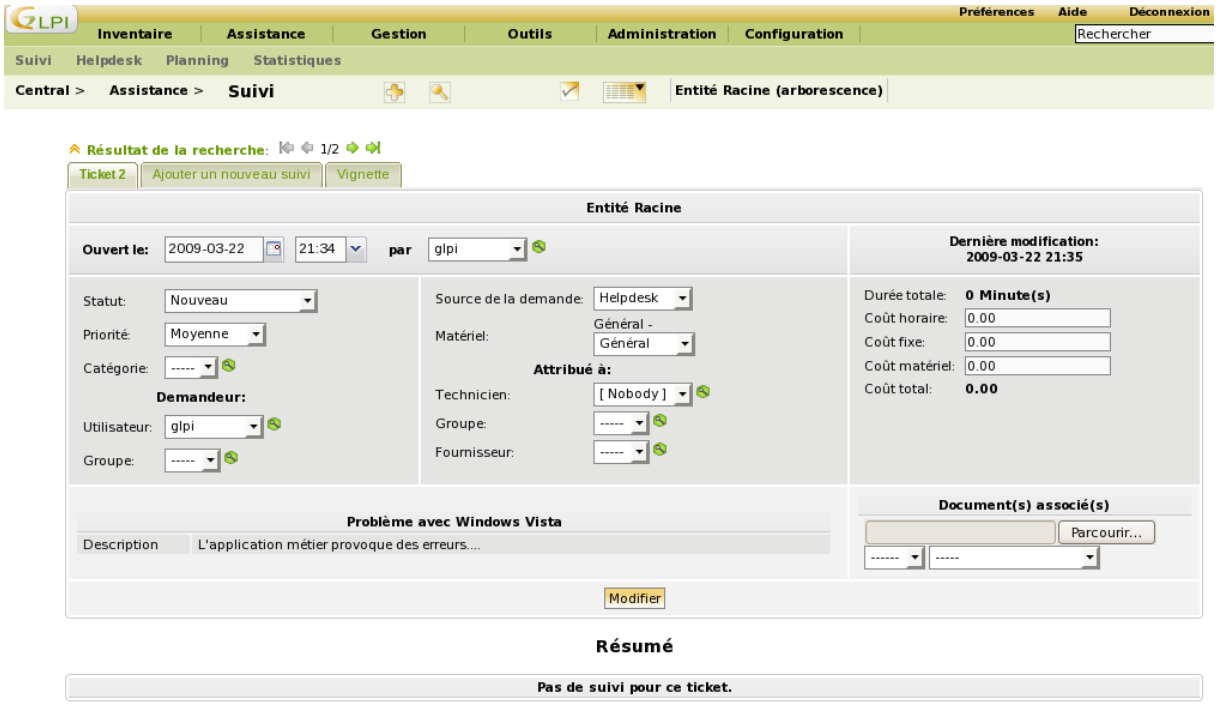
Dans sa version actuelle (0.72), GLPI ne propose pas de hook pour modifier l'écran de saisie d'un ticket. Donc si nous voulons ajouter de nouvelles informations, deux alternatives sont possibles :

– patcher les scripts de GLPI pour afficher ces champs
– gérer ces champs dans un formulaire spécifique associé au plugin

Pour des raisons de modularité et pour simplifier les mises à jour de GLPI, nous allons implémenter la 2ème solution alors que la 1ère est probablement plus ergonomique pour l'utilisateur.

