GLPI plugins Documentation

Release 0.1

Teclib'

Mar 06, 2018

Contents

You'll find here user documentation for various GLPI plugins.

CHAPTER 1

Presentation

Plugins are a way to extend GLPI possibilities without modifying core source code. In a perfect world, plugins should provided specific features; while core's may provided features everyone (or almost!) mays find usefull.

Some of the existing plugins are very complex, other are very simple. Some may require you to read their documentation before using, some does not... Some may provide you configuration possibilities, acces rights management, and so on.... Or not! There are so many plugins already!

If you're looking for a feature that does not exists yet in GLPI's core, take a look at the plugins repository. You can search for a plugin name or a feature, see what core's versions are compatible, and so on.

The goal of the present documentation is to centralize documentations, but we cannot pretend all plugins documentations are hosted here.

Anyways, happy GLPI-ing!

CHAPTER 2

Empty

- Sources link: https://github.com/pluginsGLPI/empty
- Download: *not relevant*

This plugins does... nothing. Really!

This is designed to be a start point for writing plugins, with very minimal defaults usefull scripts, and some advices. If you're looking for plugins possibilities, take a look at *the example plugin*.

2.1 Create a new plugin

An utility script to create a new plugin - `plugin.sh` is provided. You will call it with a plugin name, a version, an optionally the path where your want it to be created:

```
$ ./plugin.sh MyGreatPlugin 0.0.1
```

The script will sanityze and lowercase the name you provided, copy the templates files into the new directory, and then make some replacements.

Using the script without destination parameter, it will create you directory plugin, `mygreatplugin` just beside its own directory. Otherwise, it would create the new directory in the specified path:

```
$ ./plugin.sh MyGreatPlugin 0.0.1 /path/to/glpi/plugins/
```

2.2 Update existing plugin

There is no automated way to update an existing plugin, because there would be too many cases to handle. But don't worry, procedure is quite simple ;)

Using empty features is as simple as creating a few files:

- `composer.json`,

- `.travis.yml`,
- `Robofile.php`,
- `.gitignore`.

If you do not have yet any composer or travis configuration file, you can just copy the ones from empty plugin. Otherwise; in you `composer.json`, just add:

```
{
  "minimum-stability": "dev",
  "prefer-stable": true
}
```

And then run `composer require glpi-project/tools`.

In the travis configuration file, just add the CS call:

```
script:
  - vendor/bin/robo --no-interaction code:cs
```

In the `.gitignore` file, add the following:

```
dist/
vendor/
.gh_token
*.min.*
```

As for the Robo.li configuration file, note that the one embed in the empty plugin is a bit specific, you'll have to edit it in order to get things working. See below for more informations.

Finally, as the `tools` project will provide you some features, you can remove duplicated tools scripts (files such as `release`, `extract_template.sh`, ...) that would be present in your plugin.

2.3 Features

2.3.1 Coding standards

The GLPI PHPCodeSniffer rulesets are provided as `vendor/glpi-project/coding-standard/GlpiStandard/`.

To check coding standards, just use the Robo.li task `code:cs`:

```
$ ./vendor/bin/robo code:cs
```

Note: The above command will ignore `vendor` and run on the current directory.

If you want to adapt ignore list or checked directories, you can just override `$csignore` and/or `$csfiles` in the `RoboFile.php` of the plugin:

```
<?php

class RoboFile extends Glpi\Tools\RoboFile
{
   protected $csignore = ['/vendor/', '/lib/'];
   protected $csfiles  = ['./', 'setup.php.tpl']
```

```
    [...]
}
```

2.3.2 Automated checks

For convenience; a `.travis.yml` file is also provided, that is setup to:

- check coding standards,
- run on a bunch on different configuration

You still have to enable travis-ci builds from the website in order to activate automated tests.

Of course, the `.travis.yml` file can be pimped; you can run unit tests, create/update a database, activate notifications, and so on. Refer to the Travis-CI documentation to know more.

2.3.3 Minifying CSS ans JS

A convenient script, using Robo.li is provided. The `RoboFile.php` file is an empty class that extends `Glpi\Tools\RoboFile` (provided by `glpi-project/tools` dependency) in which you can set your own stuff.

That way, you can quite easily update the common file and get your own tasks remaining the same.

To get the required libs installed, you'll have to get composer installed and then run:

```
$ composer install -o
```

There are three available targets:

- `minify` that will minify all CSS and JS files (see below),
- `minify:css` that will minify all css stylesheets files in the `css` directory of your plugin, creating a `.min.css` version along with the original file,
- `minify:js` that will minify all javascript files in the `js` directory of your plugin, creating a `.min.js` version along with the original file.

Just choose a target, and run something like:

```
$ ./vendor/bin/robo minify:css
```

Note: Remember compiled files should not be commited on your repository sources. It is a release process to minify files.

Also remember to adapt your scripts so they load your minified versions if available, and the original one otherwise :)

As of GLPI 9.2; you do not have to care about loading minified files when using `add_css` and `add_javascript` hooks! You just need to call not minified script; and GLPI will use the minified version if it exists and if not in *DEBUG* mode.

2.3.4 Translations

GLPI and its plugins use gettext for internationnalization. Several steps are required before that can work:

1 translatable strings must be extracted from source files, a `POT` file will be created or updated accordingly, 2 `PO` files must be created or updated from the `POT` file, 3 `PO` files must be translated, 4 `MO` files must be compiled from the latest `PO`.

In the `vendor/bin` directory, you'll find a `extract_template.sh` script. This is designed to extract translatable strings from your source code (see first point above).

Once it has been ran, a `locale/mygreatplugin.pot` file will be created/updated.

For the second and third step, you'll have to make a choice. You can use gettext tools to update your `PO` files and translate them using a dedicated tool, like poedit; or you can use an online translation system like Transifex or Zanata. GLPI core and many of existing plugins are translated using Transifex right now.

Once you get your updated `PO` files, you'll have to compile them to `MO` files. You can run it manually, the release script will compile them again anywways; see the *compiling MO files section*.

2.3.5 Release script

A release script is provided in `vendor/bin/plugin-release`. This is a "simple" Python script; you should just have Python installed on your system (this is installed by default on most linux distributions).

Warning: Note that for the moment, the release script is only compatible if you use semantic versionning!

Using just the defaults, the script will try to retrieve the latest tag in your git repository, add third party dependencies and create a *Release* on the github project:

```
$ ./vendor/bin/plugin-release
Do you want to build version 1.9.5? [Yes/no] y
Building glpi-order-1.9.5...
Archiving GIT tag 1.9.5
Adding vendor libraries
$ ls dist
glpi-order-1.9.5.tar.bz2  glpi-order-1.9.5.tar.bz2.asc
```

Requirements

You will need a python interpreter installed as well as the following modules:

- termcolor,
- gitdb,
- github (to check for existing versions in also in drafts, and to create github releases), unless you use the `--nogithub` option

If you want to get help on the script, try to run `./vendor/bin/plugin-release -h`.

Process

The release process will achieve the following tasks for you:

- check if the version constant is the same as the tag you've requested;

- check if the version in the website XML file is the same as the tag you've requested;
- check if a release already exists, locally, and remotely (assuming your project is hosted in the *pluginsGLPI* organization and the release is public);
- make a *git archive* of the paths that are not excluded (`.git`, `tools`, `tests`, ...);
- if any, install composer dependencies;
- if any, compile you `MO` files;
- if any, compile you CSS stylesheets and your Javascript files (using Robo.li);
- create a release archive with all that; that will be available in the `dist` directory;
- use GPG to sign the archive.

Note: The standard release process will not work on your files directly, it will make a copy in the `dist/src` directory before. The only exception is the *MO compiling option*.

In order to check if all is OK before doing real release; create your tag and run `./vendor/bin/plugin-release -C` **before pushing your tag**. That way, you'll be able to fix potential issues and re-create your tag locally (remember published tags should **never** be removed).

Compiling MO files

The release process will automatically compile every `PO` file it will found in your `locales` directory. But you probably want the sources to contain the latests `MO` files, for testing purposes. The release script provide the `--compile-mo` (or `-m`) to achieve that:

```
$ ./vendor/bin/plugin-release --compile-mo
```

Warning: The above command will work on your plugins files directly; not on a copy as does other commands.

