
GLPI Developer Documentation Documentation

Release 9.2

Teclib'

Jan 25, 2018

Contents

1	Source Code management	3
1.1	Branches_test	3
1.2	Testing	3
1.3	File Hierarchy System	4
1.4	Workflow	5
2	Coding standards	9
2.1	Indentation	9
2.2	Spacing	9
2.3	Control structures	9
2.4	Including files	10
2.5	PHP tags	10
2.6	Functions	10
2.7	Classes	11
2.8	Variables and Constants	12
2.9	Comments	12
2.10	Variables types	14
2.11	Quotes / double quotes	15
2.12	Files	16
2.13	Database queries	16
2.14	Checking standards	16
3	Developer API	19
3.1	Main framework objects	19
3.2	Querying database	23
3.3	DBIterator	24
3.4	Database model	27
3.5	Translations	29
3.6	Tools	31
4	shenzhendoit.am	35
4.1	Review process	35
4.2	Prepare next major release	36
5	Plugins	37
5.1	Requirements	37
5.2	Guidelines	40

5.3	Hooks	41
6	Packaging	49
6.1	Sources	49
6.2	Filesystem Hierarchy Standard	49
6.3	Apache Configuration File	50
6.4	Logs files rotation	52
6.5	SELinux stuff	52
6.6	Use system cron	52
6.7	Using system libraries	53
6.8	Using system fonts rather than bundled ones	54
6.9	Notes	54



Source Code management

GLPI source code management is handled by [GIT](#) and hosted on [GitHub](#).

In order to contribute to the source code, you will have to know a few things about Git and the development model we follow.

1.1 Branches_test

On the Git repository, you will find several existing branches:

- *master* contains the next major release source code,
- *xx/bugfixes* contains the next minor release source code,
- you should not care about all other branches that may exists, they should have been deleted right now.
- *xx/bugfixes* contains the next minor release source code doit
- SZDoit doit

The *master* branch is where new features are added. This code is reputed as **non stable**.

The *xx/bugfixes* branches is where bugs are fixed. This code is reputed as *stable*.

Those branches are created when any major version is released. At the time I wrote these lines, latest stable version is *9.1* so the current bugfix branch is *9.1/bugfixes*. As old versions are not supported, old bugfixes branches will not be changed at all; while they're still existing.

1.2 Testing

Unfortunately, tests in GLPI are not numerous... But that's willing to change, and we'd like - as far as possible - that proposals contains unit tests for the bug/feature they're related.

Anyways, existing unit tests may never be broken, if you made a change that breaks something, check your code, or change the unit tests, but fix that! ;)

1.3 File Hierarchy System

Note: This lists current files and directories listed in the source code of GLPI. Some files are not part of distributed archives.

This is a brief description of GLPI main folders and files:

-  *.tx*: Transifex configuration
-  *ajax*
 -  **.php*: Ajax components
-  *files* Files written by GLPI or plugins (documents, session files, log files, ...)
-  *front*
 -  **.php*: Front components (all displayed pages)
-  *config* (only populated once installed)
 -  *config_db.php*: Database configuration file
 -  *local_define.php*: Optional file to override some constants definitions (see *inc/define.php*)
-  *css*
 -  *...*: CSS stylesheets
 -  **.css*: CSS stylesheets
-  *inc*
 -  **.php*: Classes, functions and definitions
-  *install*
 -  *mysql*: MariaDB/MySQL schemas
 -  **.php*: upgrades scripts and installer
-  *js*
 -  **.js*: Javascript files
-  *lib*
 -  *...*: external Javascript libraries
-  *locales*
 -  *glpi.pot*: Gettext's POT file
 -  **.po*: Gettext's translations
 -  **.mo*: Gettext's compiled translations
-  *pics*
 -  **.**: pictures and icons

-  *plugins*:
 -  *...*: where all plugins lands
-  *scripts*: various scripts which can be used in crontabs for example
-  *tests*: unit and integration tests
-  *tools*: a bunch of tools
-  *vendor*: third party libs installed from composer (see `composer.json` below)
-  *.gitignore*: Git ignore list
-  *.htaccess*: Some convenient apache rules (all are commented)
-  *.travis.yml*: Travis-CI configuration file
-  *apirest.php*: REST API main entry point
-  *apirest.md*: REST API documentation
-  *apixmlrpc.php*: XMLRPC API main entry point
-  *AUTHORS.txt*: list of GLPI authors
-  *CHANGELOG.md*: Changes
-  *composer.json*: Definition of third party libraries (see [composer website](#))
-  *COPYING.txt*: Licence
-  *index.php*: main application entry point
-  *phpunit.xml.dist*: unit testing configuration file
-  *README.md*: well... a README ;)
-  *status.php*: get GLPI status for monitoring purposes

1.4 Workflow

1.4.1 In short...

In a short form, here is the workflow we'll follow:

- [create a ticket](#)
- fork, create a specific branch, and hack
- open a PR (Pull Request)

Each bug will be fixed in a branch that came from the correct *bugfixes* branch. Once merged into the requested branch, developer must report the fixes in the *master*; with a simple cherry-pick for simple cases, or opening another pull request if changes are huge.

Each feature will be hacked in a branch that came from *master*, and will be merged back to *master*.

1.4.2 General

Most of the times, when you'll want to contribute to the project, you'll have to retrieve the code and change it before you can report upstream. Note that I will detail here the basic command line instructions to get things working; but of course, you'll find equivalents in your favorite Git GUI/tool/whatever ;)

Just work with a:

```
$ git clone https://github.com/glpi-project/glpi.git
```

A directory named `glpi` will be created where you've issued the clone.

Then - if you did not already - you will have to create a fork of the repository on your github account; using the *Fork* button from the [GLPI's Github page](#). This will take a few moments, and you will have a repository created, *{your user name}/glpi - forked from glpi-project/glpi*.

Add your fork as a remote from the cloned directory:

```
$ git remote add my_fork https://github.com/{your user name}/glpi.git
```

You can replace *my_fork* with what you want but *origin* (just remember it); and you will find your fork URL from the Github UI.

A basic good practice using Git is to create a branch for everything you want to do; we'll talk about that in the sections below. Just keep in mind that you will publish your branches on your fork, so you can propose your changes.

When you open a new pull request, it will be reviewed by one or more member of the community. If you're asked to make some changes, just commit again on your local branch, push it, and you're done; the pull request will be automatically updated.

1.4.3 Bugs

If you find a bug in the current stable release, you'll have to work on the *bugfixes* branch; and, as we've said already, create a specific branch to work on. You may name your branch explicitly like *9.1/fix-thing* or to reference an existing issue *9.1/fix-1234*; just prefix it with *{version}/fix-*.

Generally, the very first step for a bug is to be [filled in a ticket](#).

From the clone directory:

```
$ git checkout -b 9.1/bugfixes origin/9.1/bugfixes
$ git branch 9.1/fix-bad-api-callback
$ git co 9.1/fix-bad-api-callback
```

At this point, you're working on an only local branch named *9.1/fix-api-callback*. You can now work to solve the issue, and commit (as frequently as you want).

At the end, you will want to get your changes back to the project. So, just push the branch to your fork remote:

```
$ git push -u my_fork 9.1/fix-api-callback
```

Last step is to create a PR to get your changes back to the project. You'll find the button to do this visiting your fork or even main project github page.

Just remember here we're working on some bugfix, that should reach the *bugfixes* branch; the PR creation will probably propose you to merge against the *master* branch; and maybe will tell you they are conflicts, or many commits you do not know about... Just set the base branch to the correct bugfixes and that should be good.

1.4.4 Features

Before doing any work on any feature, make sure it has been discussed by the community. Open - if it does not exist yet - a ticket with your detailed proposition. For technical features, you can work directly on github; but for work proposals, you should take a look at our [feature proposal platform](#).

If you want to add a new feature, you will have to work on the *master* branch, and create a local branch with the name you want, prefixed with *feature/*.

From the clone directory:

```
$ git branch feature/my-killer-feature
$ git co feature/my-killer feature
```

You'll notice we do not change branch on the first step; that is just because *master* is the default branch, and therefore the one you'll be set on just after cloning. At this point, you're working on an only local branch named *feature/my-killer-feature*. You can now work and commit (as frequently as you want).

At the end, you will want to get your changes back to the project. So, just push the branch on your fork remote:

```
$ git push -u my_fork feature/my-killer-feature
```

1.4.5 Commit messages

There are several good practices regarding commit messages, but this is quite simple:

- the commit message may refer an existing ticket if any,
 - just make a simple reference to a ticket with keywords like `refs #1234` or `see #1234`,
 - automatically close a ticket when commit will be merged back with keywords like `closes #1234` or `fixes #1234`,
- the first line of the commit should be as short and as concise as possible
- if you want or have to provide details, let a blank line after the first commit line, and go on. Please avoid very long lines (some conventions talk about 80 characters maximum per line, to keep it readable).