4.3.1.Ajout d'une option dans le menu Plugins du suivi d'un ticket

Pour faire afficher ce nouveau formulaire, nous devons ajouter un nouvel onglet au formulaire de suivi des tickets :



(écran 8)

Ce nouvel onglet "Vignette" est créé par le hook suivant défini dans le fichier **setup.php** (voir §3.2.3) :

```
$PLUGIN_HOOKS['headings']['vignette'] = 'plugin_vignette_get_headings';
```

La fonction **plugin_vignette_get_headings()** doit retourner la liste des libellés pour chaque onglet que l'on souhaite ajouter dans l'écran n°8. Mais nous souhaitons modifier uniquement le formulaire de "Suivi". Pour cela, nous filtrons le "type" de l'objet concerné en ne nous attachant qu'au type "TRACKING_TYPE" :

```
function plugin_vignette_get_headings($type, $withtemplate)
{
      // Les types d'objets sont définis dans config/define.php
      switch ($type) {
      case TRACKING_TYPE:
                // Une seul nouvel onglet...
                return array( 1 => "Vignette" );
                break;
      }
      return false;
```

```
}
```

Puis nous devons donner l'action à effectuer quand l'option "Vignette" est sélectionnée. Toujours dans le fichier **setup.php**, nous trouvons la ligne suivante :

```
$PLUGIN_HOOKS['headings_action']['vignette'] =
                                    'plugin_vignette_headings_action';
```

La fonction **plugin_vignette_headings_action()** ressemble à la fonction précédente, mais, au lieu de retourner les valeurs des libellés, elle retourne le nom des fonctions à appeler quand le choix du menu est sélectionné :

```
function plugin_vignette_headings_action($type)
{
        switch ($type) {
        case TRACKING_TYPE:
                return array(1 => 'plugin_headings_vignette');
                break;
        }
        return false;
}
```

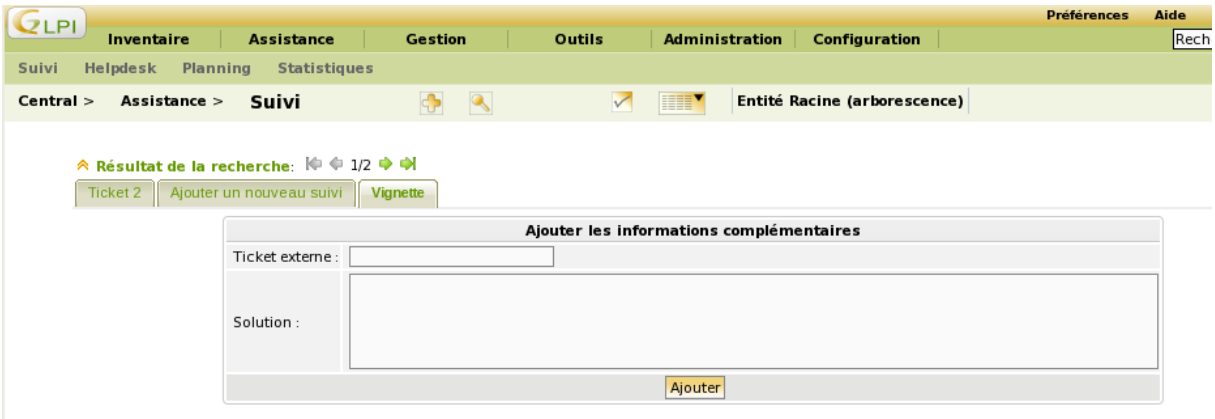
4.3.2.Formulaire de gestion des champs complémentaires