Pre-releases

Per default, the release script will work only on existing tags. Any pre-release should have its own tag; but you may want to create a release archive without any tags in some circumstances.

In order to tell the release script what it should archive, you'll have to specify several parameters:

- `--commit` (or `-c`) giving the commit hash,
- `--release` (or `-r`) giving the release version (usually, it will be the next release version),
- `--extra` (or `-e`) to specify an extra string (such as *alpha*, *beta*, *rc1*, etc...)

As an example with the *order* plugin:

```
$ ./vendor/bin/plugin-release --commit 632d515d4ac0 --release 1.9.5 --extra alpha1
$ ls dist
glpi-order-1.9.5-alpha1-20161103-632d515d4a.tar.bz2
```

Signing releases

Signing releases with a GPG key would permit users to check download integrity before installing. You'll need a GPG key publically available to users; the sign option is activated per default, you can deactivate using the `--nosign` (or `-S`) option.

A file containing the signature with the same name as the archive with a `.asc` extension will be created in the `dist` directory.

GitHub release

The release script will create a release on your GitHub repository, as a draft, unless you use `--nogithub` (or `-g`) option.

Note: Unfortunately, I was not able to get the newly created archive uploaded to this new release... Maybe that could be fixed in the future.

In order to use this feature, you will need the github installed; and you will need an access token. Access token is valid per user, and gives accesss to all his repositories.

You'll have to go to your github account settings page, in the personnal access token tab. Click on *generate new token*, give the description you want, and make sure you'll check the *public_repo* box only (no need to check anything else, you can create several access token if you need).

The token will be displayed only once; store it in the `.gh_token` file in your plugin directory; and that's all!

CHAPTER 3

Example

- Sources link: https://github.com/pluginsGLPI/example
- Download: *not relevant*

An example plugin... This is designed to show you various possibilities that are offered to plugins from GLPI core.

As this plugin does nothing in reality, there is no documentation for it. You would like to refer to *plugins presentation*, plugins developer manual or even *empty plugin* to get started.

CHAPTER 4

Generic Object

- Sources link: https://github.com/pluginsGLPI/genericobject
- Download: https://github.com/pluginsGLPI/genericobject/releases

This user manual applies to version 2.5 of the GLPI Generic Object Plugin.

4.1 Requirements

This plugin requires :

- PHP 5.3 or higher
- GLPI >= 9.2

4.2 Features

This plugin allows you to add new inventory objects types, integrated into GLPI framework.

It supports following GLPI features:

- entity and sub-entities management;
- search;
- templates;
- history;
- helpdesk integration;
- CSV file injection plugin integration;
- Item uninstallation plugin integration;
- order management plugin integration.

4.3 Example of usage

Objective: Manage your car fleet like the rest of your IT Assets.

- Create a new type of inventory object *car*.
- Add the accurate fields for a *car*, like: *name*, *serial number*, *inventory number*, *type*, *model*, *color*, *state*, etc.
- Describe the behaviour of a *car*: visible in subentity, retain history, etc.
- Adjust the rights on *cars*.
- Activate the *cars* object.
- Manage your collection of *cars* in GLPI.

4.4 Install the Plugin

- Uncompress the archive.
- Move the `genericobject` directory to the `<GLPI_ROOT>/plugins` directory
- Navigate to the *Configuration* > *Plugins* page
- Install and activate the plugin

4.5 Usage

4.5.1 Create a new object type

This is the first step.

- Click on the + button in the plugin configuration form.
- Create the new type of inventory object:
  - *name*: mandatory, lowercase, and must be composed of letters only;
  - *label*: by default, the same as the name.
- Validate.
- Activate the new item type to use it.

Example: Create a new type of inventory object *car*.

4.5.2 Edit labels

For each type, a language file is available in `<GLPI_ROOT>/files/_plugins/genericobject/locales/itemtype/`

The plugin creates :

- a language file for the current language
- a language file for the default GLPI language

Note: If the current and default languages are the same, only one file is created.

To change the label of the itemtype, for the english language, edit the file:

```
<?php
// <GLPI_ROOT>/files/_plugins/genericobject/locales/<itemtype>/<itemtype>.en_GB.php
$LANG['genericobject']['<itemtype>'][1] = "<type's label>";
```

4.5.3 Setup behaviour

Example: Describe the behaviour of a *car*: visible in subentity, retain history, etc.

The new type will be managed the same way as the usual GLPI types (computer, monitor, network device, etc.)

Note: All objects are at least assigned to an *entity*

The Behaviour tab allows you to define:

- *child-entities:* allows the type to be recursive;
- *Helpdesk:* allows an object to be associated to a ticket;
- *Trash:* use GLPI's trash functionnality;
- *Notes:* use GLPI's note functionnality;
- *History:* allow history for this type;
- *Templates:* allows template management;
- *Documents:* allows documents to be attached to an object of this type;
- *Loans:* allows objects to be loaned;
- *Contracts:* link an object to one or more contracts;
- *Network connections:* allow ports to be used and management for this type;
- *CSV file injection plugin:* allows this type to be available for use in the plugin;
- *Item uninstallation plugin:* allows this type to be uninstalled;
- *Order management plugin:* allows this type to be linked to an order;

4.5.4 Add Fields

Example: Add the accurate fields for a *car*, like: *name*, *serial number*, *inventory number*, *type*, *model*, *color*, *state*, etc.

Navigate to the *Fields* tab.

The plugin comes with several ready to use fields:

- Name
- Type
- Model
- Serial number

- Inventory number
- Item's user
- Group
- Status
- Comments
- Notes
- Location
- Other
- Manufacturer
- URL
- Creation date
- Expiration date
- Category
- Visible in Helpdesk
- Technician in charge of the hardware
- Domain
- Contact
- Contact number

Note: Using some behaviour will automatically add some fields to the object:

- network connection => location
- loans => location
- helpdesk => is visible in Helpdesk
- notes => notepad

Helpdesk integration

To use an object in the helpdesk, use following setup:

- In *Behaviour* tab : *use helpdesk* must be set to **Yes**.
- if the *User* field is defined, it allows item to be visible in the *My Items* list (as item whose owner is the user).
- if the *Group* field is defined, it allows item to be visible in the *My Items* list too (as item belonging to a group in which the user belongs to).
- if *Helpdesk visible* field is set and if the value is set to **No** in the object, then the object won't be visible at all in the helpdesk.

4.5.5 Add new fields

Note: New fields will be available for all object's types.

- Create a new file named `<GLPI_ROOT>/files/_plugins/genericobject/fields/<type>.constant.php`

For example, for a *car* type the constant file will be `<GLPI_ROOT>/files/_plugins/genericobject/fields/car.constant.php`.

Please note that the file's first line must be the following, otherwise the new fields won't appear in the list:

```
<?php
global $GO_FIELDS, $LANG;
```

- Add the new fields definitions.

4.5.6 Add a simple dropdown field

```
<?php
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['name']       = $LANG[
↪'genericobject']["<type's name>"][2];
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['field']      = 'color';
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['input_type'] = 'dropdown';
```

Note: The language variable must be defined in the language file (see explaination above).

- Add a dropdown field that is assigned to an entity:

```
<?php
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['name']         = $LANG[
↪'genericobject']["<type's name>"][2];
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['field']        = 'color';
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['input_type']   = 'dropdown';
//Does the dropdown take care of entities ? (true/false)
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['entities_id']  = true;
//Can values be recursive ? (true/false, only taking in account if entities_id is set␣
↪to true)
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['is_recursive'] = true;
```

4.5.7 Add a tree dropdown field

```
<?php
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['name']       = $LANG[
↪'genericobject']["<type's name>"][2];
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['field']      = 'color';
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['input_type'] = 'dropdown';
//Is it a tree-dropdown, or a simple one ? (true/false)
$GO_FIELDS['plugin_genericobject_mytypecolors_id']['is_tree']    = true;
```

Note: You can use at the same time the following parameters : *entities_id*, *is_recursive*, *is_tree*.

