FusionDirectory development Documentation

Release 1.2

Benoit Mortier Côme Chilliet

Feb 13, 2019

Contents

Contents:

CHAPTER 1

Contributing to FusionDirectory

How to contribute to FusionDirectory

1.1 Registering a FusionDirectory Account

To create a FusionDirectory project Account you must go to FusionDirectory Sign UP

1.2 Guidelines for better contributions

There is a complete set of guidelines that you should read

Contributing to FusionDirectory

CHAPTER 2

Coding Standards

2.1 Source Code management

FusionDirectory source code management is handled by GIT and hosted on GitHub.

There is two repositories :

- FusionDirectory Core
- FusionDirectory plugins

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

2.1.1 Branches

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

- *master* contains latest released official code,
- *xxx-fixes* contains the next minor release source code,
- *xxx-dev* contains the next major release source-code,

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

The *xxx-fixes* branches is where bugs are fixed. This code is reputed as *stable*.

2.1.2 File Hierarchy System

Note: This lists current files and directories listed in the FusionDirectory **core** source code.

This is a brieve description of FusionDirectory core main folders and files:

- *.tx*: Transifex configuration
- *contrib*
  - *apache*
    - *fusiondirectory-apache.conf*: FusionDirectory apache configuration
  - *bin*
    - *fusiondirectory-insert-schema*: FusionDirectory console schema management tool
    - *fusiondirectory-setup*: FusionDirectory console management tool
  - *docs*
    - *README*: Global Readme
    - *README.cnconfig*: Readme about openldap and cn=config
    - *README.ldap-migration*: Readme about ldap migration issue when using FusionDirectory
    - *UPGRADE*: Full Documentation on how to upgrade from one version to another
  - *images*
    - *favicon.png*: FusionDirectory favicon in png
    - *favicon.svg*: FusionDirectory favicon in svg
    - *Fusiondirectory-logo-noir.eps*: FusionDirectory logo in Encapsulated Postscript
    - *Fusiondirectory-logo.png*: FusionDirectory logo in png
  - *lighttpd*
    - *fusiondirectory-lighttpd.conf*: FusionDirectory lighttpd configuration file
  - *man*
  - *openldap* : FusionDirectory core schema
  - *smarty*
    - *plugins*
      - *block.render.php*: FusionDirectory lighttpd configuration file
      - *function.filePath.php*: FusionDirectory lighttpd configuration file
      - *function.iconPath.php*: FusionDirectory lighttpd configuration file
      - *function.msgPool.php*: FusionDirectory lighttpd configuration file
  - *fusiondirectory.conf*: FusionDirectory configuration template
- *html*
  - *images*: static images
  - *include*: static images
  - *plugins*: static images
  - *themes*: static images

        - *.css: themes css
            - *icons*: icons for the theme following free desktop specification
            - *images*: static images for this theme if needed (not recommended)
            - *svg*: icons in svg format
- *ihtml* : smarty theme tpl folders
    - *themes*
        - *breezy* : official theme
        - *legacy* : old one for testing only
- *include* : core FusionDirectory library and helpers
    - *exporter*: export to pdf and xls
    - *password-methods*: all password methods understod by FusionDirectory
    - *select*: all object specific select dialog and methods
    - *simpleplugin*: core FusionDirectory library
- *locale*
    - *ar*: ISO code of the language
        - *fusiondirectory.po*: Gettext's translations
    - ...
- *plugins*
    - *addons*
    - *admin*: administration plugins
    - *config*: configuration plugins
    - *generic*: base core plugins
    - *personal* : base personal plugins
- *.gitignore*: Git ignore list
- *AUTHORS.txt*: list of FusionDirectory authors
- *Changelog*: Changes
- *COPYING*: Licence
- *README.md*: all about FusionDirectory :)

Note: This lists current files and directories that can be listed in the FusionDirectory **plugin** source code.

This is a brieve description of a FusionDirectory plugin folders and files:

- *addons*: used if the plugin put things in the addons menu category
- *admin*: main dir for all plugins going into the admin menu category
- *config*: configuration dir, used if the plugin need to store option in ldap
- *contrib*: used to put all the contributed files like schema, docs, manpages etc..
  - *openldap*: Schemas for the openldap server
  - *docs*: Documentation how to use the plugin
- *html*: used to put all the images or other public files
  - *plugins*
    - *plugin_name*
      - *images*: images which are not icons
  - *themes*
    - *breezy*
      - *icons*: icons to add to default breezy theme
      - *48*: sorted by size, for instance 48x48
      - *apps*: then by category, use apps for application icons
      - *myapp.png*: format should be png. This example file would be used as *geticon.php?context=applications&icon=myapp*
- *ihtml*: used to put all the smarty template files
  - *themes*
    - *breezy*: smarty templates
- *locale*: used for localization of the plugin
  - *en*: language iso code
    - *fusiondirectory.po*: message file
- *personal*: used when plugin is to be used to manage user properties
- *includes*: used for files available for inclusion for other plugins

2.2 Coding Style

2.2.1 Scope of style guidelines

In order to keep the code consistent, please use the following conventions. These conventions are no judgement call on your coding abilities, but more of a style and look call.