La fonction **plugin_headings_vignette()** a pour vocation de compléter le formulaire de Suivi d'un ticket avec nos champs complémentaires ! Les extraits de code suivants illustrent ce que la fonction doit produire : un formulaire HTML !

```
function plugin_headings_vignette($type, $ID, $withtemplate=0)
{
  global $CFG_GLPI, $DB;
  global $LANG;

  // $ID est le numéro du ticket
  if (!$withtemplate || $type != TRACKING_TYPE) return ;

  echo "<div align='center'>";

  // Initialise les libellés :
  $title = "Ajouter";
  $but_label = "Ajouter";
  $but_name = "add";
  $solution = "";
  $extticket = "";
  $extID = "";

  // Lit les informations actuelles :
  $query = "SELECT * FROM glpi_plugin_vignette_extinfo ".
                    "WHERE id_tracking = '$ID'";

  if ($result = $DB->query($query)){
    if ($DB->numrows($result) > 0) {
       $row = $DB->fetch_assoc($result);
       if (!empty($row['vignette_solution'])) {
          $title = "Modifier";
          $but_label = "Modifier";
          $but_name = "modify";
          $solution = $row['vignette_solution'];
```

```
         $extticket = $row['vignette_extticket'];
         $extID = $row['ID'];
      }
   }
  }

  // Affichage du formulaire :
  echo "<form action=\"".$CFG_GLPI["root_doc"].
       "/plugins/vignette/front/plugin_vignette.form.php\" method='post'>\n";
  echo "<input type='hidden' name='id_tracking' value='$ID_tracking'>";
  echo "<input type='hidden' name='ID' value='$extID'>";

  echo "<table class='tab_cadre' style='margin: 0; margin-top: 5px;'>\n";
  echo " <tr><th colspan='2'>$title</th></tr>\n";
  echo " <tr class='tab_bg_1'>";
  echo " <td>Ticket externe : </td>":
  echo   <td><input type='text' name='vignette_extticket' ".
                     "size='20' value='$extticket'></td>";
  echo " </tr>";
  echo " <tr class='tab_bg_1'>";
  echo " <td>Solution : </td>";
  echo "<td><textarea name='vignette_solution' rows='4' cols='100'>".
                                  "$solution</textarea></td>";
  echo " </tr>";
  echo " <tr class='tab_bg_2'>";
  echo "  <td colspan= '2' align='center'>";
  echo "<input type='submit' name='$but_name' class='submit' ".
                                          "value='$but_label'></td>";
  echo " </tr>\n";
  echo "</table>\n";

  echo "</form>\n";
  echo "</div>";
}
```

La sélection de l'onglet "Vignette" affiché avec le formulaire de Suivi d'un ticket, provoque l'appel de notre fonction et affiche notre nouveau formulaire comme montré ci-dessous :



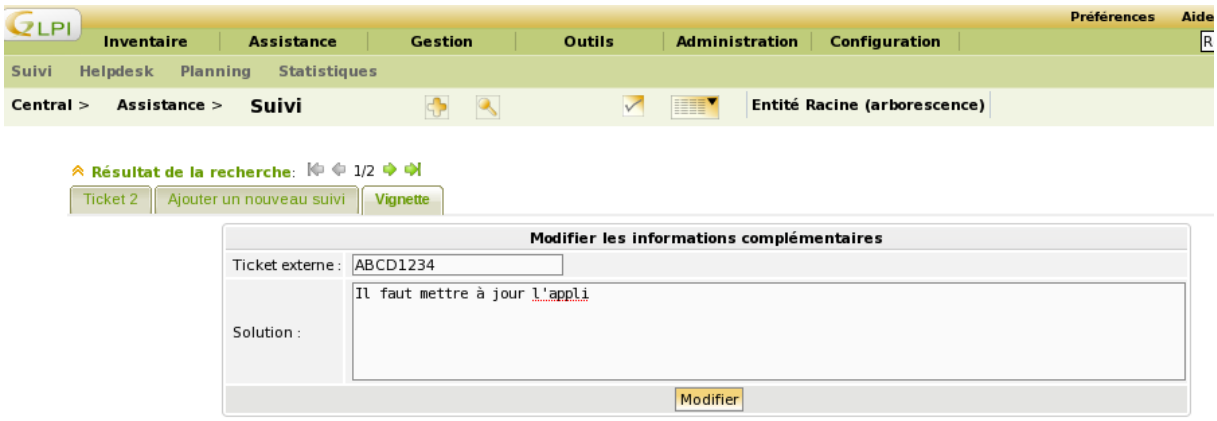
(écran 9)

Nous pouvons ajouter de nouvelles informations en appuyant sur le bouton <Ajouter> ce qui appelle le script **$ROOT/plugins/vignette/front/plugin_vignette.form.php** assez simple, dont le rôle est de stocker les nouvelles informations en base. Pour cela la classe de base **CommonDBTM** dont dérive la classe **plugin_Vignette** va nous être d'un grand secours.