4.5.8 Add a dropdown field that is based on a GLPI-core object (user, location...)

```
<?php
$GO_FIELDS['users_id_passengers_id']['name']       = 'Passenger';
$GO_FIELDS['users_id_passengers_id']['input_type'] = 'dropdown';
```

Note: Name between brackets (`[]`) **MUST** begin with `users_id` in order to be recognized as a field based on GLPI users' list.

See file `<GLPI_ROOT>/files/_plugins/genericobject/fields/field.constant.php` to get a complete list of available fields.

4.5.9 Add a global dropdown

A global dropdown can be used in all itemtypes. A good example would be :

```
<?php
$GO_FIELDS['categories_id']['name']          = $LANG['common'][36];
$GO_FIELDS['categories_id']['input_type']    = 'dropdown';
$GO_FIELDS['categories_id']['dropdown_type'] = 'global';
```

A specific category table will be created for each itemtype. The table name and field name will the computed this way:

- table : `glpi_plugin_genericobject_<itemtypename>_category`
- field name : `plugin_genericobject_<itemtype>categories_id`

4.5.10 Add an integer field

```
<?php
$GO_FIELDS['testinteger']['name']       = 'testinteger';
$GO_FIELDS['testinteger']['input_type'] = 'integer';
$GO_FIELDS['testinteger']['min']        = 10; //not mandatory, by default 0
$GO_FIELDS['testinteger']['max']        = 40; //not mandatory, by default 100
$GO_FIELDS['testinteger']['step']       = 3; //not mandatory, by default 1
```

4.5.11 Add a text field

```
<?php
$GO_FIELDS['mytextfield']['name']       = 'My text field';
$GO_FIELDS['mytextfield']['input_type'] = 'text';
```

Changed in version 2.1.2: By adding the following argument, you can tell the plugin that this field can be automatically generated when using a template:

```php
<?php
$GO_FIELDS['mytextfield']['autoname'] = true;
```

4.5.12 Add a Yes/No field

```php
<?php
$GO_FIELDS['mybooleanfield']['name']       = 'My boolean field';
$GO_FIELDS['mybooleanfield']['input_type'] = 'bool';
```

4.5.13 Add a date field

```php
<?php
$GO_FIELDS['creationdate']['name']       = $LANG['genericobject']['fields'][30];
$GO_FIELDS['creationdate']['input_type'] = 'date';
```

4.5.14 Add a date & time field

```php
<?php
$GO_FIELDS['creationdate']['name']       = $LANG['genericobject']['fields'][30];
$GO_FIELDS['creationdate']['input_type'] = 'datetime';
```

Note: If you don't want a field to be modified using massive actions, add the following line to its definition:

```php
<?php
$GO_FIELDS['myfield']['massiveaction'] = false;
```

4.6 Add global fields

To make your fields accessible to all itemtypes:

- Create a file named `<GLPI_ROOT>/files/_plugins/genericobject/fields/field.constant.php`
- Put your definitions in this file.

4.7 Setup Rights

You can define access rights for each object's type, for each profile. Available options are:

- *right on the type*: *no access*, *read*, *write*.
- *right to associate this type of object to tickets*: *yes*, *no*.

To associate the rights you can either:

- Use the *Rights* tab in the *itemtype* form.
- Navigate to *Administration* > *Profiles* and administer the rights for each profile.

4.8 Use the new field

Activate the new type, it's now ready to be used.

The new type is available for users in the *Plugins* > *Objects management* menu.

4.9 Use case of Generic Object as a CMMS

4.9.1 Purpose of this documentation

Showing a complete usage of Generic Object as a CMMS (Computerized Maintenance Management System) in biomedical environment.

At the end of this use case, you will have :

- a dedicated *Biomed* entity (under *Root entity*)
- containing *Biomedical* objects (in *Assets* menu)
- with built-in and user-defined fields
- manages by users with *Admin_biomed* profile

4.9.2 Steps

Following steps assume you have a Super-Admin authorization :

- Installing Generic Object on GLPI (validated with genericobject >= 0.85-1.0 and GLPI >= 0.90)
- Generic Object configuration
- GLPI configuration
- Start using Generic Object and GLPI

4.9.3 Installing Generic Object on GLPI

See *Install the Plugin* section.

4.9.4 Generic Object configuration

Create your type of object

See *Create a new object type* section and use *biomedical* as internal identifier. Label will be set automatically to *Biomedical* (with an uppercase *B*).

After a logoff/login, you will see *Biomedical* menu in Assets.

Define Biomedical's new fields

These fields will be usable only by Biomedical's objects :

- Create a new file named : `<GLPI_ROOT>/files/_plugins/genericobject/fields/biomedical.constant.php`
- Add following content :

```
<?php
global $GO_FIELDS, $LANG;

// CODE CNEH
$GO_FIELDS['plugin_genericobject_cnehcodes_id']['name']       = $LANG['genericobject
↪']['PluginGenericobjectBiomedical'][1];
$GO_FIELDS['plugin_genericobject_cnehcodes_id']['field']      = 'cnehcode';
$GO_FIELDS['plugin_genericobject_cnehcodes_id']['input_type'] = 'dropdown';

//  REFORME (yes or no)
$GO_FIELDS['reformed']['name']       = $LANG['genericobject']['
↪'PluginGenericobjectBiomedical'][2];
$GO_FIELDS['reformed']['input_type'] = 'bool';

// CLASSE CE (3 choix possibles 1,2a ou 2b)
$GO_FIELDS['plugin_genericobject_classeces_id']['name']       = $LANG['genericobject
↪']['PluginGenericobjectBiomedical'][3];
$GO_FIELDS['plugin_genericobject_classeces_id']['field']      = 'classce';
$GO_FIELDS['plugin_genericobject_classeces_id']['input_type'] = 'dropdown';

// UF (Unité Fonctionnelle)
$GO_FIELDS['plugin_genericobject_ufs_id']['name']       = $LANG['genericobject']['
↪'PluginGenericobjectBiomedical'][4];
$GO_FIELDS['plugin_genericobject_ufs_id']['field']       = 'uf';
$GO_FIELDS['plugin_genericobject_ufs_id']['input_type'] = 'dropdown';

// PRESTATAIRE BIOMED
$GO_FIELDS['plugin_genericobject_prestataires_id']['name']       = $LANG[
↪'genericobject']['PluginGenericobjectBiomedical'][5];
$GO_FIELDS['plugin_genericobject_prestataires_id']['field']       = 'prestataire␣
↪biomed';
$GO_FIELDS['plugin_genericobject_prestataires_id']['input_type'] = 'dropdown';

// TYPE D'EQUIPEMENT BIOMED
$GO_FIELDS['plugin_genericobject_typedequipementbiomeds_id']['name']       = $LANG[
↪'genericobject']['PluginGenericobjectBiomedical'][6];
$GO_FIELDS['plugin_genericobject_typedequipementbiomeds_id']['field']       = "type d
↪'equipement biomed";
$GO_FIELDS['plugin_genericobject_typedequipementbiomeds_id']['input_type'] = 'dropdown
↪';

// Criticite
$GO_FIELDS['plugin_genericobject_criticites_id']['name']       = $LANG['genericobject
↪']['PluginGenericobjectBiomedical'][7];
$GO_FIELDS['plugin_genericobject_criticites_id']['field']      = 'criticite';
$GO_FIELDS['plugin_genericobject_criticites_id']['input_type'] = 'dropdown';

// Numéro marquage CE
$GO_FIELDS['plugin_genericobject_marquageces_id']['name']       = $LANG['genericobject
↪']['PluginGenericobjectBiomedical'][8];
```

```
$GO_FIELDS['plugin_genericobject_marquageces_id']['field']      = 'marquagece';
$GO_FIELDS['plugin_genericobject_marquageces_id']['input_type'] = 'dropdown';

// Classe électrique
$GO_FIELDS['plugin_genericobject_classeelecs_id']['name']       = $LANG['genericobject
↪']['PluginGenericobjectBiomedical'][9];
$GO_FIELDS['plugin_genericobject_classeelecs_id']['field']      = 'classeelec';
$GO_FIELDS['plugin_genericobject_classeelecs_id']['input_type'] = 'dropdown';
?>
```

Warning: Trailing `s_id` is mandatory in `[plugin_genericobject_field*s_id*]` because the GLPI framework requires foreign key fields to end with `s_id`. In database, `glpi_plugin_genericobject_fields` is table name and `id`, its foreign key. See GLPI developer documentation.

