forma.lms - doc plugin system

Index

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 initializes the instance (see below for availables categories).

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.

PluginManager has a static method to load entire new forma.lms functionalities.

get_feature($mvc_app, $mvc_name)

*Given a mvc app (for example lms or adm) $mvc_app and an mvc name $mvc_name, it returns an instance of the controller linked to the specified mvc app and name searching in the table **core_requests**.*

Plugin structure

A plugin is composed by:

- Plugin.php: contains the callback that are executed during installation and activation.
- manifest.xml: contains the information of the plugin, **!important** the parameter <name> 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.
- *Event*.php: contains all the hooks of the plugin.
- db a folder containing the scripts to be executed during installation, uninstallation, activation, deactivation. For example:
  - activate.sql
  - install.sql
- translations a folder containing the xml of the languages. The files must be like the standard translations of forma. For example:
  - lang[italian].xml

Plugin categories

A plugin can have multiple purposes, it can implement:

- Event: add listeners to events
- Feature: add a new feature to platform
- "Standard category": a "standard" interface (authentication, conference, report, …)

Feature category

A folder called features containing a replica of forma.lms structure in order to implement the new features of the plugin. For example:

- features
  - appLms
    - controllers
      - DummyLmsController.php
    - models
      - DummyLms
    - views
      - dummy
        - show.html.twig
  - lib
    - lib.dummy.php

Event Category

You can hook events in forma.lms using the file Event.php. Write all the addListener in the file like this:

```php
<?php

\appCore\Events\DispatcherManager::addListener('core.dummy.event',
function($event) {
   echo "event-ciao";
});
```

Plugin.php

The plugin.php file is used to manage the core of the plugin, you can:

- Declare function that are called during installation, uninstallation, activation, deactivation. Simply doing:

```php
<?php
namespace Plugin\test;
defined("IN_FORMA") or die('Direct access is forbidden.');

class Plugin extends \FormaPlugin {
   public function install(){
      //code executed after installation
   }
   public function uninstall(){
      //code executed after uninstallation
   }
   public function activate(){
      //code executed after activation
   }
   public function deactivate(){
      //code executed after deactivation
   }
}
```

- Using parent's method to:
  - load settings;
  - get plugin's name;
  - add requests;
  - add menus

```php
public function install(){
   //addSetting( key , type , size )
   parent::addSetting('test_key', 'string', 255);

   // get plugin name
   parent::getName();

   //addCoreMenu( name , mvcPath , parent (the name of the parent) , icon , is_active )
   parent::getCoreMenu('_test_name', 'lms/test/test', 'lms/test/test', false, '', false);
}

public function activate(){
   //addRequest( app , name , controller , model )
   parent::addRequest("lms", "test", "testLmsController", "testLms");
}
```

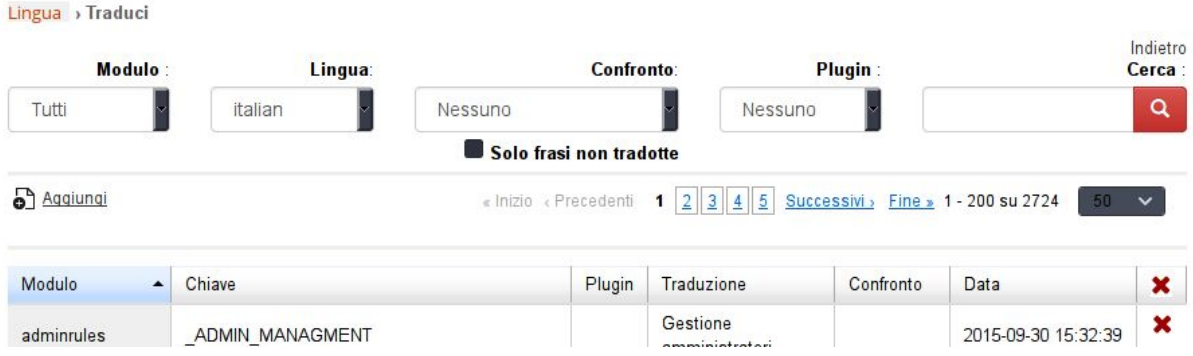
The namespace must be **Plugin\PLUGIN_NAME.**

Translations

You can manage the translations of your plugin using xml files in the translations folder. The files must be formatted as the translations file in forma.

```
<?xml version="1.0"?>
<LANGUAGES>
  <DATE>20170713</DATE>
  <LANG id="italian">
      <lang_code>italian</lang_code>
      <lang_description>Italiano</lang_description>
      <lang_charset>utf-8</lang_charset>
      <lang_browsercode>it</lang_browsercode>
      <lang_direction>ltr</lang_direction>
      <platform id="all">
      <!-- This will create new keys with a new module -->
      <module id="test">
      <key id="test&amp;amp;_TEST_A" attributes="" save_date="2015-09-30
15:32:39"><![CDATA[Prova 1]]></key>
      <key id="test&amp;amp;_TEST_B" attributes="" save_date="2015-09-30
15:32:39"><![CDATA[Prova 2]]></key>
      </module>
      <!-- This will override the standard translation (not in the
database, only in visualization) -->
      <module id="standard">
      <key id="standard&amp;amp;_NAME" attributes="" save_date="2017-03-14
16:10:41"><![CDATA[BBB]]></key>
      </module>
      </platform>
  </LANG>
</LANGUAGES>
```