Le code de **plugin_vignette.form.php** est :

```
<?php
define('GLPI_ROOT', '/opt/glpi');
include_once (GLPI_ROOT."/inc/includes.php");
include_once "../inc/plugin_vignette.classes.php";

checkRight('config',"w");

$vignette = new plugin_Vignette();
if (isset($_POST["add"])) {
        $newID = $vignette->add($_POST);
        glpi_header($_SERVER['HTTP_REFERER']);
}
elseif (isset($_POST["modify"])) {
        $newID = $vignette->update($_POST);
        glpi_header($_SERVER['HTTP_REFERER']);
}
else {
        glpi_header("../index.php");
}
?>
```

On voit qu'une fois le formulaire traité, la fonction globale **glpi_header()** réinvoque le script d'affichage initial, ce qui provoque maintenant l'affichage suivant :



(écran 10)

eh oui ! on détecte bien qu'il y a dorénavant des informations rattachées à ce ticket. Ce que prouve aussi le contenu de la base :

```
mysql> select * from glpi_plugin_vignette_extinfo;
+----+-------------+--------------------+------------------------------+
| ID | id_tracking | vignette_extticket | vignette_solution            |
+----+-------------+--------------------+------------------------------+
|  1 |          22 | ABCD1234           | Il faut mettre à jour l'appli |
+----+-------------+--------------------+------------------------------+
1 row in set (0.00 sec)
```

4.3.3.Affichage des vignettes et copies d'écran

Comme nous l'énoncions en introduction, nous voulons aussi pouvoir attacher à un ticket des copies d'écran. GLPI le permet déjà puisqu'on peut télécharger n'importe quel type de fichier avec un ticket. Cependant, GLPI ne propose pas de pré-visualisation si ces fichiers sont des images.

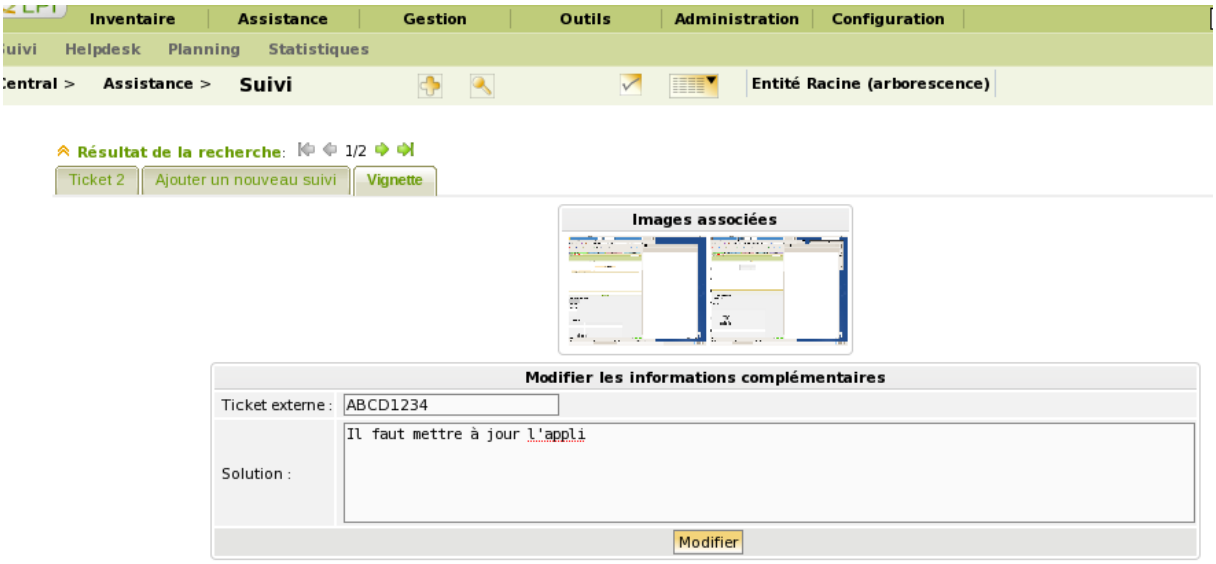
La modification à apporter est assez simple : il faut modifier la fonction **plugin_headings_vignette()** du §4.3.2 pour qu'elle affiche les vignettes correspondant aux images associées au ticket.