1.4.6 Third party libraries

Third party libraries are handled using the [composer tool](#).

To install existing dependencies, just install composer from their website or from your distribution repositories and then run:

```
$ composer install
```

To add a new library, you will probably find the command line on the library documentation, something like:

```
$ composer require libauthor/libname
```



2.1 Indentation

- 3 spaces
- Max line width: 100

```
<?php
// base level
    // level 1
        // level 2
            // level 1
// base level
```

2.2 Spacing

We've adopted "french spacing" rules in the code. The rule is:

- for *simple* punctuation (, , .): use *one space after* the punctuation sign
- for *double* punctuation (! , ? , :): use *one space after and one space before* the punctuation sign
- for *opening* punctuation ((, { , [): use *one space before* the punctuation sign
- for *closing* punctuation () , } ,]): use *one space after* the punctuation sign, excepted for line end, when followed by a semi-colon (;)

Of course, this rules only applies on the source code, not on the strings (translatable strings, comments, ...)!

2.3 Control structures

Multiple conditions in several indented lines

```
<?php
if ($test1) {
    for ($i=0 ; $i<$end ; $i++) {
        echo "test ".( $i<10 ? "0$i" : $i )."<br>";
    }
}

if ($a==$b
    || ($c==$d && $e==$f)) {
    ...
} else if {
    ...
}

switch ($test2) {
    case 1 :
        echo "Case 1";
        break;

    case 2 :
        echo "Case 2";
        // No break here : because...

    default :
        echo "Default Case";
}
}
```

2.4 Including files

Use `include_once` in order to include the file once and to raise warning if file does not exists:

```
include_once GLPI_ROOT."/inc/includes.php";
```

2.5 PHP tags

Short tag (`<?`) is not allowed; use complete tags (`<?php`).

```
<?php
// code
```

The PHP closing tag `?>` must be avoided on full PHP files (so in most of GLPI's files!).

2.6 Functions

Function names must be written in *camelCaps*:

```
<?php
function userName($a, $b) {
    //do something here!
}
```

If parameters add block doc for these parameters, please see the *Comments* section for any example.

If function from parent add

```
<?php
function getMenuContent ()
```

If it's a new function, add in block doc (see the *Comments* section):

```
@since version 9.1
```

2.6.1 Call static methods

Function location	How to call
class itself	<code>self::theMethod()</code>
parent class	<code>parent::theMethod()</code>
another class	<code>ClassName::theMethod()</code>

2.7 Classes

Class names must be written in *CamelCase*:

GLPI do not use [PHP namespaces](#) right now; so be carefull when creating new classes to take a name that does not exists yet.

```
<?php
class MyExampleClass extends AnotherClass {
    // do something
}
```

Note: even if GLPI does not use namespaces, some libs does, you will have to take care of that. You can also if you wish use namespaces for PHP objects call.

For example, the folloing code:

```
<?php
try {
    ...
    $something = new stdClass();
    ...
} catch (Exception $e{
    ...
}
```

Could also be written as (see the `\`):

```
<?php
try {
    ...
    $something = new \stdClass();
    ...
} catch (\Exception $e{
    ...
}
```

2.8 Variables and Constants

- Variable names must be as descriptive and as short as possible, stay clear and concise.
- In case of multiple words, use the `_` separator,
- Variables must be **lower case**,
- Global variables and constants must be **UPPER case**.

```
<?php
$user          = 'glpi';
// put elements in alphabetic order
$users        = array('glpi', 'glpi2', 'glpi3');
$users        = array('glpi1' => 'valeur1',
                      'nexglpi' => array('down' => '1',
                                         'up'   => array('firstfield' => 'newvalue')),
                      'glpi2'  => 'valeur2');
$users_groups = array('glpi', 'glpi2', 'glpi3');

$CFG_GLPI = array();
```

2.9 Comments

To be more visible, don't put inline block comments into `/* */` but comment each line with `//`. Put docblocks comments into `/** */`.

Each function or method must be documented, as well as all its parameters (see *Variables types* below), and its return.

For each method or function documentation, you'll need at least to have a description, the version it was introduced, the parameters list, the return type; each blocks separated with a blank line. As an example, for a void function:

```
<?php
/**
 * Describe what the method does. Be concise :)
 *
 * You may want to add some more words about what the function
 * does, if needed. This is optionnal, but you can be more
 * descriptive here:
 * - it does something
 * - and also something else
 * - but it doesn't make coffee, unfortunately.
 *
 * @since 9.2
 *
 * @param string $param      A parameter, for something
 * @param boolean $other_param Another parameter
 *
 * @return void
 */
function myMethod($param, $other_param) {
    //[...]
}
```

Some other informations way be added; if the function requires it.

Refer to the [PHPDocumentor website](#) to get more informations on documentation. The latest GLPI API documentation is also available online.

Please follow the order defined below:

1. Description,
2. Long description, if any,
3. *@deprecated*.
4. *@since*,
5. *@var*,
6. *@param*,
7. *@return*,
8. *@see*,
9. *@throw*,
10. *@todo*,

2.9.1 Parameters documentation

Each parameter must be documented in its own line, beginning with the `@param` tag, followed by the *Variables types*, followed by the param name (`$param`), and finally with the description itself. If your parameter can be of different types, you can list them separated with a `|` or you can use the `mixed` type; it's up to you!

All parameters names and description must be aligned vertically on the longest (plu one character); see the above example.

2.9.2 Override method: `@inheritDoc?` `@see?` `docblock?` `no docblock?`

There are many question regarding the way to document a child method in a child class.

Many editors use the `{@inheritDoc}` tag without anything else. **This is wrong**. This *inline* tag is confusing for many users; for more details, see the [PHPDocumentor documentation](#) about it. This tag usage is not forbidden, but make sure to use it properly, or just avoid it. An usage exemple:

```
<?php
abstract class MyClass {
    /**
     * This is the documentation block for the curent method.
     * It does something.
     *
     * @param string $sthing Something to send to the method
     *
     * @return string
     */
    abstract public function myMethod($sthing);
}

class MyChildClass extends MyClass {
    /**
     * {@inheritDoc} Something is done differently for a reason.
     *
     */
}
```

```
* @param string $sthing Something to send to the method
*
* @return string
*/
public function myMethod($sthing) {
    [...]
}
```

Something we can see quite often is just the usage of the `@see` tag to make reference to the parent method. **This is wrong.** The `@see` tag is designed to reference another method that would help to understand this one; not to make a reference to its parent (you can also take a look at [PHPDocumentor documentation about it](#)). While generating, parent class and methods are automatically discovered; a link to the parent will be automatically added. An usage example:

```
<?php
/**
 * Adds something
 *
 * @param string $type Type of thing
 * @param string $value The value
 *
 * @return boolean
 */
public function add($type, $value) {
    // [...]
}

/**
 * Adds myType entry
 *
 * @param string $value The value
 *
 * @return boolean
 * @see add()
 */
public function addMyType($value) {
    return $this->addType('myType', $value);
}
```

Finally, should I add a docblock, or nothing?

PHPDocumentor and various tools will just use parent docblock verbatim if nothing is specified on child methods. So, if the child method acts just as its parent (extending an abstract class, or some super class like `CommonGLPI` or `CommonDBTM`); you may just omit the docblock entirely. The alternative is to copy paste parent docblock entirely; but that way, it would be required to change all children docblocks when parent if changed.

2.10 Variables types

Variables types for use in DocBlocks for Doxygen:

Type	Description
mixed	A variable with undefined (or multiple) type
integer	Integer type variable (whole number)
float	Float type (point number)
boolean	Logical type (true or false)
string	String type (any value in "" or ' ')
array	Array type
object	Object type
ressource	Resource type (as returned from <code>mysql_connect</code> function)

Inserting comment in source code for doxygen. Result : full doc for variables, functions, classes. . .

2.11 Quotes / double quotes

- You must use single quotes for indexes, constants declaration, translations, . . .
- Use double quote in translated strings
- When you have to use tabulation character (`\t`), carriage return (`\n`) and so on, you should use double quotes.
- For performances reasons since PHP7, you may avoid strings concatenation.