Define fields labels

See *Edit labels* section.

- Edit your locales file, for example : `<GLPI_ROOT>/files/_plugins/genericobject/locales/biomedical/biomedical.fr_FR.php`
- Add following content at the end of file :

```
<?php
// Fields
$LANG['genericobject']['PluginGenericobjectBiomedical'][1]="Code CNEH";
$LANG['genericobject']['PluginGenericobjectBiomedical'][2]="Réformé";
$LANG['genericobject']['PluginGenericobjectBiomedical'][3]="Classe CE";
$LANG['genericobject']['PluginGenericobjectBiomedical'][4]="UF";
$LANG['genericobject']['PluginGenericobjectBiomedical'][5]="Prestataire Biomed";
$LANG['genericobject']['PluginGenericobjectBiomedical'][6]="Type d'équipement biomed";
$LANG['genericobject']['PluginGenericobjectBiomedical'][7]="Criticité";
$LANG['genericobject']['PluginGenericobjectBiomedical'][8]="Marquage CE";
$LANG['genericobject']['PluginGenericobjectBiomedical'][9]="Classe électrique";
```

Define behaviours

In *Plugins* > *Objects management* menu, on *Main* tab, select :

- *Item in the dustbin*
- *Historical*
- *Financial and administratives information*
- *Documents*
- *Global search*
- *Assistance*
- *Templates*
- *Contracts*
- *Global search*

This will add ready to use fields to your type of object.

Add fields to your type of object

In *Plugins* > *Objects management* menu, on *Fields* tab, you can now add fields to Biomedical type of object :

- ready to use fields (GLPI's built-in fields)
- new fields (defined in *Define Biomedical's new fields* section)

4.9.5 GLPI configuration

Define Admin_biomed profile

1. Clone *Admin* profile
2. Set following rights in *Admin_biomed* profile :
   - *Administration* > *Profiles* > *Admin_biomed* > *Assets tab* > *Unselect all*
   - *Administration* > *Profiles* > *Admin_biomed* > *Assistance tab* > *Association* > *Associable items to a ticket* > *Biomedical*
   - *Administration* > *Profiles* > *Admin_biomed* > *Management tab* > *Select all*
   - *Administration* > *Profiles* > *Admin_biomed* > *Objects management tab* > *Biomedical* > *Select all*

Note: With these settings, *Admin_biomed* users only see *Biomedical* in Assets menu.

Define Biomed entity and authorizations rules

1. Create *Biomed* entity under *Root entity* in *Administration* > *Entities*
2. Configure authorizations rules to assign *Admin_biomed* profile to *Biomed* entity users.

4.9.6 Start using Generic Object and GLPI

As *Admin_biomed* user, you can create your first object in *Assets* > *Biomedical*.

In order to gain time, define values in *Setup* > *Dropdowns* > *Objects management* for new fields.

CHAPTER 5

Fields

- Sources link: https://github.com/pluginsGLPI/fields
- Download: https://github.com/pluginsGLPI/fields/releases

Fields plugin allows to add additional fields on various items GLPI manages: tickets, computers, users, ...

5.1 Requirements

This plugin requires :

- GLPI 0.85 or higher,
- PHP 5.4 or higher when used with GLPI 9.1, and PHP 5.5 or higher when used with a previous GLPI release.

5.2 Features

- Add containers block on various items,
- Add fields into blocks,
- Display blocks in an existing tab or in their own tab,
- Many fields types available,
- Fields can be flagged as required,
- Manage profiles ACLs per container,
- ...

5.3 Install the Plugin

- Uncompress the archive.
- Move the `fields` directory to the `<GLPI_ROOT>/plugins` directory
- Navigate to the *Configuration* > *Plugins* page,
- Install and activate the plugin.

5.4 Usage

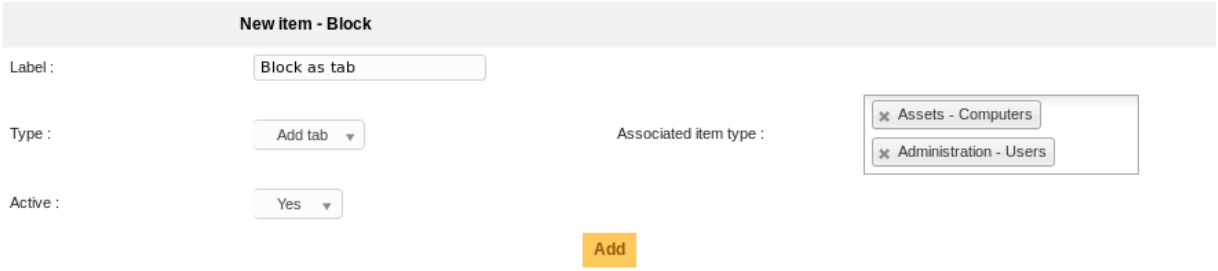
The plugin will create required tables in the database and some files on disk (into `<GLPI_ROOT>/files/_plugins/fields`) automatically. Those files and tables will be updated along with the plugin.

Plugin usage is quite simple:

- create a block linked with some items,
- create fields in this block.

You will access the plugin configuration from the *Setup* > *Aditionnal fields* menu.

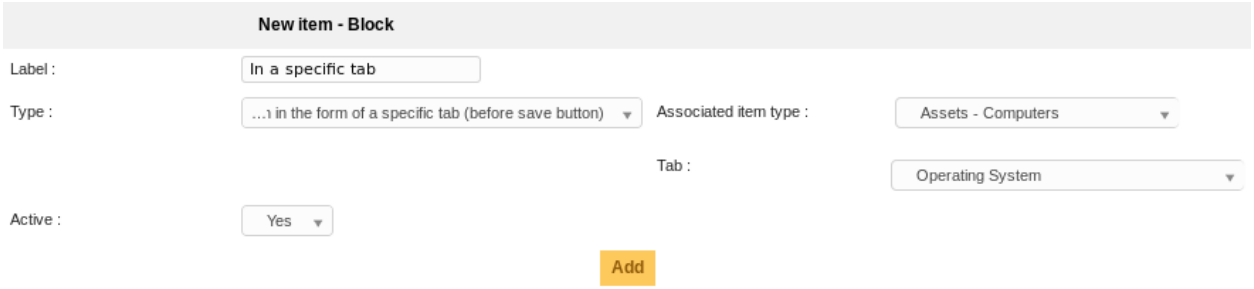
5.4.1 Adding a new block



- *Label*: the label of the block that will appears on items forms,
- *Type*: type of display, one of:
  - *Add tab*: will add a new tab on the item form containing the block,
  - *Insertion in the form*: will add the block at the end of the main item form,
  - *Insertion in the form of a specific tab*: will add the block at the end of a specific tab of the item.
- *Associated item type*: a list of items types on which the block will be added,
- *Active*: whether the block is active or not.

Note: You can add only one block of the type *Insertion in the form* for an associated item type.