Sachant que les associations entre les tickets et les fichiers attachés sont décrites dans la table **glpi_docs** et que les fichiers sont stockés sous le répertoire **$ROOT/files**, il suffit de rajouter la portion de code suivante :

```
function plugin_headings_vignette(...)
{
  ....
  $query = "SELECT * FROM glpi_docs WHERE FK_tracking = $ID_tracking";
  if ($result = $DB->query($query)){
    if ($DB->numrows($result) > 0) {
      echo "<table class=\"tab_cadre\" style=\"margin: 0; ".
                                        "margin-top: 5px;\">\n";
      echo "<tr><th colspan=\"2\">Images associ&eacute;es</th></tr>\n";

      echo "<tr><td>\n";
      while ($row = $DB->fetch_assoc($result)) {
         # Tester les PNG, GIF, PJG
         $fname = GLPI_ROOT."/files/".$row['filename'];
         if (@getimagesize($fname) === FALSE) continue;
         $type = dirname($row['filename']);

         echo "<a target=\"_blank\" href=\"$fname\">".
             "<img src=\"".$CFG_GLPI['root_doc'].
             "/plugins/vignette/front/plugin_vignette_thumbnail.php?image=".
             urlencode(basename($fname)).
             "&type=$type\" width=100 height=80></a> \n";
      }
      echo "</td></tr>\n";
      echo "</table>\n";
    }
  }
  ...
}
```

Le code du script **plugin_vignette_thumbnail.php**, qui construit les vignettes "à la volée", est disponible sur le site [MAJE1].

L'écran ci-dessous montre l'affichage du ticket auquel on a adjoint deux images au format PNG :



(écran 11)

Et en cliquant sur une vignette, on provoque l'affichage dans une nouvelle fenêtre de l'image complète.

4.4.Cas de destruction de tickets - Hook "delete"

Si vous avez mis en oeuvre le code proposé jusqu'ici, ou si tout simplement, vous avez compris le principe de notre plugin, vous constatez qu'on est donc capable de rattacher des informations complémentaires et des vignettes à un ticket donné.

Mais que se passe-t-il en cas de destruction d'un ticket ? Et bien, GLPI ne se "souvient" pas que nous avons attaché des informations complémentaires à un ticket - d'ailleurs, si vous détruisez un ticket, vous constaterez que les pièces jointes ne sont pas supprimées non plus (par choix d'implémentation sans aucun doute) !

Il nous incombe donc de nettoyer la table **glpi_plugin_vignette_extinfo** lorsqu'on détruit un ticket. C'est ici que le hook **item_delete** prend toute son utilité. Dans le fichier **setup.php**, nous définissons la fonction suivante, qui sera appelée avec comme paramètre, l'ID du ticket détruit. Il nous faut ensuite récupérer l'ID correspondant dans notre table, car la fonction **deleteFromDB()** de la classe **CommonDBTM** effectue toujours des destructions en utilisant une colonne ID comme critère. Ce qui nous donne :

```
function plugin_vignette_item_delete($parm)
{
    if (!isset($parm['ID']))
          return true;

    switch ($parm['type']) {

    case TRACKING_TYPE:
      // A partir de l'ID du ticket, il faut retrouver l'ID dans notre table !
      $db = new plugin_Vignette();
      if ($vid = $db->getVIDFromTID($parm['ID']))
           $db->deleteFromDB($vid);
           break;
   }

    return true;
```

```
}
```

et donc nous rajoutons la fonction **getVIDFromTID()** dans le fichier **inc/plugin_vignette.classes.php** :

```
public function getVIDFromTID($tid)
{
    global $DB;

    $query = "SELECT * FROM ".$this->table." WHERE id_tracking = $tid";
    if ($result = $DB->query($query)) {
          if ($DB->numrows($result) == 1) {
                $fields = $DB->fetch_assoc($result);
                return $fields['ID'];
          }
    }

    return NULL;
}
```

5.Autres extensions possibles

5.1.Formulaire de recherche

GLPI propose des hooks pour que les plugins proposent leur propre formulaire de recherche.