Examples:

```
<?php
//for that one, you should use double, but this is at your option...
$a = "foo";

//use double quotes here, for $foo to be interpreted
// => with double quotes, $a will be "Hello bar" if $foo = 'bar'
// => with single quotes, $a will be "Hello $foo"
$a = "Hello $foo";

//use single quotes for array keys
$tab = [
    'lastname' => 'john',
    'firstname' => 'doe'
];

//Do not use concatenation to optimize PHP7
//note that you cannot use functions call in {}
$a = "Hello {$tab['firstname']}";

//single quote translations
$str = __('My string to translate');

//Double quote for special characters
$html = "<p>One paragraph</p>\n<p>Another one</p>";

//single quote cases
switch ($a) {
    case 'foo' : //use single quote here
        ...
    case 'bar' :
        ...
}
```

2.12 Files

- Name in lower case.
- Maximum line length: 100 characters
- Indenttion: 3 spaces

2.13 Database queries

- Queries must be written onto several lines, one statement item by line.
- All SQL words must be **UPPER case**.
- For MySQL, all item based must be slash protected (table name, field name, condition),
- All values from variable, even integer should be single quoted

```
<?php
$query = "SELECT *
        FROM `glpi_computers`
        LEFT JOIN `xyzt` ON (`glpi_computers`.`fk_xyzt` = `xyzt`.`id`
                            AND `xyzt`.`toto` = 'jk')
        WHERE @id@ = '32'
           AND ( `glpi_computers`.`name` LIKE '%toto%'
                OR `glpi_computers`.`name` LIKE '%tata%' )
        ORDER BY `glpi_computers`.`date_mod` ASC
        LIMIT 1";

$query = "INSERT INTO `glpi_alerts`
        (`itemtype`, `items_id`, `type`, `date`) // put field's names to
→avoid mistakes when names of fields change
        VALUE ('contract', '5', '2', NOW())";
```

2.14 Checking standards

In order to check some stabdards are respected, we provide some custom [PHP CodeSniffer](#) rules. From the GLPI directory, just run:

```
phpcs --standard=tools/phpcs-rules.xml inc/ front/ ajax/ tests/
```

If the above command does not provide any output, then, all is OK :)

An example error output would looks like:

```
phpcs --standard=tools/phpcs-rules.xml inc/ front/ ajax/ tests/
FILE: /var/www/webapps/glpi/tests/HtmlTest.php
-----
FOUND 3 ERRORS AFFECTING 3 LINES
-----
 40 | ERROR | [x] Line indented incorrectly; expected 3 spaces, found
```

```
|          |      4
59 | ERROR | [x] Line indented incorrectly; expected 3 spaces, found
|          |      4
64 | ERROR | [x] Line indented incorrectly; expected 3 spaces, found
|          |      4
```



Apart from the current documentation, you can also check the [full PHP documentation of GLPI](#) (generated with `apigen`).

3.1 Main framework objects

GLPI contains numerous classes; but there are a few common objects you'd have to know about. All GLPI classes are in the `inc` directory.

3.1.1 CommonGLPI

This is **the** main GLPI object, most of GLPI or Plugins class inherit from this one, directly or not. The class is in the `inc/commonglpi.class.php` file.

This object will help you to:

- manage item type name,
- manage item tabs,
- manage item menu,
- do some display,
- get URLs (form, search, ...),
- ...

See the [full API documentation for CommonGLPI object](#) for a complete list of methods provided.

3.1.2 CommonDBTM

This is an object to manage any database stuff; it of course inherits from *CommonGLPI*. The class is in the `inc/commondbtm.class.php` file.

It aims to manage database persistence and tables for all objects; and will help you to:

- add, update or delete database rows,
- load a row from the database,
- get table informations (name, indexes, relations, ...)
- ...

The `CommonDBTM` object provides several of the *available hooks*.

See the [full API documentation for CommonDBTM object](#) for a complete list of methods provided.

3.1.3 CommonDropdown

This class aims to manage dropdown (lists) database stuff. It inherits from *CommonDBTM*. The class is in the `inc/commondropdown.class.php` file.

It will help you to:

- manage the list,
- import data,
- ...

See the [full API documentation for CommonDropdown object](#) for a complete list of methods provided.

3.1.4 CommonTreeDropdown

This class aims to manage tree lists database stuff. It inherits from *CommonDropdown*. The class is in the `inc/commonreedropdown.class.php` file.

It will mainly help you to manage the tree aspect of a dropdown (parents, children, and so on).

See the [full API documentation for CommonTreeDropdown object](#) for a complete list of methods provided.

3.1.5 CommonImplicitTreeDropdown

This class manages tree lists that cannot be managed by the user. It inherits from *CommonTreeDropdown*. The class is in the `inc/commonimplicittreedropdown.class.php` file.

See the [full API documentation for CommonTreeDropdown object](#) for a complete list of methods provided.

3.1.6 CommonDBVisible

This class helps with visibility management. It inherits from *CommonDBTM*. The class is in the `inc/commondbvisible.class.php` file.

It provides methods to:

- know if the user can view item,
- get dropdown parameters,
- ...

See the [full API documentation for CommonDBVisible object](#) for a complete list of methods provided.

3.1.7 CommonDBConnexity

This class factorizes database relation and inheritance stuff. It inherits from *CommonDBTM*. The class is in the `inc/commondbconnexity.class.php` file.

It is not designed to be used directly, see *CommonDBChild* and *CommonDBRelation*.

See the [full API documentation for CommonDBConnexity object](#) for a complete list of methods provided.

3.1.8 CommonDBChild

This class manages simple relations. It inherits from *CommonDBConnexity*. The class is in the `inc/commondbchild.class.php` file.

This object will help you to define and manage parent/child relations.

See the [full API documentation for CommonDBChild object](#) for a complete list of methods provided.

3.1.9 CommonDBRelation

This class manages relations. It inherits from *CommonDBConnexity*. The class is in the `inc/commondbrelation.class.php` file.

Unlike *CommonDBChild*; it is designed to declare more *complex relations; as defined in the database model*. This is therefore more complex than just using a simple relation; but it also offers many more possibilities.

In order to setup a complex relation, you'll have to define several properties, such as:

- `$itemtype_1` and `$itemtype_2`; to set both itm types used;
- `$items_id_1` and `$items_id_2`; to set field id name.

Other properties let you configure how to deal with entites inheritance, ACLs; what to log on each part on several actions, and so on.

The object will also help you to:

- get search options and query,
- find rights in ACLs list,
- handle massive actions,
- ...

See the [full API documentation for CommonDBRelation object](#) for a complete list of methods provided.

3.1.10 CommonDevice

This class factorizes common requirements on devices. It inherits from *CommonDropdown*. The class is in the `inc/commondevice.class.php` file.

It will help you to:

- import devices,
- handle menus,
- do some display,
- ...

See the [full API documentation](#) for `CommonDevice` object for a complete list of methods provided.

3.1.11 Common ITIL objects

All common ITIL objects will help you with [ITIL](#) objects management (Tickets, Changes, Problems).

CommonITILObject

Handle ITIL objects. It inherits from *CommonDBTM*. The class is in the `inc/commonitilobject.class.php` file.

It will help you to:

- get users, suppliers, groups, ...
- count them,
- get objects for users, technicians, suppliers, ...
- get status,
- ...

See the [full API documentation](#) for `CommonITILObject` object for a complete list of methods provided.

CommonITILActor

Handle ITIL actors. It inherits from *CommonDBRelation*. The class is in the `inc/commonitilactor.class.php` file.

It will help you to:

- get actors,
- show notifications,
- get ACLs,
- ...

See the [full API documentation](#) for `CommonITILActor` object for a complete list of methods provided.

CommonITILCost

Handle ITIL costs. It inherits from *CommonDBChild*. The class is in the `inc/commonitilcost.class.php` file.

It will help you to:

- get item cost,
- do some display,
- ...

See the [full API documentation](#) for `CommonITILCost` object for a complete list of methods provided.

CommonITILTask

Handle ITIL tasks. It inherits from *CommonDBTM*. The class is in the `inc/commonitiltask.class.php` file.

It will help you to:

- manage tasks ACLs,
- do some display,
- get search options,
- ...

See the [full API documentation](#) for *CommonITILTask* object for a complete list of methods provided.

CommonITILValidation

Handle ITIL validation process. It inherits from *CommonDBChild*. The class is in the `inc/commonitilvalidation.class.php` file.

It will help you to:

- manage ACLs,
- get and set status,
- get counts,
- do some display,
- ...

See the [full API documentation](#) for *CommonITILValidation* object for a complete list of methods provided.



3.2 Querying database

To query database, you can use the `$DB::request()` method and give it a full SQL query.

Warning: Whether this is possible to use full SQL to query database using this method, it should be avoid when possible, and you'd better use *DBIterator* instead.

To make a database query that could not be done using *DBIterator* (calling SQL functions such as `NOW()`, `ADD_DATE()`, ... for example), you can do:

```
<?php
$DB->request('SELECT id FROM glpi_users WHERE end_date > NOW()');
```



3.3 DBIterator

3.3.1 Goals

Provide a simple request generator:

- without having to write SQL
- without having to quote table and field name
- without having to take care of freeing resources
- iterable

3.3.2 Basic usage

```
<?php
foreach ($DB->request(...) as $id => $row) {
    //... work on each row ...
}

$req = $DB->request(...);
if ($row = $req->next()) {
    // ... work on a single row
}
```

3.3.3 Arguments

The request method takes two arguments:

- *table name(s)*: a string or an array of string (optional when given as FROM option)
- *option(s)*: array of options

3.3.4 Giving full SQL statement

If the only option is a full SQL statement, it will be used. This usage is deprecated, and should be avoid when possible.