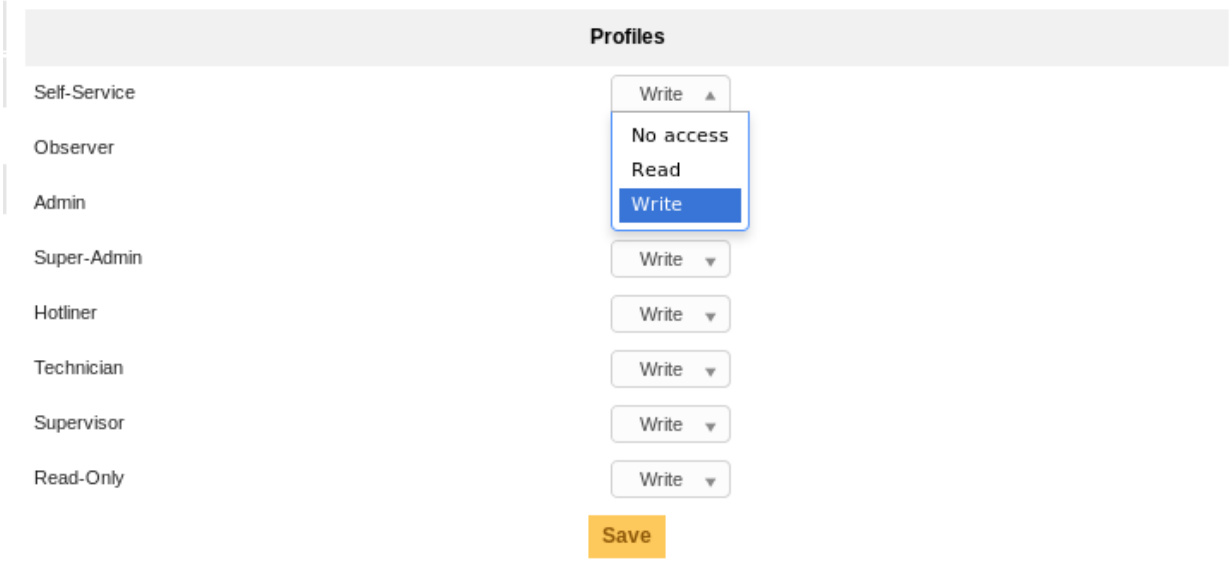
Inserting a block in a specific tab is only possible for one item:



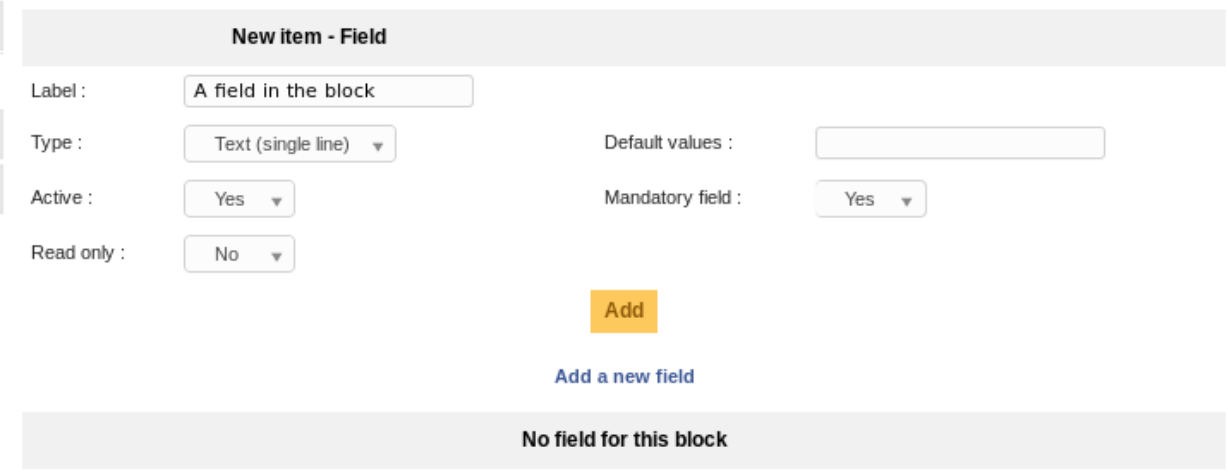
Once a block have been created, it is possible to change its label or is active flag, but not its type or associated items list.

Profiles

You can refine profiles access on the block using the *Profiles* tab from the block. You will be able to choose either *No access*, *Read* or *Write* for every existing profile.



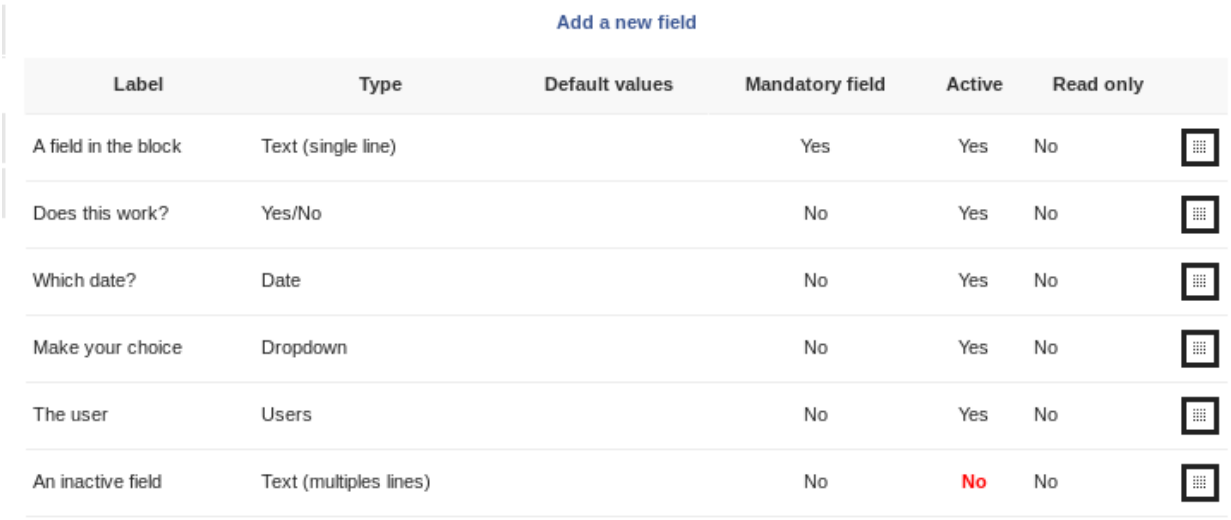
5.4.2 Adding a new field



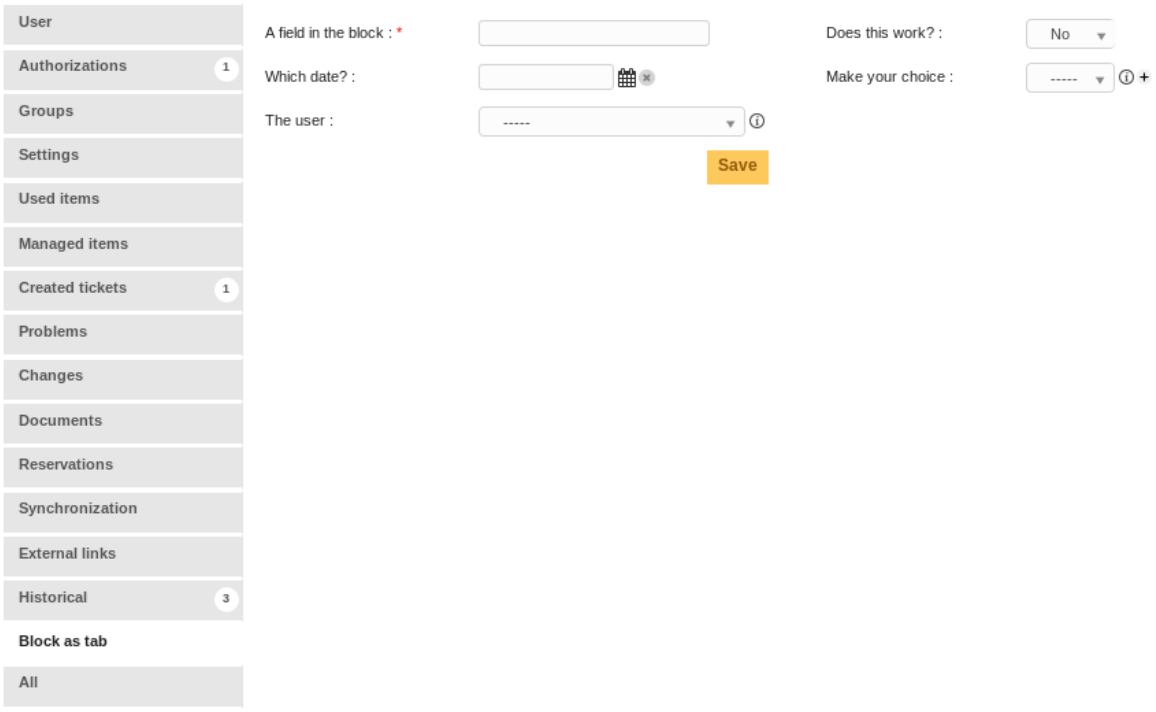
- *Label*: the label of the block that will appears on items forms,
- *Type* field type, one of:
  - *Header*: an header label, for visual distinction,
  - *Text (single line)*: a single line (`input/@type="text"`) of text,
  - *Text (multiples lines)*: a multiline (`textarea`) text,
  - *Number*: a single number (no text allowed),
  - *Dropdown*: a configurable dropdown list, values are configured from the main GLPI dropdown configuration you can find in *Setup* > *Dropdowns* menu,
  - *Yes/No*: a dropdown list with only *Yes* and *No* values,
  - *Date*: a standalone date with a date picker,
  - *Date & time*: a date field with a date picker and a time dropdown field (step is configured from GLPI core),
  - *Users*: list of users.
- *Active*: whether the block is active or not,
- *Read only*: flag this field as read only,
- *Default values*: default field values,
- *Mandatory field*: flag this field as mandatory.