Tout d'abord, il faut écrire la fonction **plugin_vignette_getSearchOption()** qui va lister les champs qui seront affichés dans le formulaire de recherche.

Chaque champ est décrit par :

- le nom de la table où il est stocké (attribut 'table')
- son nom de champ au sens SQL (attribut 'field')
- un nom de champ qui est une clé étrangère (champ 'linkfield')
- un libellé pour l'affichage (attribut 'name')

Puis il faut invoquer la fonction **searchForm()** qui affiche le formulaire. Une fois la sélection effectuée par l'utilisateur, le fonction **showList()** est appelée. La construction de la requête SQL est à sa charge.

Comme l'algorithme de recherche est basé sur une requête SQL, GLPI permet de définir des fonctions **plugin_vignette_addLeftJoin()**, **plugin_vignette_forceGroupBy()**, **plugin_vignette_addWhere()**, **plugin_vignette_addHaving()**, **plugin_vignette_addSelect()** et **plugin_vignette_addOrderBy()**, afin de créer la requête SQL correspondant exactement à nos besoins.

5.2.Exportations de données

Notre table **glpi_plugin_vignette_extinfo** est sauvegardée automatiquement par GLPI lors de l'exportation des données au format SQL ou XML (menu *Central* > *Administration* > *Données*).

GLPI permet de générer des rapports dynamiques, il

5.3.Autres possibilités

- les modifications "massives" : dans le langage de GLPI, c'est la possibilité d'appliquer une action à un ensemble d'objet sélectionnés dans une liste
- l'intégration avec le planning d'intervention
- gérer les champs de saisie de type "liste à sélection" : pour GLPI, il s'agit là de champs "dropDown" et GLPI permet aisément la mise en relation entre le contenu d'une table et une liste de saisie

Ces autres fonctionnalités pourront faire l'objet d'un 2ème article si ce 1er article vous a intéressé...

Conclusion

Dans cet article nous avons tenté d'éclaircir les informations disponibles sur le site [GLPI1] en les illustrant avec un cas concret. Un autre bon moyen de développer son propre plugin consiste à étudier quelques-uns des nombreux plugins déjà développés et proposés sur le site de GLPI.

Comme l'API de GLPI est très riche (voir référence [GLPI2]), nous vous conseillons aussi d'étudier le contenu du répertoire **./inc**. Il regorge de fonctions et de classes qui permettent d'accéder aux objets de GLPI et assurent des traitements communs.

Finalement, vous n'oublierez pas de placer GLPI en mode "debug" (menu *Central* > *Configuration* > *Générale*), vos erreurs PHP seront alors affichées ainsi que les valeurs des tableaux **$_SESSION**, **$_POST** et **$_GET**, sans oublier les requêtes SQL.... que du très utile ! A vos claviers et bonne extension !

Auteur :

Jérôme Delamarche
jdelamarche@maje.biz

L'auteur remercie Jean-Mathieu Doléans, co-auteur du projet GLPI pour ses conseils avisés et la relecture de cet article.

Références

[GLPI1] Développement de plugin pour GLPI
https://dev.indepnet.net/plugins/wiki/CreatePlugin

[GLPI2] GLPI Documentation
https://dev.indepnet.net/glpidoc/index.html

[GLPI3] Valeurs prédéfinies des types de plugin
https://dev.indepnet.net/plugins/wiki/PluginTypesReservation

[GLPI4] Migrer un plugin de la v0.71 vers la v0.72
https://dev.indepnet.net/plugins/wiki/MigratePluginFrom071to072

[MAJE1] Code source de ce plugin
http://www.maje.biz/downloads/glpi_vignette.tar.gz