3.3.5 Without option

In this case, all the data from the selected table is iterated:

```
<?php
$DB->request(['FROM' => 'glpi_computers']);
// => SELECT * FROM `glpi_computers`

$DB->request('glpi_computers');
// => SELECT * FROM `glpi_computers`
```

3.3.6 Fields selection

Using one of the SELECT, FIELDS, DISTINCT FIELDS or SELECT DISTINCT options

```
<?php
$DB->request(['SELECT' => 'id', 'FROM' => 'glpi_computers']);
// => SELECT `id` FROM `glpi_computers`

$DB->request('glpi_computers', ['FIELDS' => 'id']);
// => SELECT `id` FROM `glpi_computers`

$DB->request(['SELECT DISTINCT' => 'name', 'FROM' => 'glpi_computers']);
// => SELECT DISTINCT `name` FROM `glpi_computers`

$DB->request('glpi_computers', ['DISTINCT FIELDS' => 'name']);
// => SELECT DISTINCT `name` FROM `glpi_computers`
```

The fields array can also contain per table sub-array:

```
<?php
$DB->request('glpi_computers', ['FIELDS' => ['glpi_computers' => ['id', 'name']]];
// => SELECT `glpi_computers`.`id`, `glpi_computers`.`name` FROM `glpi_computers`
```

3.3.7 Multiple tables, native join

You need to use criteria, usually a FKEY, to describe howto join the tables:

```
<?php
$DB->request(['FROM' => ['glpi_computers', 'glpi_computerdisks'],
             'FKEY' => ['glpi_computers'=>'id',
                       'glpi_computerdisks'=>'computer_id']]);
$DB->request(['glpi_computers', 'glpi_computerdisks'],
             ['FKEY' => ['glpi_computers'=>'id',
                       'glpi_computerdisks'=>'computer_id']]);
// => SELECT * FROM `glpi_computers`, `glpi_computerdisks`
//      WHERE `glpi_computers`.`id` = `glpi_computerdisks`.`computer_id`
```

3.3.8 Left join

Using the JOIN option, with some criteria, usually a FKEY:

```
<?php
$DB->request(['FROM' => 'glpi_computers',
             'JOIN' => ['glpi_computerdisks' => ['FKEY' => ['glpi_computers'=>'id',
                                                             'glpi_computerdisks'=>
                                                             'computer_id']]]]);
// => SELECT * FROM `glpi_computers`
//      LEFT JOIN `glpi_computerdisks`
//      ON (`glpi_computers`.`id` = `glpi_computerdisks`.`computer_id`)
```

3.3.9 Counting

Using the COUNT option:

```
<?php
$DB->request(['FROM' => 'glpi_computers', 'COUNT' => 'cpt']);
// => SELECT COUNT(*) AS cpt FROM `glpi_computers`
```

3.3.10 Order

Using the ORDER option, with value a field or an array of field. Field name can also contains ASC or DESC suffix.

```
<?php
$DB->request(['FROM' => 'glpi_computers', 'ORDER' => 'name']);
// => SELECT * FROM `glpi_computers` ORDER BY `name`

$DB->request('glpi_computers', ['ORDER' => ['date_mod DESC', 'name ASC']]);
// => SELECT * FROM `glpi_computers` ORDER BY `date_mod` DESC, `name` ASC
```

3.3.11 Request pager

Using the START and LIMIT options:

```
<?php
$DB->request('glpi_computers', ['START' => 5, 'LIMIT' => 10]);
// => SELECT * FROM `glpi_computers` LIMIT 10 OFFSET 5
```

3.3.12 Criteria

Other option are considered as an array of criteria (implicit logical AND)

The WHERE can also be used for legibility.

Simple criteria

A field name and its wanted value:

```
<?php
$DB->request(['FROM' => 'glpi_computers', 'WHERE' => ['is_deleted' => 0]]);
// => SELECT * FROM `glpi_computers` WHERE `is_deleted` = 0

$DB->request('glpi_computers', ['is_deleted' => 0,
                               'name' => 'foo']);
// => SELECT * FROM `glpi_computers` WHERE `is_deleted` = 0 AND `name` = 'foo'

$DB->request('glpi_computers', ['users_id' => [1,5,7]]);
// => SELECT * FROM `glpi_computers` WHERE `users_id` IN (1, 5, 7)
```

Logical OR, AND, NOT

Using the OR, AND, or NOT option with an array of criteria:

```
<?php
$DB->request('glpi_computers', ['OR' => ['is_deleted' => 0,
                                       'name' => 'foo']]);
// => SELECT * FROM `glpi_computers` WHERE (`is_deleted` = 0 OR `name` = 'foo')

$DB->request('glpi_computers', ['NOT' => ['id' => [1,2,7]]]);
// => SELECT * FROM `glpi_computers` WHERE NOT (`id` IN (1, 2, 7))
```

Operators

Default operator is =, but other operators can be used, by giving an array containing operator and value.

```
<?php
$DB->request('glpi_computers', ['date_mod' => ['>', '2016-10-01']]);
// => SELECT * FROM `glpi_computers` WHERE `date_mod` > '2016-10-01'

$DB->request('glpi_computers', ['name' => ['LIKE', 'pc00%']]);
// => SELECT * FROM `glpi_computers` WHERE `name` LIKE 'pc00%'
```

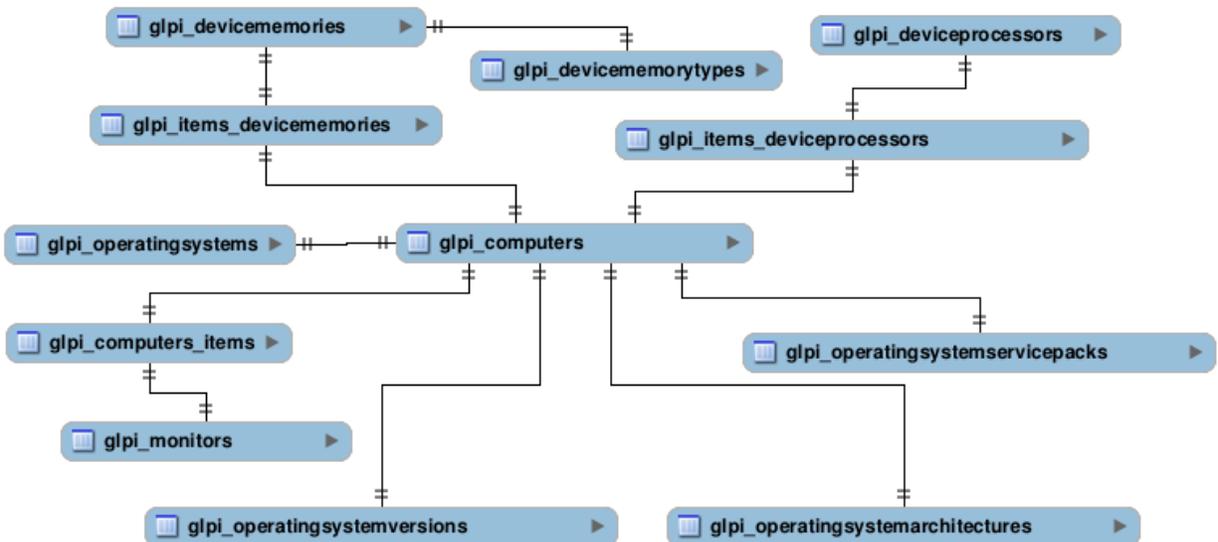
Know operators are =, <, <=, >, >=, LIKE, REGEXP, NOT LIKE and NOT REGEX.



3.4 Database model

Current GLPI database contains more than 250 tables; the goal of the current documentation is to help you to understand the logic of the project, not to detail each table and possibility.

As on every database, there are tables, relations between them (more or less complex), some relations have descriptions stored in a another table, some tables way be linked with themselves. . . Well, it's quite common :) Let's start with a simple example:



Note: The above schema is an example, it is far from complete!

What we can see here:

- computers are directly linked to operating systems, operating systems versions, operating systems architectures, ...,
- computers are linked to memories, processors and monitors using a relation table (which in that case permit to link those components to other items than a computer),
- memories have a type.

As stated in the above note, this is far from complete; but this is quite representative of the whole database schema.

3.4.1 Naming conventions

All tables and fields names are lower case and follows the same logic. If you do not respect that; GLPI will fail to find relevant informations.

Tables

Tables names are linked with PHP classes names; they are all prefixed with `glpi_`, and class name is set to plural. Plugins tables must be prefixed by `glpi_plugin_`; followed by the plugin name, another dash, and then pluralized class name.

A few examples:

PHP class name	Table name
Computer	glpi_computers
Ticket	glpi_tickets
ITILCategory	glpi_itilcategories
PluginExampleProfile	glpi_plugin_example_profiles

Fields

Warning: Each table **must** have an auto-incremented primary key named `id`.