2.2.2 Performance and Readability

- It is more important to be correct than to be fast.
- It is more important to be maintainable than to be fast.
- Fast code that is difficult to maintain is likely going to be looked down upon.

2.2.3 Indentation and line length

- 2 spaces
- Max line width: 80

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

As a basic style rule, please use 2 spaces instead of tabulators. This will remove problems when using "diff".

Note: For VI users, this can be achieved by the following settings:

- set expandtab
- set shiftwidth=2
- set softtabstop=2
- set tabstop=2

2.2.4 Spacing

Use a space before affectations, around operators, before parenthesis or braces.

```
<?php
// Methods
foo($parameter);

// Arrays
$b = $value[0];

// Readability
if ($b + 5 > foo(bar() + 4)) {
}
```

For vars declaration place values on the same column

```
<?php
var $most           = "something";
var $iHaveALongName = "value";
var $otherName      = "otherValue";
```

Always use spaces to seperate arguments after commat:

```
<?php
function foo ($param1, $param2)
```

Always use single spaces to split logical and mathematical operations:

```
<?php
if ($a > 6 && $b == 17 && (foo($b) < 1)) {
}
```

2.2.5 Braces

If statements with or without else clauses are formatted like this:

```
<?php
if ($value) {
  foo();
  bar();
}

if ($value) {
  foo();
} else {
  bar();
}
```

Switches are formatted like this:

```
<?php
switch ($reason) {
  case 'fine':
    foo();
    break;

  case 'well':
    bar();
    break;
}
```

Always use use braces for single line blocks:

```
<?php
if ($value) {
  foo();
}
```

Function definitions, Classes and Methods have an opening brace on the next line:

```
<?php
function bar ()
{
...
}
```

2.2.6 Casing

Always use camel casing with lowercase characters in the beginning for multiword identifiers.

```
<?php
function checkForValidity ()
{
  $testSucceeded = FALSE;
  ...
}
```

2.2.7 Naming

Non trivial variable names should speak for themselves from within the context.

```
<?php
// Use
$hour = 5;
// instead of
$g = 5;
```

Find short function names that describe what the function does, in order to make the code read like a written sentence.

```
<?php
if (configReadable("/etc/foo.conf")) {
}
```

Use uppercase for constants/defines and _ to separate if there is more than one word :

```
<?php
if ($speedUp == TRUE) {
  $wait = SHORT_WAIT;
} else {
  $wait = LONG_WAIT;
}
```

2.2.8 Arrays

Arrays must be declared using the long notation syntax (`array()`).

2.2.9 PHP specific

Use return without parenthesis

```
<?php
return TRUE; // good

return(TRUE); // bad
```

Open and close tags

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

```php
<?php
  // Something here
?>
```

2.2.10 Including files

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

```php
<?php
require_once("class_setupStep.inc");
```

2.2.11 Quotes / double quotes

- You must use single quotes for indexes, constants declaration, translations, ...
- 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
<?php
//for that one, you should use single, 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 = array(
   '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' :
      ...
}
```

2.2.12 Files Format

- UTF-8, LF - not CR LF

2.2.13 Checking standards

In order to check some standards are respected, we provide some custom PHP CodeSniffer rules.

First clone the dev-tools that contains Fusiondirectory developement tools

```
git clone https://gitlab.fusiondirectory.org/fusiondirectory/dev-tools.git
```

Then run the codesniffer rules from the top directory:

```
find . -type f -name '*.php' -o -name '*.inc' -exec phpcs --standard=../dev-tools/php-
↪codesniffer-rules/FDStandard/ruleset.xml "{}" \;
```

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

An example error output would looks like:

```
FILE: fusiondirectory/fusiondirectory/include/class_ldap.inc
----------------------------------------------------------------------
FOUND 9 ERROR(S) AFFECTING 4 LINE(S)
----------------------------------------------------------------------
260 | ERROR | Case breaking statement indented incorrectly; expected 10
    |       | spaces, found 8
802 | ERROR | Assignment blocks should have all assignment tokens on the same
    |       | column
965 | ERROR | Expected 1 space before "?"; 0 found
965 | ERROR | Expected 1 space after "?"; 0 found
965 | ERROR | Expected 1 space before ":"; 0 found
965 | ERROR | Expected 1 space after ":"; 0 found
973 | ERROR | Expected 1 space before "?"; 0 found
973 | ERROR | Expected 1 space after "?"; 0 found
973 | ERROR | Expected 1 space before ":"; 0 found
----------------------------------------------------------------------
```

2.3 Headers for the files in FusionDirectory code

Each file inside FusionDirectory code must have an header mentioning copyright and license

The header should look like this