> **Warning:** Of course, you will need to pay attention playing with the various options... For example, if you flag a field as mandatory and as read only, and if you do not provide a default value, form validation will fail.
>
> Another example: you should not set a text default value for a number field... That kind of things ;)

Once fields have been created, you can reorder or edit them from the Fields tab of the block:



And see what it should look like on a *User* item for example:



5.5 Search

All fields that have been added are available in the attached items search forms.

5.6 Simplified Interface

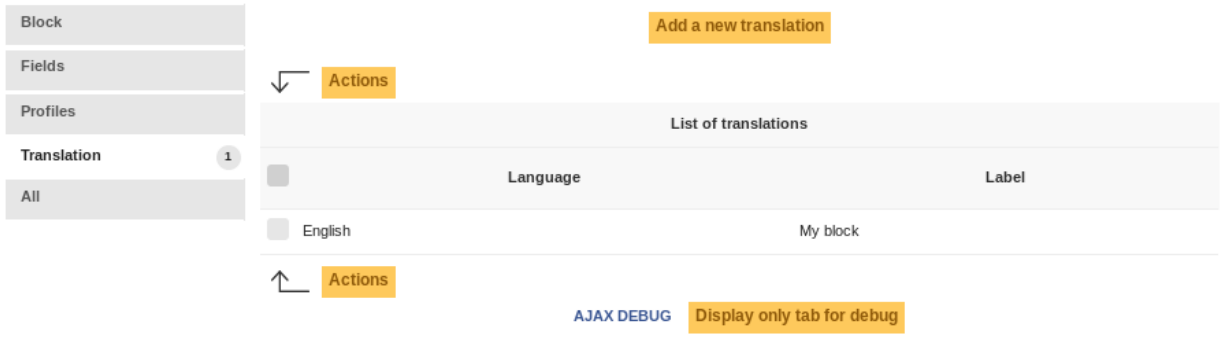
Only blocks attached to tickets and with the type *Insertion in the form* will be displayed in the simplified interface of ticket creation. Of course, it will also take care of current profile rights!

5.7 Translations

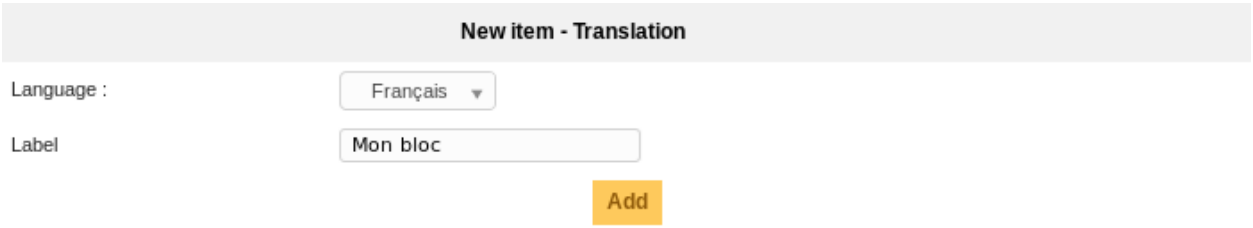
New in version 1.4.0.

Plugin itself is translated using the transifex services; but you aslo can translate the label values for blocs and fields; the process is identical on both those ones:

On the block form, choose the *Translations* tab:



On block creation, a new translation with current language and value set as label will be created; you can add as many translations as you want.



Dropdown values can be translated using the core's dropdown configuration.

Warning: As of GLPI 9.1.1; some issues on the core side prevent dropdown plugin values to be translated; the possibility has been deactivated in the plugin. It will be enabled again once the core has been fixed.

5.8 Entities

If you are using entities in your GLPI instance, a block will be displayed only if the entity it is attached to is compatible with the one of the item it applies to.

When you create or edit a block in multi entities mode, you'll have to choose if the block will be available in children entities:

As an example, let's say our entities have the following structure:

- root
  - A
    - C
  - B

The following table will tell you in wich cases the block is displayed or not:

Block entity	Recursive block	Item entity	Block displayed
A	Yes or No	A	Yes
A	Yes or No	B	No
A	Yes	C	Yes
A	No	C	No

CHAPTER 6

Form creator

- Sources link: https://github.com/pluginsGLPI/formcreator
- Download: https://github.com/pluginsGLPI/formcreator/releases

FormCreator is a plugin which allow creation of custom forms of easy access.

At the same time, the plugin allow the creation of one or more tickets when the form is filled.

6.1 Features

- Direct access to forms self-service interface in main menu,
- Highlighting forms in homepages,
- Access to forms controlled: public access, identified user access, restricted access to some profiles,
- Simple and customizable forms,
- Forms organized by categories, entities and languages,
- Questions of any type of presentation: Textareas, lists, LDAP, files, etc,
- Questions organised in sections. Choice of the display order,
- Possibility to display a question based on certain criteria (response to a further question),
- A sharp control on responses from forms: text, numbers, size of fields, email, mandatory fields, regular expressions, etc,
- Creation of one or more tickets from form answers,
- Adding a description per fields, per sections, per forms, entities or languages,
- Formatting the ticket set: answers to questions displayed, tickets templates,
- Preview form created directly in the configuration.

You can take an look to the full changelog from plugin sources.

6.2 Install the Plugin

- Uncompress the archive.
- Move the `formcreator` directory to the `<GLPI_ROOT>/plugins` directory
- Navigate to the *Configuration* > *Plugins* page,
- Install and activate the plugin.

Warning: The plugin's directory must have the same name as the plugin:

- **Good**: *glpi/plugins/formcreator*
- **Bad**: *glpi/plugins/formcreator-master*
- **Bad**: *glpi/plugins/formcreator-0.90-1.3.2*

Only one directory must contains the plugin's files of a single plugin in the GLPI plugins directory. **Don't rename the plugin's directory for backup, move it!**