Field naming is mostly up to you; except for identifiers and foreign keys. Just keep clear and concise!

To add a foreign key field; just use the foreign table name without `glpi_` prefix, and add `_id` suffix.

Warning: Even if adding a foreign key in a table should be perfectly correct; this is not the usual way things are done in GLPI, see *Make relations* to know more.

A few examples:

Table name	Foreign key field name
glpi_computers	computers_id
glpi_tickets	tickets_id
glpi_itilcategories	itilcategories_id
glpi_plugin_example_profiles	plugin_example_profiles_id

Make relations

On most cases, you may want to made possible to link many different items to something else. Let's say you want to make possible to link a *Computer*, a *Printer* or a *Phone* to a *Memory* component. You should add foreign keys in items tables; but on something as huge as GLPI, it maybe not a good idea.

Instead, create a relation table, that will reference the memory component along with a item id and a type, as for example:

```
CREATE TABLE `glpi_items_devicememories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `items_id` int(11) NOT NULL DEFAULT '0',
  `itemtype` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `devicememories_id` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  KEY `items_id` (`items_id`),
  KEY `devicememories_id` (`devicememories_id`),
  KEY `itemtype` (`itemtype`,`items_id`),
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Again, this is a very simplified example of what already exists in the database, but you got the point ;)

In this example, `itemtype` would be `Computer`, `Printer` or `Phone`; `items_id` the id of the related item.

3.4.2 Indexes

In order to get correct performances querying database, you'll have to take care of setting some indexes. It's a nonsense to add indexes on every fields in the database; but some of them must be defined:

- foreign key fields;
- fields that are very often used (for example fields like `is_visible`, `itemtype`, ...),
- primary keys ;)

You should just use the field name as key name.



3.5 Translations

Main GLPI language is british english (`en_GB`). All string in the source code must be in english, and marked as translatable, using some convenient functions.

Since 0.84; GLPI uses `gettext` for localization; and `Transifex` is used for translations. If you want to help translating GLPI, please register on [transifex](#) and join our [translation mailing list](#)

What the system is capable to do:

- replace variables (on LTR and RTL languages),
- manage plural forms,
- add context informations,
- ...

Here is the workflow used for translations:

1. Developers add string in the source code,
2. String are extracted to POT file,
3. POT file is sent to Transifex,
4. Translators translate,
5. Developers pull new translations from Transifex,
6. MO files used by GLPI are generated.

3.5.1 Functions

There are several standard functions you will have to use in order to get translations. Remember the translation domain will be *glpi* if not defined; so, for plugins specific translations, do not forget to set it!

Note: All translations functions take a `$domain` as argument; it defaults to `glpi` and must be changed when you are working on a plugin.

Simple translation

When you have a “simple” string to translate, you may use several functions, depending on the particular use case:

- `__($str, $domain='glpi')` (what you will probably use the most frequently): just translate a string,
- `_x($ctx, $str, $domain='glpi')`: same as `__()` but provide an extra context,
- `__s($str, $domain='glpi')`: same as `__()` but escape HTML entities,
- `_sx($ctx, $str, $domain='glpi')`: same as `__()` but provide an extra context and escape HTML entities,

Handle plural forms

When you have a string to translate, but which rely on a count or something. You may as well use several functions, depending on the particular use case:

- `_n($sing, $plural, $nb, $domain='glpi')` (what you will probably use the most frequently): give a string for singular form, another for plural form, and set current “count”,
- `_sn($str, $domain='glpi')`: same as `_n()` but escape HTML entities,
- `_nx($ctx, $str, $domain='glpi')`: same as `_n()` but provide an extra context,

Handle variables

You may want to replace some parts of translations; for some reason. Let's say you would like to display current page on a total number of pages; you will use the `sprintf` method. This will allow you to make replacements; but without relying on arguments positions. For example:

```
<?php
$pages = 20; //total number of pages
$current = 2; //current page
$string = sprintf(
    __('Page %1$s on %2$s'),
    $pages,
    $total
);
echo $string; //will display: "Page 2 on 20"
```

In the above example, `%1$s` will always be replaced by 2; even if places has been changed in some translations.

Warning: You may sometimes see the use of `printf()` which is an equivalent that directly output (echo) the result. This should be avoided!



3.6 Tools

Differents tools are available on the `tools` folder; here is an non exhaustive list of provided features.

3.6.1 locale/

The locale directory contains several scripts used to maintain *translations* along with Transifex services:

- `extract_template.sh` is used to extract translated string to the POT file (before sending it to Transifex),
- `locale\update_mo.pl` compiles MO files from PO file (after they've been updated from transifex).

3.6.2 cliinstall.php

Installs a GLPI database from command line.

You have to specify both user and database name at last, using `--user` and `--db` parameters.

You can also add extra parameters:

- `--host` let you specify database host. It defaults to *localhost*,
- `--pass` let you specify database user's password. It defaults to no password,
- `--lang` let you specify language that will be used. It defaults to `en_GB`,
- `--tests`
- `--force` will force the installation even if database is already populated, **all existing data will be lost**.

You can get the usage by simply calling the script:

```
$ php tools/cliinstall.php
usage: tools/cliinstall.php [ --host=<dbhost> ] --db=<dbname> --user=<dbuser> [ --
↳pass=<dbpassword> ] [ --lang=xx_XX ] [ --tests ] [ --force ]
```

3.6.3 cliupdate.php

Let you update your GLPI database from command line. The script does not take any argument, just call it and you're done:

```
$ php tools/cliupdate.php
Current GLPI Data version: 9.2
Current GLPI Code version: 9.2
Default GLPI Language: en_GB
No migration needed.
```

3.6.4 genapidoc.sh

Generate GLPI phpdoc using [apigen](#). `apigen` command must be available in your path.

Generated documentation will be available in the `api` directory. Note that you can also look at the [online version](#).

3.6.5 convert_search_options.php

Search options have changed in GLPI 9.2 (see [PR #1396](#)). This script is a helper to convert existing search options to new way.

Note: The script output can probably **not be used as is**; but it would probably help you a lot!

3.6.6 make_release.sh

Builds GLPI release tarball:

- install and cleanup third party libraries,
- remove files and directories that should not be part of tarball,
- minify CSS and Javascript files,
- ...

3.6.7 modify_headers.pl

Update copyright header based on the contents of the `HEADER` file.

3.6.8 getsearchoptions.php

This script is designed to be called from a browser, not from the command line. It will display existing search options for an item specified with the `type` argument.

For example, open `http://localhost/glpi/tools/getsearchoptions.php?type=Computer`, and you will see search options for *Computer* type.

3.6.9 Not yet documented...

Note: Following scripts are not yet documented... Feel free to open a pull request to add them!

- `autoupdatelocales.sh`: Probably obsolete
- `check_dict.pl`
- `check_functions.pl`
- `checkforms.php`: Check forms opened / closed
- `checkfunction.php`: Check for obsolete function usage
- `cleanhistory.php`: Purge history with some criteria
- `diff_plugin_locale.php`: Probably obsolete
- `find_twin_in_dict.sh`: Check duplicates key in language template
- `findtableswithoutclass.php`
- `fix_utf8_bomfiles.sh`
- `fk_generate.php`
- `genphpcov.sh`
- `glpiuser.php`
- `ldap-glpi.ldif`: An LDAP export
- `ldap-schema.txt`: An LDAP export
- `ldapsync.php`
- `notincludedlanguages.php`: Get all po files not used in GLPI
- `test_langfiles.php`
- `testmail.php`
- `testunit.php`
- `update_registered_ids.php`: Purge history with some criteria

3.6.10 Out of date

Warning: Those tools are outdated, and kept for reference, or need some work to be working again. Use them at your own risks, or do not use them at all :)

phpunit/

This directory contains a set of unit tests that have not really been integrated in the project. Since, some unit tests have been rewritten, but not everything has been ported :/

php.vim

A vimfile for autocompletion and highlithing in VIM. This one is very outaded; it should be replaced with a most recent version, or being removed.

generate_bigdump.php

This script is designed to generate many data in your GLPI instance. It relies on the `generate_bigdump.function.php` file.



ESP32 is bluetooth+wifi

4.1 Review process

Here is the process you must follow when you are reviewing a PR.

1. Make sure the destination branch is the correct one:
 - *master* for new features,
 - *xx/bugfixes* for bug fixes
2. Check if unit tests are not failing,
3. Check if coding standards checks are not failing,
4. Review the code itself. It must follow *GLPI's coding standards*,
5. Using the Github review process, approve, request changes or just comment the PR,
 - If some new methods are added, or if the request made important changes in the code, you should ask the developer to write some more unit tests
6. A PR can be merged if two developers approved it, or if one developer approved it more than one day ago,
7. A bugfix PR that has been merged into the *xx/bugfixes* branch must be reported on the *master* branch. If the *master* already contains many changes, you may have to change some code before doing this. If changes are consequent, maybe should you open a new PR against the *master* branch for it,
8. Say thanks to the contributor :-)



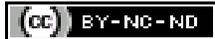
4.2 Prepare next major release

Once a major release has been finished, it's time to think about the next one!

You'll have to remember a few steps in order to get that working well:

- bump version in `config/define.php`
- create SQL empty script (copying last one) in `install/mysql/glpi-{version}-empty.sql`
- change empty SQL file calls in `inc/toolbox.class.php` (look for the `$DB->runFile` call)
- create a PHP migration script copying provided template `install/update_xx_xy.tpl.php`
 - change its main comment to reflect reality
 - change method name
 - change version in `displayTitle` and `setVersion` calls
- add the new case in `install/update.php` and `tools/cliupdate.php`; that will include your new PHP migration script and then call the function defined in it
- change the `include` and the function called in the `--force` option part of the `tools/cliupdate.php` script

That's all, folks!



GLPI provides facilities to develop plugins, and there are many [plugins that have been already published](#).

Generally speaking, there is really a few things you have to do in order to get a plugin working; many considerations are up to you. Anyways, this guide will provide you some guidelines to get a plugins repository as consistent as possible :)

If you want to see more advanced examples of what it is possible to do with plugins, you can take a look at the [example plugin source code](#).

5.1 Requirements

- plugin will be installed by creating a directory in the `plugins` directory of the GLPI instance,
- plugin directory name should never change,
- each plugin **must** at least provides `setup.php` and `hook.php` files,
- if your plugin requires a newer PHP version than GLPI one, or extensions that are not mandatory in core; it is up to you to check that in the install process.

5.1.1 setup.php

The plugin's `setup.php` file will be automatically loaded from GLPI's core in order to get its version, to check pre-requisites, etc.

This is a good practice, thus not mandatory, to define a constant name `{PLUGINNAME}_VERSION` in this file.

This is a minimalist example, for a plugin named `myexample` (functions names will contain plugin name):