In the language manager there is a new filter called “Plugin”. You can use it to view only the translations of the specified plugin.



manifest.xml

- *The name property must be the name of the folder*
- *The forma_version property is used to specify the compatibility with the minimum and the maximum version of forma.lms*
- *The dependencies property is used to specify the plugins that are necessary in order to install the plugin.*

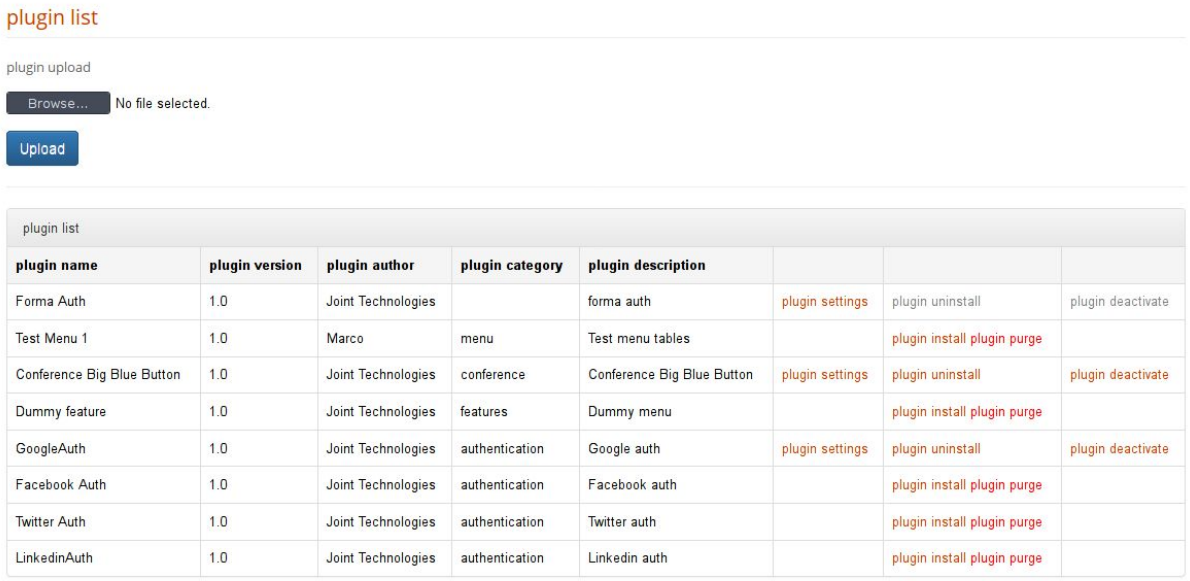
```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest>
    <title>Dummy feature</title>
    <author>forma.lms</author>
    <version>1.0</version>
    <category>features</category>
    <description>Dummy menu</description>
    <link>http://www.formalms.org/</link>
    <license>GNU General Public License version 2</license>
    <name>Dummy</name>
    <name>Dummy</name>
    <forma_version>
        <min>2.0</min>
        <max>2.0</max>
    </forma_version>
    <dependencies>
        <plugin1_name>1.0.0</plugin1_name>
        <plugin2_name>1.4.0</plugin2_name>
    </dependencies>
</plugin_manifest>
```

Inside forma.lms

You can find the plugin manager in

Admin area / Configuration / Plugin Manager

This is the table to manage all the plugins:



plugin list

plugin name	plugin version	plugin author	plugin category	plugin description			
Forma Auth	1.0	Joint Technologies		forma auth	plugin settings	plugin uninstall	plugin deactivate
Test Menu 1	1.0	Marco	menu	Test menu tables		plugin install plugin purge	
Conference Big Blue Button	1.0	Joint Technologies	conference	Conference Big Blue Button	plugin settings	plugin uninstall	plugin deactivate
Dummy feature	1.0	Joint Technologies	features	Dummy menu		plugin install plugin purge	
GoogleAuth	1.0	Joint Technologies	authentication	Google auth	plugin settings	plugin uninstall	plugin deactivate
Facebook Auth	1.0	Joint Technologies	authentication	Facebook auth		plugin install plugin purge	
Twitter Auth	1.0	Joint Technologies	authentication	Twitter auth		plugin install plugin purge	
LinkedinAuth	1.0	Joint Technologies	authentication	Linkedin auth		plugin install plugin purge	

- You can upload a new plugin (must be a zip containing the folder). **You need to set the "write" permission to the plugins folder.**
- You can install, uninstall, activate, deactivate, manage the settings, and delete the files (purge) of each plugin. Except for the ones defined as ***core*** or are dependencies or dependance of other plugins. A core plugin cannot be uninstalled.

New requests system

Inside forma's database there is a new table called ***core_requests***.
It is used by forma to load controllers inside index files.
This is an example:

You want forma to call DummyLmsController::show() (loading the model DummyLms.php) if the request is *index.php?r=lms/test/show.*

app	**name**	**controller**	**model**	**plugin**
lms	test	DummyLmsController	DummyLms	EXTERNAL_ID

Forma will break the request *index.php?r=lms/dummy/show* into

- lms
- dummy
- show

Find into the table an occurrence for **app = 'lms'** and **name = 'dummy'**. If a plugin ID is specified it will check if the plugin is enabled and then include the controller and model files.

PluginmanagerAdm

PluginmanagerAdm is the class used to manage the core 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.

Backwards compatibility

<todo>