6.3 Configuration and usage

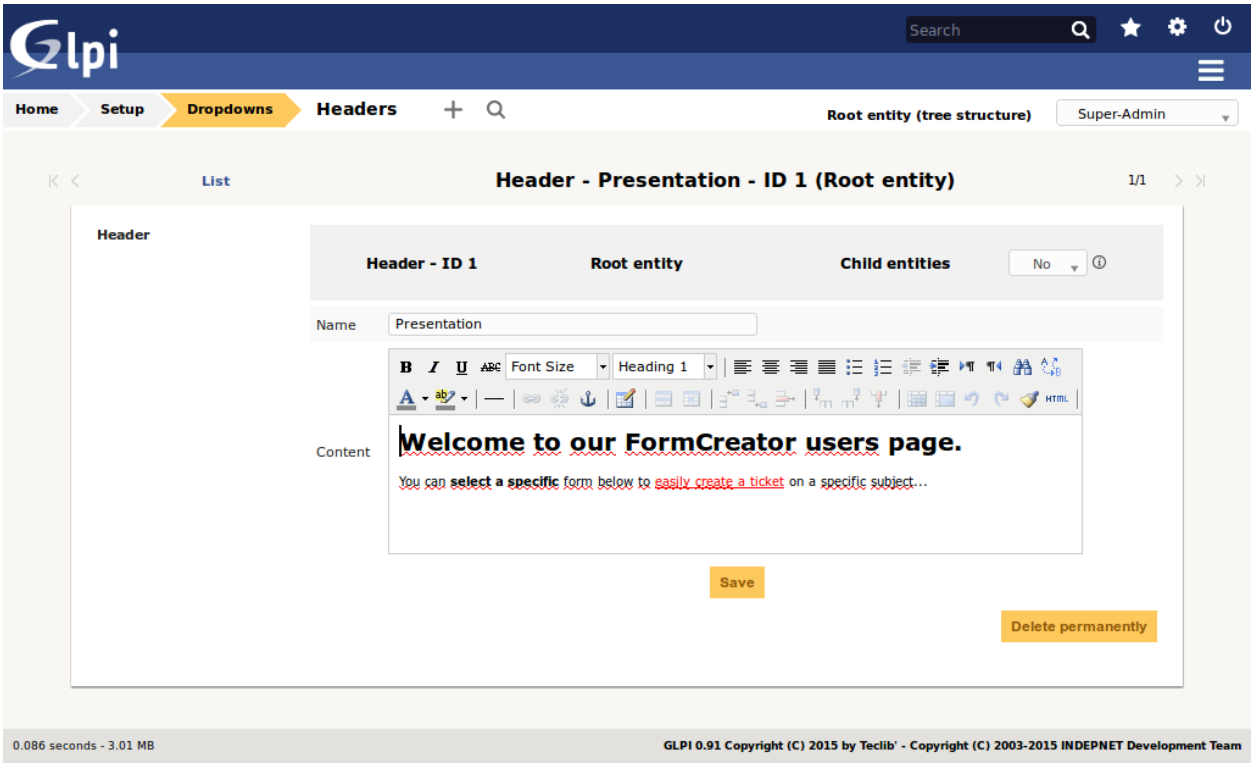
6.3.1 Headers

Menu : *Setup* > *Dropdowns : Forms* > *Headers*

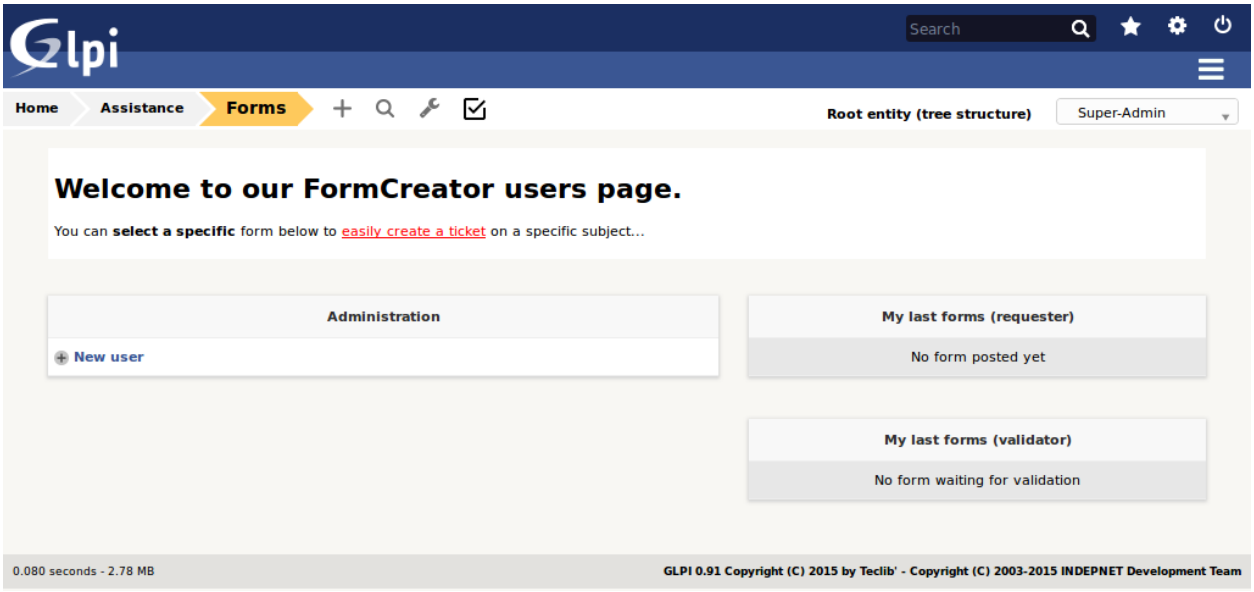
Headers are rich texts _(text with presentation: bold, italic, titles, colors, etc.)_ which are defined per entities. They can be translated since GLPI 0.85 like all other dropdowns _(Dropdowns translation must be enabled on GLPI general configuration page: Setup > General : General setup > Translate dropdowns = Yes)_.

These texts are shown on the users forms list, over the forms. It can be used to indicate some general advertisement over all the forms below.

Configuration



Render



6.3.2 Categories

Menu : *Setup > Dropdowns : Forms > Form categories*

Form categories allow you to arrange your forms list.

You can add or edit categories generally from the Setup menu : *Setup* > *Dropdowns*.

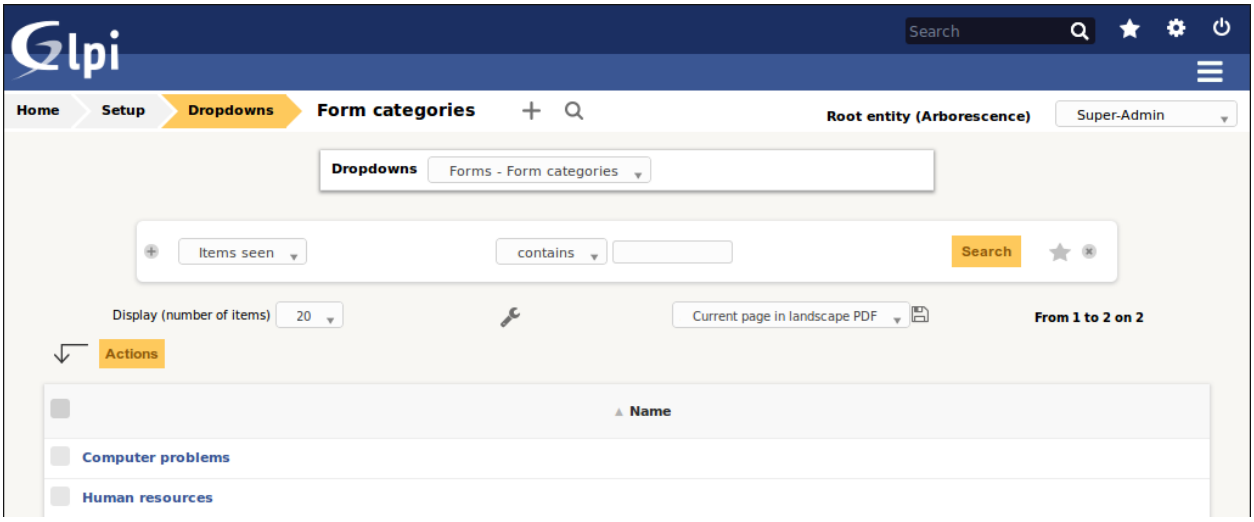
You can also add new categories directly from the form page like all GLPI dropdowns.

They are defined by entities and can be translated since GLPI 0.85 like all other dropdowns.