```
<?php
define('MYEXAMPLE_VERSION', '1.2.10');
```

```

/**
 * Init the hooks of the plugins - Needed
 *
 * @return void
 */
function plugin_init_myexample() {
    //some code here, like call to Plugin::registerClass(), populating PLUGIN_HOOKS? ..
    ↪.
}

/**
 * Get the name and the version of the plugin - Needed
 *
 * @return array
 */
function plugin_version_myexample() {
    return [
        'name'           => 'Plugin name that will be displayed',
        'version'        => MYEXAMPLE_VERSION,
        'author'         => 'John Doe and <a href="http://foobar.com">Foo Bar</a>',
        'license'        => 'GLPv3',
        'homepage'       => 'http://perdu.com',
        'minGlpVersion' => '9.1'
    ];
}

/**
 * Optional : check prerequisites before install : may print errors or add to message_
↪after redirect
 *
 * @return boolean
 */
function plugin_myexample_check_prerequisites() {
    // Version check
    if (version_compare(GLPI_VERSION, '9.1', 'lt') || version_compare(GLPI_VERSION, '9.
↪2', 'ge')) {
        if (method_exists('Plugin', 'messageIncompatible')) {
            //since GLPI 9.2
            Plugin::messageIncompatible('core', 9.1, 9.2);
        } else {
            echo "This plugin requires GLPI >= 9.1 and < 9.2";
        }
        return false;
    }
    return true;
}

/**
 * Check configuration process for plugin : need to return true if succeeded
 * Can display a message only if failure and $verbose is true
 *
 * @param boolean $verbose Enable verbosity. Default to false
 *
 * @return boolean
 */
function plugin_myexample_check_config($verbose = false) {
    if (true) { // Your configuration check
        return true;
    }
}

```

```

    }

    if ($verbose) {
        echo "Installed, but not configured";
    }
    return false;
}

```

Note: Since GLPI 9.2, you can rely on `Plugin::messageIncompatible()` to display internationalized messages when GLPI or PHP versions are not met.

On the same model, you can use `Plugin::messageMissingRequirement()` to display internationalized message if any extension, plugin or GLPI parameter is missing.

5.1.2 hook.php

This file will contains hooks that GLPI may call under some user actions. Refer to core documentation to know more about available hooks.

For instance, a plugin need both an install and an uninstall hook calls. Here is the minimal file:

```

<?php
/**
 * Install hook
 *
 * @return boolean
 */
function plugin_myexample_install() {
    //do some stuff like instanciating databases, default values, ...
    return true;
}

/**
 * Uninstall hook
 *
 * @return boolean
 */
function plugin_myexample_uninstall() {
    //to some stuff, like removing tables, generated files, ...
    return true;
}

```

5.1.3 Coding standards

You must respect GLPI's *global coding standards*.

In order to check for coding standards compliance, you can add the *glpi-project/coding-standard* to your composer file, using:

```
$ composer require --dev glpi-project/coding-standard
```

This will install the latest version of the coding-standard used in GLPI core. If you want to use an older version of the checks (for example if you have a huge amount of work to fix!), you can specify a version in the above command like `glpi-project/coding-standard:0.5`. Refer to the [coding-standard project changelog](#) to know more ;)

You can then for example add a line in your `.travis.yml` file to automate checking:

```
script:
  - vendor/bin/phpcs -p --ignore=vendor --ignore=js --standard=vendor/glpi-project/
  ↪ coding-standard/GlpiStandard/ .
```

Note: Coding standards and their checks are enabled per default using the [empty plugin facilities](#)



5.2 Guidelines

5.2.1 Directories structure

Real structure will depend of what your plugin propose. See [requirements](#) to find out what is needed.

The plugin directory structure should look like the following:

-  *MyPlugin*
 -  *front*
 - *  ...
 -  *inc*
 - *  ...
 -  *locale*
 - *  ...
 -  *tools*
 - *  ...
 -  *README.md*
 -  *LICENSE*
 -  *setup.php*
 -  *hook.php*
 -  *MyPlugin.xml*
 -  *MyPlugin.png*
 -  ...
 -  ...

- *front* will host all PHP files directly used to display something to the user,
- *inc* will host all classes,
- if you internationalize your plugin, localization files will be found under the *locale* directory,
- if you need any scripting tool (like something to extract or update your translatable strings), you can put them in the *tools* directory

- a *README.md* file describing the plugin features, how to install it, and so on,
- a *LICENSE* file containing the license,
- *MyPlugin.xml* and *MyPlugin.png* can be used to reference your plugin on the [plugins directory website](#),
- the required *setup.php* and *hook.php* files.

5.2.2 Versioning

We recommend you to use [semantic versioning](#) for your plugins. You may find existing plugins that have adopted another logic; some have reasons, others don't... Well, it is up to you finally :-)

Whatever the versioning logic you adopt, you'll have to be consistent, it is not easy to change it without breaking things, once you've released something.

5.2.3 ChangeLog

Many projects make releases without providing any changelog file. It is not simple for any end user (whether a developer or not) to read a repository log or a list of tickets to know what have changed between two releases.

Keep in mind it could help users to know what have been changed. To achieve this, take a look at [Keep an ChangeLog](#), it will explain you some basics and give you guidelines to maintain such a thing.

5.2.4 Third party libraries

Just like GLPI, you should use the *composer tool to manage third party libraries* for your plugin.



5.3 Hooks

GLPI provides a certain amount of “hooks”. Their goal is for plugins (mainly) to work on certain places of the framework; like when an item has been added, updated, deleted, ...

This page describes current existing hooks; but not the way they must be implemented from plugins. Please refer to the plugins development documentation.

5.3.1 Standards Hooks

Usage

Aside from their goals or when/where they're called; you will see three types of different hooks. Some will receive an item as parameter, others an array of parameters, and some won't receive anything. Basically, the way they're declared into your plugin, and the way you'll handle that will differ.

All hooks called are defined in the *setup.php* file of your plugin; into the `$PLUGIN_HOOKS` array. The first key is the hook name, the second your plugin name; values can be just text (to call a function declared in the *hook.php* file), or an array (to call a static method from an object):

```
<?php
//call a function
$PLUGIN_HOOKS['hook_name']['plugin_name'] = 'function_name';
//call a static method from an object
$PLUGIN_HOOKS['other_hook']['plugin_name'] = ['ObjectName', 'methodName'];
```

Without parameters

Those hooks are called without any parameters; you cannot attach them to any itemtype; basically they'll permit you to display extra informations. Let's say you want to call the `display_login` hook, in you `setup.php` you'll add something like:

```
<?php
$PLUGIN_HOOKS['display_login']['myPlugin'] = 'myplugin_display_login';
```

You will also have to declare the function you want to call in you `hook.php` file:

```
<?php
/**
 * Display informations on login page
 *
 * @return void
 */
public function myplugin_display_login () {
    echo "That line will appear on the login page!";
}
```

The hooks that are called without parameters are: `display_central`, `post_init` `init_session`, `change_entity`, `change_profile` and `display_login`.

With item as parameter

Those hooks will send you an item instance as parameter; you'll have to attach them to the itemtypes you want to apply on. Let's say you want to call the `pre_item_update` hook for *Computer* and *Phone* item types, in your `setup.php` you'll add something like:

