Tutorial plugin system forma.lms

Introduction

In this tutorial I will explain you how to *pluginize (convert into a plugin)* a functionality of *forma.lms*. Firstly let's understand how the main classes (PluginManager and PluginAdm) work.

PluginAdm

PluginAdm is the class used to manage the
method of each plugin. The main methods is:

```
getPlugins( $onlyActive = false )
```

It returns an array containing all the plugins, if $onlyActive is set to true it returns only active plugins.

All the other functionalities are not called inside forma.lms
(except for the installPlugin method called inside the installer)

PluginManager

PluginManager is the class used by forma.lms to load the plugins features. It has this __construct

__construct($category)
Given a category $category it initialize the instance.

You can load or run plugins using these methods of an instance of the PluginManager class.

run_plugin($plugin, $method, $parameters = array())
Given a plugin name $plugin and a method $method it runs the static method of the specified plugin of the category set into the constructor. You can specify parameters passing it through $parameters.

run($method, $parameters = array())
Given a method $method it runs all the static method of the category set into the constructor. You can specify parameters passing it through $parameters.

get_plugin($plugin, $parameters = array())
Given a plugin $plugin it returns an instance of the specified plugin of the category set into the constructor passing $parameters into its constructor.

Plugin structure

A plugin is composed by three main type of file:

- Plugin.php: contains the callback that are executed during installation and activation.
- manifest.php: contains the information of the plugin, the most important parameter is <name> because it must be the same as the folder name or forma.lms will not read the plugin.
- *_CATEGORY_NAME*.php: A plugin can have multiple category files: one for each functionality it is extended.

First Example

We will try to *pluginize* the learning object (*LO*) functionality of forma.lms.

Inside html/appLms/class.module/class.organization.php in the class Organization_Module we can see that inside each "controller" (I mean the different actions inside the switch) of the method loadBody there is a function used to get the instance of the specified *LO*: createLO.

html/appLms/lib/lib.module.php

```
createLO( $objectType, $idResource = false, $environment = false )
```

Inside we can see a query in which we get the name of the file containing the class of the specified *LO* (inside **$objectType**).

```
35  d$query = "SELECT className, fileName FROM %lms_lo_types WHERE
    objectType='".$objectType."'";
36  $rs = sql_query( $query );
37  list( $class_name, $file_name ) = sql_fetch_row( $rs );
38  if (trim($file_name) == "") return false;
```

Then the file is included and it is created a new instance.

```
45  require_once(dirname(__FILE__).'/../class.module/learning.object.php'
    );
46  if (file_exists(_base_ .
    '/customscripts/'._folder_lms_.'/class.module/'.$file_name ) &&
    Get::cfg('enable_customscripts', false) == true ){
47      require_once(_base_ .
    '/customscripts/'._folder_lms_.'/class.module/'.$file_name );
48  } else {
49      require_once(Docebo::inc(_lms_.'/class.module/'.$file_name));
50  }
51  $lo = new $class_name($id_resource, $environment);
52  return $lo;
```

In first place we don't need a table no more. We will search for the LO directly from the active plugins. Secondly we don't need to check if folders exists and include files because the plugin manager handles this for us. So the final code will be:

```
35  $plugin_manager = new PluginManager('lo');
36  $lo = $plugin_manager->get_plugin($objectType, array($id_resource,
    $environment));
37  return $lo!==false ? $lo : false;
```

Now we can proceed in converting every single *LO* into a plugin. In this example we will *pluginize* the *htmlpage* object. Let's create the `manifest.xml` file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest>
  <title>HTML Page</title>
  <author>Tutorial</author>
  <name>htmlpage</name>
  <version>1.0</version>
  <category>lo</category>
  <description>HTML Page LO plugin</description>
</plugin_manifest>
```

Now let's create the `Plugin.php` file.

```php

<?php
namespace Plugin\htmlpage;
defined("IN_FORMA") or die('Direct access is forbidden.');
class Plugin extends \FormaPlugin {

}
```

In order to make the htmlpage *LO* work, we need to load the table that it needs to store data. So we will use the install callback to insert the table in the database. Forma will call this method each time the plugin is installed.

```php
<?php
namespace Plugin\htmlpage;
defined("IN_FORMA") or die('Direct access is forbidden.');
class Plugin extends \FormaPlugin {

  public function install(){
      $query="  CREATE TABLE `learning_htmlpage` (
                  `idPage` int(11) PRIMARY KEY AUTO_INCREMENT,
                  `title` varchar(150) NOT NULL DEFAULT '',
                  `textof` text NOT NULL,
                  `author` int(11) NOT NULL DEFAULT '0'
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 ";
      if (sql_query($query)){
          return true;
      }
      return false;
  }

}
```

Then we remove the table same when we uninstall the plugin.

```php
<?php
namespace Plugin\htmlpage;
defined("IN_FORMA") or die('Direct access is forbidden.');
class Plugin extends \FormaPlugin {

  public function install(){
      $query="  CREATE TABLE `learning_htmlpage` (
                  `idPage` int(11) PRIMARY KEY AUTO_INCREMENT,
                  `title` varchar(150) NOT NULL DEFAULT '',
                  `textof` text NOT NULL,
                  `author` int(11) NOT NULL DEFAULT '0'
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8 ";
      if (sql_query($query)){
          return true;
      }
      return false;
  }

  public function uninstall(){
      $query="DROP TABLE IF EXISTS `learning_htmlpage` ";
      if (sql_query($query)){
          return true;
      }
      return false;
  }

}
```

The last file that we have to create is the file that will contain all the features of *LO*. We call it lo.php.
The class that manage the htmlpage learning object is Learning_Htmlpage located in appLms/class.module/learning.htmlpage.php. We have to copy all the methods inside our newly created lo.php file.

```php
<?php
namespace Plugin\htmlpage;

class lo extends \FormaPlugin {
   //copy all methods here,
   //remember the namespace and the new name of this class
}
```

We have to do some fix to the code, for example check that the namespace is correct (we are in *Plugin\htmlpage*) and the name of the construct method is updated.
After that go to forma administration (under configuration/plugin manager and activate the new plugin).
I'm not including in this tutorial the tracking part, but now if you try to create a new htmlpage you can see that forma.lms will use our new plugin.