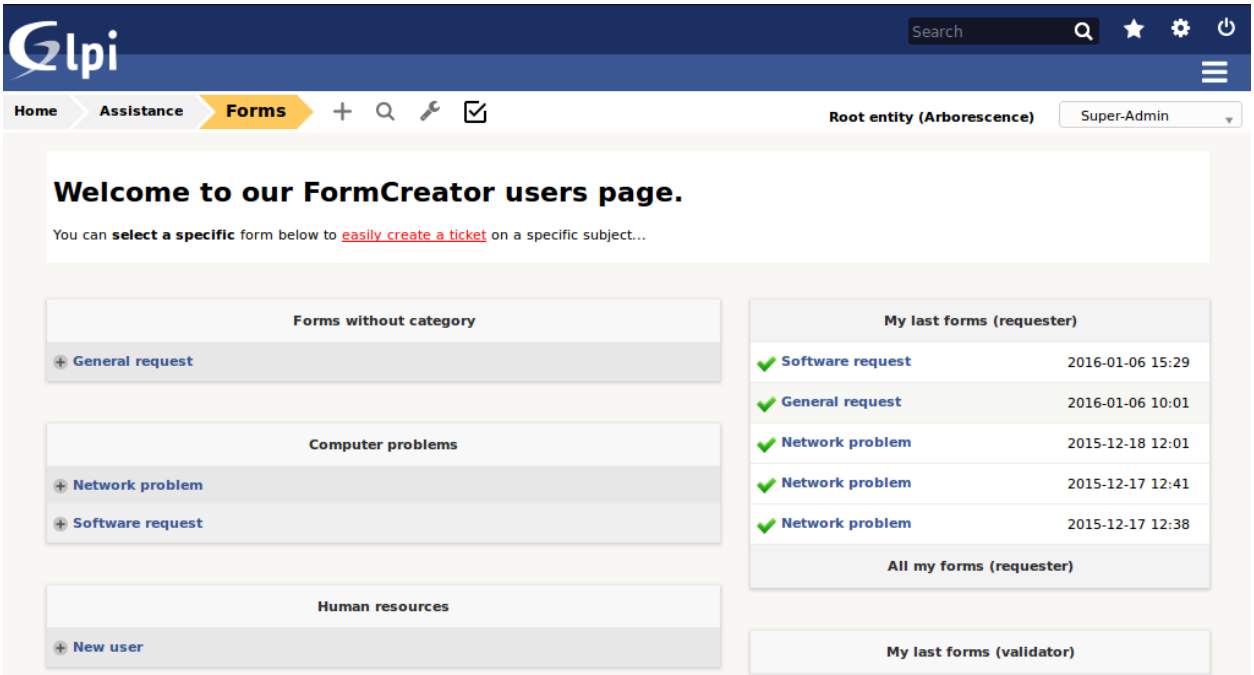
Note: Dropdowns translation must be enabled on GLPI general configuration page *Setup* > *General*, *General setup* > *Translate dropdowns* = *Yes*

Note: Categories may be associated to *Knowledge base categories*. This link is necessary to allow FAQ entries to show along your forms.

Configuration

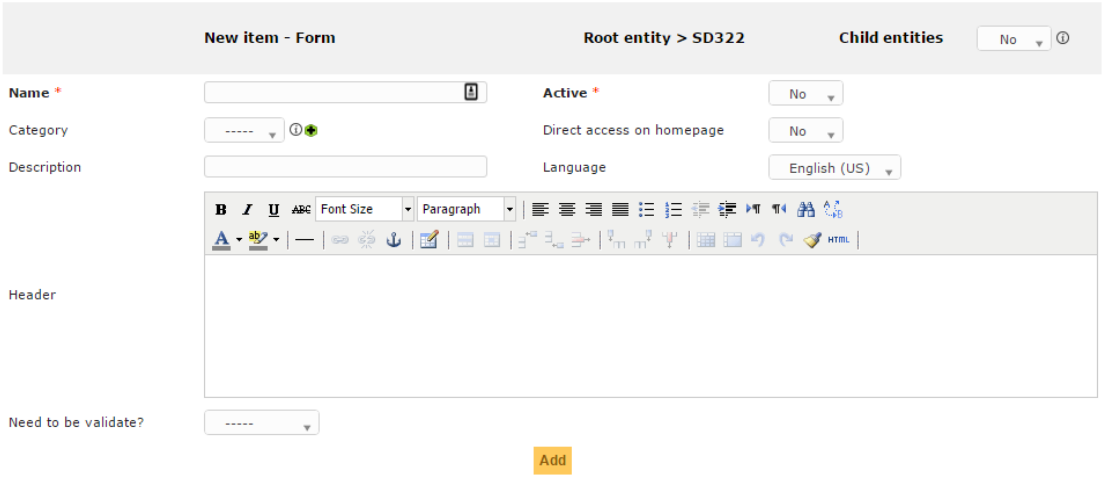


Render



6.3.3 Questions

After the creation of a form, create fields for for the user to fill out.



The name of the questions will appear on the left and the field type selected on the right

The Description will be under the filed input.

Based on question type options will be based on type chosen.

If validation is desired, it can be implemented following PHP Regular Expressions.

If you want to show or hide questions depending on the answers of other questions, use the *show fields* area when editing a question. In the version 2.5.0 you may use more complex expressions checking for the content of several

questions, and use logic operator **OR** and **AND**. The precedence of boolean operators applies, meaning that **AND** has precedence over **OR**.

6.3.4 Helpdesk

The plugin can provide its own design for helpdesk.

To enable it, edit an entity, open the *Forms* tab and set the field *Helpdesk mode* to *Service catalog simplified* or *Service catalog extended*. This setting handles inheritance from parent entity to children entities.

Users using the simplified interface will benefit a new interface allowing them to:

- browse forms and FAQ with the unified interface
- follow the process of their requests
- book assets
- view their feeds

Users using the extended interface have a more complete view on their requests.

Forms with *Direct access on homepage* enabled and *FAQ items* in the *Knowledge base* will will appear in the interface. Users may search by browsing the categories on the left of the screen, and may also search for forms or FAQ with a natural language search engine.

FAQ items needs to be associated to knowledge base categories. The knowledge base categories must be associated to form categories (available in *Setup > Dropdowns : Forms > Form categories*) to show their content.

6.4 Contributing

6.4.1 Adding new features

You are a developer ?

You can help us adding new features by forking the GitHub repository.

Develop the new feature on your branch and then, ask for a pull request to merge your development into the repository.

You can also help us debug the declared issues and requested features.

You are not a developer ?

You can submit your ideas of new feature by creating an issue or adding comment on an existing issue (to add explanations or just to say "I'm also interested").

See also the roadmap

6.4.2 Help us debug

If you are a developer and want to code the fix yourself, just read the paragraph bellow...

But if you are not a developer or don't want to develop fixes yourself, you can still help us by creating issues. Indicate your GLPI and plugin's version and steps to reproduce for a faster and easier fix.

6.4.3 Documentation

You can fork the documentation repository to add new contents or fix some issues in existing content.

Make your changes on your branch and then, ask for a pull request) to merge it into the repository.

6.4.4 Translations

If you want Formcreator to be available in your native language and have a little time, you can help us:

Join us on Transifex.

Current available languages : Czech (Czech Republic), English (United Kingdom) , French (France), German (Germany), Hungarian (Hungary), Polish (Poland), Portuguese (Brazil), Romanian (Romania), Russian (Russia), Spanish (Argentina), Turkish (Turkey)

6.5 Team

6.5.1 Developers

- Jérémy MOREAU,
- Alexandre Delaunay,
- François Legastelois,
- dethegeek

6.5.2 Testers

- Manu1400,
- dgsn,
- Max6Devil,
- Kahalisto,
- satyan01,
- biet-j,
- reg25,
- Gilmor49,
- nblaisonneau,
- CHCI74,
- salgueirosa,
- jotafe,
- Ruiseart,
- babalou,

- blienard,
- proprux,
- ThedarksideoftheForce,
- AdAugustaPerAngusta,
- MaxG89,
- klodnitsky,
- consolko,
- Boris31,
- fire2418,
- osfrance,
- kaioisdead,
- KevinSupertramp,
- matth974,
- J-n-s,
- wawax,
- Sismic,
- nicholaseduardo
- and much more…

6.5.3 Translators

French (France)

- Jérémy MOREAU,
- Alexandre DELAUNAY,
- Walid Nouh,
- Thierry Bugier Pineau,
- Benoit Le Rohellec,
- Emmanuel Haguet

Turkish (Turkey)

- Kaya Zeren

English (United Kingdom)

- Jérémy MOREAU,
- Andrejs Klodnickis

Polish (Poland)

- Sismic,
- Ryszard Jeziorski,
- awiamo,
- Grzegorz Kaniewski

Russian (Russia)

- Alexey Petukhov,
- Igor Nikitin,
- Andrejs Klodnickis

Portuguese (Brazil)

- Pedro de Oliveira Lira,
- Rafael Viana

Hungarian (Hungary)

- Laszlo Czirbesz]

Spanish (Argentina)

- Luis A. Uriarte

Romanian (Romania)

- Doru DEACONU

Czech (Czech Republic)

- David Stepan