```
<?php
$PLUGIN_HOOKS['pre_item_update']['myPlugin'] = [
    'Computer' => 'myplugin_updateitem_called',
    'Phone'    => 'myplugin_updateitem_called'
];
```

You will also have to declare the function you want to call in you `hook.php` file:

```
<?php
/**
 * Handle update item hook
 *
 * @param CommonDBTM $item Item instance
 *
 * @return void
 */
public function myplugin_updateitem_called (CommonDBTM $item) {
    //do everything you want!
```

```

//remember that $item is passed by reference (it is an object)
//so changes you will do here will be used by the core.
if ($item::getType() === Computer::getType()) {
    //we're working with a computer
} elseif ($item::getType() === Phone::getType()) {
    //we're working with a phone
}
}

```

The hooks that are called wit item as parameter are: `item_empty`, `pre_item_add`, `post_prepareadd`, `item_add`, `pre_item_update`, `item_update`, `pre_item_purge`, `pre_item_delete`, `item_purge`, `item_delete`, `pre_item_restore`, `item_restore`, `autoinventory_information`, `item_add_targets`, `item_get_events`, `item_action_targets`, `item_get_datas`.

With array of parameters

These hooks will work just as the *hooks with item as parameter* expect they will send you an array of parameters instead of only an item instance. The array will contain two entries: `item` and `options`, the first one is the item instance, the second options that have been passed:

```

<?php
/**
 * Function that handle a hook with array of parameters
 *
 * @param array $params Array of parameters
 *
 * @return void
 */
public function myplugin_params_hook(array $params) {
    print_r($params);
    //Will display:
    //Array
    //(
    //    [item] => Computer Object
    //        (...)
    //
    //    [options] => Array
    //        (
    //            [_target] => /front/computer.form.php
    //            [id] => 1
    //            [withtemplate] =>
    //            [tabnum] => 1
    //            [itemtype] => Computer
    //        )
    //)
}

```

The hooks that are called wit an array of parameters are: `post_item_form`, `pre_item_form`, `pre_show_item`, `post_show_item`, `pre_show_tab`, `post_show_tab`, `item_transfer`.

Some hooks will receive a specific array as parameter, they will be detailed below.

Unclassified

Hooks that cannot be classified in above categories :)

add_javascript Add javascript in **all** pages headers

New in version 9.2: Minified javascript files are checked automatically. You will just have to provide a minified file along with the original to get it used!

The name of the minified `plugin.js` file must be `plugin.min.js`

add_css Add CSS stylesheet on **all** pages headers

New in version 9.2: Minified CSS files are checked automatically. You will just have to provide a minified file along with the original to get it used!

The name of the minified `plugin.css` file must be `plugin.min.css`

display_central Displays something on central page

display_login Displays something on the login page

status Displays status

post_init After the framework initialization

rule_matched After a rule has matched.

This hook will receive a specific array that looks like:

```
<?php
$hook_params = [
    'sub_type' => 'an item type',
    'rule_id'  => 'tule id',
    'input'    => array(), //original input
    'output'   => array() //output modified by rule
];
```

init_session At session initialization

change_entity When entity is changed

change_profile When profile is changed

Items business related

Hooks that can do some busines stuff on items.

item_empty When a new (empty) item has been created. Allow to change / add fields.

post_prepareadd Before an item has been added, after `prepareInputForAdd()` has been run, so after rule engine has ben run, allow to edit `input` property, setting it to false will stop the process.

pre_item_add Before an item has been added, allow to edit `input` property, setting it to false will stop the process.

item_add After adding an item, `fields` property can be used.

pre_item_update Before an item is updated, allow to edit `input` property, setting it to false will stop the process.

item_update While updating an item, `fields` and `updates` properties can be used.

pre_item_purge Before an item is purged, allow to edit `input` property, setting it to false will stop the process.

item_purge After an item is purged (not pushed to trash, see `item_delete`). The `fields` property still available.

pre_item_restore Before an item is restored from trash.

item_restore After an item is restored from trash.

pre_item_delete Before an item is deleted (moved to trash), allow to edit `input` property, setting it to false will stop the process.

item_delete After an item is moved to trash.

autoinventory_information After an automated inventory has occurred

item_transfer When an item is transferred from an entity to another

item_can New in version 9.2.

Allow to restrict user rights (can't grant more right). If `right` property is set (called during `CommonDBTM::can`) changing it allow to deny evaluated access. Else (called from `Search::addDefaultWhere`) `add_where` property can be set to filter search results.

Items display related

Hooks that permits to add display on items.

pre_item_form New in version 9.1.2.

Before an item is displayed; just after the form header if any; or at the beginning of the form. Waits for a `<tr>`.

post_item_form New in version 9.1.2.

After an item form has been displayed; just before the dates or the save buttons. Waits for a `<tr>`.

pre_show_item Before an item is displayed

post_show_item After an item has been displayed

pre_show_tab Before a tab is displayed

post_show_tab After a tab has been displayed

Notifications

Hooks that are called from notifications

item_add_targets When a target has been added to an item

item_get_events After notifications events have been retrieved

item_action_targets After target addresses have been retrieved

item_get_datas After data for template have been retrieved

5.3.2 Functions hooks

Usage

Functions hooks declarations are the same than standards hooks one. The main difference is that the hook will wait as output what have been passed as argument.

```
<?php
/**
 * Handle hook function
 *
 * @param array $$data Array of something (assuming that's what we're receiving!)
```

```
*
* @return array
*/
public function myplugin_updateitem_called ($data) {
    //do everything you want
    //return passed argument
    return $data;
}
```

Existing hooks

unlock_fields After a fields has been unlocked. Will receive the `$_POST` array used for the call.

restrict_ldap_auth Additional LDAP restrictions at connection. Must return a boolean. The dn string is passed as parameter.

undiscloseConfigValue Permit plugin to hide fields that should not appear from the API (like configuration fields, etc). Will receive the requested fields list.

infocom Additional infocom informations oin an item. Will receive an item instance as parameter, is expected to return a table line (`<tr>`).

retrieve_more_field_from_ldap Retrieve additional fields from LDAP for a user. Will receive the current fields lists, is expected to return a fields list.

retrieve_more_data_from_ldap Retrieve additional data from LDAP for a user. Will receive current fields list, is expected to return a fields list.

display_locked_fields To manage fields locks. Will receive an array with `item` and `header` entries. Is expected to output a table line (`<tr>`).

migratetypes Item types to migrate, will receive an array of types to be updated; must return an array of item types to migrate.

5.3.3 Automatic hooks

Some hooks are automated; they'll be called if the relevant function exists in you plugin's `hook.php` file. Required function must be of the form `plugin_{plugin_name}_{hook_name}`.

MassiveActionsFieldsDisplay Add massive actions. Will receive an array with `item` (the item type) and `options` (the search options) as input. These hook have to output its content, and to return true if there is some specific output, false otherwise.

dynamicReport Add parameters for print. Will receive the `$_GET` array used for query. Is expected to return an array of parameters to add.

AssignToTicket Declare types an ITIL object can be assigned to. Will receive an empty array and is expected to return a list an array of type of the form:

```
<?php
return [
    'TypeClass' => 'label'
];
```

MassiveActions If plugin is parameted to provide massive actions (via `$PLUGIN_HOOKS['use_massive_actions']`), will pass the item type as parameter, and expect an array of additional massives actions of the form:

```
<?php
return [
    'Class::method' => 'label'
];
```

getDropDown To declare extra dropdowns. Will not receive any parameter, and is expected to return an array of the form:

```
<?php
return [
    'Class::method' => 'label'
];
```

rulePrepareInputDataForProcess Provide data to process rules. Will receive an array with `item` (data used to check criteria) and `params` (the parameters) keys. Is expected to return an array of rules.

executeActions Actions to execute for rule. Will receive an array with `output`, `params` and `action` keys. Is expected to return an array of actions to execute.

`preProcessRulePreviewResults`

Todo: Write documentation for this hook.

`use_rules`

Todo: Write documentation for this hook. It lloks at bit particular.

ruleCollectionPrepareInputDataForProcess Prepare input data for rules collections. Will receive an array of the form:

```
<?php
array(
    'rule_itemtype' => 'name fo the rule itemtype',
    'values' => array(
        'input' => 'input array',
        'params' => 'array of parameters'
    )
);
```

Is expected to return an array.

`preProcessRuleCollectionPreviewResults`

Todo: Write documentation for this hook.

ruleImportComputer_addGlobalCriteria Add global criteria for computer import. Will receive an array of global criteria, is expected to return global criteria array.

ruleImportComputer_getSqlRestriction Adds SQL restriction to links. Will receive an array of the form:

```
<?php
array(
    'where_entity' => 'where entity clause',
```

```
'input'      => 'input array',  
'criteria'   => 'complex cirteria array',  
'sql_where'  => 'sql where clause as string',  
'sql_from'   => 'sql from clause as string'  
)
```

Is expected to return the input array modified.



Various Linux distributions provides packages (*deb*, *rpm*, ...) for GLPI (Debian, Mandriva, Fedora, Redhat/CentOS, ...) and for some plugins.

Here is some information about using and creating package:

- for users to understand how GLPI is installed
- for support to understand how GLPI work on this installation
- for packagers

6.1 Sources

GLPI public tarball is designed for ends-user; it will not fit packaging requirements. For example, this tarball bundle a lot of third party libraries, it does not ships unit tests, etc.

A better candidate would be to retrieve directly a tarball from github as package source.

6.2 Filesystem Hirerarchie Standard

Most distributions requires that packages follows the [FHS \(Filesystem Hierarchy Standard\)](#):

- `/etc/glpi` for configuration files: `config_db.php` and `config_db_slave.php`. Prior to 9.2 release, other files stay in `glpi/config`; beginning with 9.2, those files have been moved;
- `/usr/share/glpi` for the web pages (read only dir);
- `/var/lib/glpi/files` for GLPI data and state information (session, uploaded documents, cache, cron, plugins, ...);
- `/var/log/glpi` for various GLPI log files.

The magic file `/usr/share/glpi/config/config_path.php` (not provided in the tarball) allows to configure various paths. The following example is the file used by Remi on its Fedora/Redhat repository:

```
<?php
// Config
define('GLPI_CONFIG_DIR',      '/etc/glpi');

// Runtime Data
define('GLPI_DOC_DIR',         '/var/lib/glpi/files');
define('GLPI_CRON_DIR',       GLPI_DOC_DIR . '/_cron');
define('GLPI_DUMP_DIR',       GLPI_DOC_DIR . '/_dumps');
define('GLPI_GRAPH_DIR',      GLPI_DOC_DIR . '/_graphs');
define('GLPI_LOCK_DIR',       GLPI_DOC_DIR . '/_lock');
define('GLPI_PICTURE_DIR',    GLPI_DOC_DIR . '/_pictures');
define('GLPI_PLUGIN_DOC_DIR', GLPI_DOC_DIR . '/_plugins');
define('GLPI_RSS_DIR',        GLPI_DOC_DIR . '/_rss');
define('GLPI_SESSION_DIR',    GLPI_DOC_DIR . '/_sessions');
define('GLPI_TMP_DIR',        GLPI_DOC_DIR . '/_tmp');
define('oGLPI_UPLOAD_DIR',    GLPI_DOC_DIR . '/_uploads');

// Log
define('GLPI_LOG_DIR',        '/var/log/glpi');

// System libraries
define('GLPI_HTMLAWED',       '/usr/share/php/htmlawed/htmlawed.php');

// Fonts
define('GLPI_FONT_FREESANS',  '/usr/share/fonts/gnu-free/FreeSans.ttf');

//Use system cron
define('GLPI_SYSTEM_CRON',   true);
```

6.3 Apache Configuration File

Here is a configuration file sample for the Apache web server:

```
#To access via http://servername/glpi/
Alias /glpi /usr/share/glpi

# some people prefer a simple URL like http://glpi.example.com
#<VirtualHost *:80>
# DocumentRoot /usr/share/glpi
# ServerName glpi.example.com
#</VirtualHost>

<Directory /usr/share/glpi>
    Options None
    AllowOverride None

    # to overwrite default configuration which could be less than recommended value
    php_value memory_limit 64M

    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all granted
    </IfModule>
```

```

<IfModule !mod_authz_core.c>
    # Apache 2.2
    Order Deny,Allow
    Allow from All
</IfModule>
</Directory>

<Directory /usr/share/glpi/install>
    # 15" should be enough for migration in most case
    php_value max_execution_time 900
    php_value memory_limit 128M
</Directory>

# This sections remplace the .htaccess files provided in the tarball
<Directory /usr/share/glpi/config>
    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all denied
    </IfModule>
    <IfModule !mod_authz_core.c>
        # Apache 2.2
        Order Deny,Allow
        Deny from All
    </IfModule>
</Directory>

<Directory /usr/share/glpi/locales>
    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all denied
    </IfModule>
    <IfModule !mod_authz_core.c>
        # Apache 2.2
        Order Deny,Allow
        Deny from All
    </IfModule>
</Directory>

<Directory /usr/share/glpi/install/mysql>
    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all denied
    </IfModule>
    <IfModule !mod_authz_core.c>
        # Apache 2.2
        Order Deny,Allow
        Deny from All
    </IfModule>
</Directory>

<Directory /usr/share/glpi/scripts>
    <IfModule mod_authz_core.c>
        # Apache 2.4
        Require all denied
    </IfModule>
    <IfModule !mod_authz_core.c>
        # Apache 2.2
        Order Deny,Allow

```

```
Deny from All
</IfModule>
</Directory>
```

6.4 Logs files rotation

Here is a logrotate sample configuration file (`/etc/logrotate.d/glpi`):

```
# Rotate GLPI logs daily, only if not empty
# Save 14 days old logs under compressed mode
/var/log/glpi/*.log {
    daily
    rotate 14
    compress
    notifempty
    missingok
    create 644 apache apache
}
```

6.5 SELinux stuff

For SELinux enabled distributions, you need to declare the correct context for the folders.

As an example, on Redhat based distributions:

- `/etc/glpi` and `/var/lib/glpi`: `httpd_sys_script_rw_t`, the web server need to write the config file in the former and various data in the latter;
- `/var/log/glpi`: `httpd_log_t` (apache log type: write only, no delete).

6.6 Use system cron

GLPI provides an internal cron for automated tasks. Using a system cron allow a more consistent and regular execution, for example when no user connected on GLPI.

Note: `cron.php` should be run as the web server user (`apache` or `www-data`)

You will need a crontab file, and to configure GLPI to use system cron. Sample cron configuration file (`/etc/cron.d/glpi`):

```
# GLPI core
# Run cron from to execute task even when no user connected
*/4 * * * * apache /usr/bin/php /usr/share/glpi/front/cron.php
```

To tell GLPI it must use the system crontab, simply define the `GLPI_SYSTEM_CRON` constant to `true` in the `config_path.php` file:

```
<?php
//[...]
```

```
//Use system cron
define('GLPI_SYSTEM_CRON', true);
```

6.7 Using system libraries

Since most distributions prefers the use of system libraries (maintained separately); you can't rely on the vendor directory shipped in the public tarball; nor use composer.

The way to handle third party libraries is to provide an autoload file with paths to you system libraries. You'll find all requirements from the `composer.json` file provided along with GLPI:

```
<?php
$vendor = '##DATADIR##/php';
// Dependencies from composer.json
// "ircmaxell/password-compat"
// => useless for php >= 5.5
//require_once $vendor . '/password_compat/password.php';
// "jasig/phpcas"
require_once '##DATADIR##/pear/CAS/Autoload.php';
// "iamcal/lib_autolink"
require_once $vendor . '/php-iamcal-lib-autolink/autoload.php';
// "phpmailer/phpmailer"
require_once $vendor . '/PHPMailer/PHPMailerAutoload.php';
// "sabre/vobject"
require_once $vendor . '/Sabre/VObject/autoload.php';
// "simplepie/simplepie"
require_once $vendor . '/php-simplepie/autoloader.php';
// "tecnickcom/tcpdf"
require_once $vendor . '/tcpdf/tcpdf.php';
// "zendframework/zend-cache"
// "zendframework/zend-i18n"
// "zendframework/zend-loader"
require_once $vendor . '/Zend/autoload.php';
// "zetacomponents/graph"
require_once $vendor . '/ezc/Graph/autoloader.php';
// "ramsey/array_column"
// => useless for php >= 5.5
// "michelf/php-markdown"
require_once $vendor . '/Michelf/markdown-autoload.php';
// "true/punycode"
if (file_exists($vendor . '/TrueBV/autoload.php')) {
    require_once $vendor . '/TrueBV/autoload.php';
} else {
    require_once $vendor . '/TrueBV/Punycode.php';
}
```

Note: In the above example, the `##DATADIR##` value will be replaced by the correct value (`/usr/share/php` for instance) from the specfile using macros. Adapt with your build system possibilities.

6.8 Using system fonts rather than bundled ones

Some distribution prefers the use of system fonts (maintained separately).

GLPI use the `FreeSans.ttf` font you can configure adding in the `config_path.php`:

```
<?php
//[...]

define('GLPI_FONT_FREESANS', '/path/to/FreeSans.ttf');
```

6.9 Notes

This informations are taken from the Fedora/EPEL spec file.

See latest version of the files:

- `config_path.path`
- `nginx/glpi.conf`
- `httpd/glpi.conf`
- `fedora-autoloader.php`

Feel free to add information about other specific distribution tips.