```
<?php
/*
  This code is part of FusionDirectory (http://www.fusiondirectory.org/)
  Copyright (C) 2017  FusionDirectory

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
```

(continues on next page)

(continued from previous page)

```
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.
*/
```

CHAPTER 3

Writing a plugin

You can write plugins for FusionDirectory using our simplePlugin class.

3.1 Plugin folder organization

The directories in a FusionDirectory plugin look like this:

- *addons*: used if the plugin put things in the addons menu category
- *admin*: main dir for all plugins going into the admin menu category
- *config*: configuration dir, used if the plugin need to store option in ldap
- *contrib*: used to put all the contributed files like schema, docs, manpages etc..
  - *openldap*: Schemas for the openldap server
  - *docs*: Documentation how to use the plugin
- *html*: used to put all the images or other public files
  - *plugins*
    - *plugin_name*
      - *images*: images which are not icons
  - *themes*
    - *breezy*
      - *icons*: icons to add to default breezy theme
      - *48*: sorted by size, for instance 48x48

            - *apps*: then by category, use apps for application icons
            - *myapp.png*: format should be png. This example file would be used as *geticon.php?context=applications&icon=myapp*

- *ihtml*: used to put all the smarty template files
  - *themes*
    - *breezy*: smarty templates
- *locale*: used for localization of the plugin
  - *en*: language iso code
    - *fusiondirectory.po*: message file
- *personal*: used when plugin is to be used to manage user properties
- *includes*: used for files available for inclusion for other plugins

3.1.1 addons, admin, config, personal

These directories should contain a subdirectory named as the plugin, or as an other plugin which we extend. For instance, argonaut plugin contains *admin/systems/argonaut/class_argonautClient.inc*

3.1.2 Installation of a plugin

For **addons**, **admin**, **config**, and **personal** folders, the content should go into <*fd_dir*>/*plugins*/<*dir*>/

For **html**, **ihtml**, **include**, the content should go into <*fd_dir*>/<*dir*>/

For **contrib/openldap**, the content should go into <*ldap_schemas_dir*>/*fusiondirectory*/

For **contrib/etc**, the content should go into */etc/fusiondirectory/*<*plugin_name*>.

For **contrib/doc**, the content may go into <*doc_dir*>/*fusiondirectory-plugin-*<*plugin_name*>.

Special cases:

- in **html/themes/<theme_name>**, svg folder may be ignored
- content of **locale** goes into <*fd_dir*>/*locale/plugins*/<*plugin_name*>/*locale*/

3.2 Getting started

This page is a how-to to help you write a dummy plugin.

3.2.1 Directory organization

Your plugin should take place in *{fd-directory}/plugins/addons/yourpluginname* for an addon.

Plugins adding a user tab should go into *{fd-directory}/plugins/personal/yourpluginname*.

Plugins adding a system tab should go into *{fd-directory}/plugins/admin/systems/yourpluginname*.

Plugins adding a service should go into *{fd-directory}/plugins/admin/systems/services/yourpluginname*.

Your main file should be named *class_MyPluginClass.inc*.

Your plugin should have a *main.inc* file if you intend it to display on its own (not as a tab of an other object).

3.2.2 Icons

If your plugin packs some icons, they need to be placed in the breezy icon theme: *{fd-directory}/html/themes/breezy/icons/{size}/{category}* Most of the time your icons are those of an application and should therefore be placed in the *apps* folder, which is for the category *applications*. For instance if the small icon for apache goes in *{fd-directory}/html/themes/breezy/icons/16/apps/apache.png* and is used in the code as *geticon.php?context=applications&icon=apache&size=16*

3.2.3 Basic plugin writing

This is the code for an empty plugin:

```
<?php
class demoPlugin extends simplePlugin
{
  // We set displayHeader to FALSE, because we don't want a header allowing to␣
↪activate/deactivate this plugin,
  // we want it activated on all objects
  var $displayHeader = FALSE;

  // Here we indicate which LDAP classes our plugin is using.
  var $objectclasses = array('demoPlugin');

  // We need this function that returns some information about the plugin
  static function plInfo ()
  {
    return array(
      'plShortName'       => _('Demo Plugin'),
      'plTitle'           => _('Demo Plugin informations'),
      'plDescription'     => _('Edit some useless personal information'),
      'plSelfModify'      => TRUE, // Does this plugin have an owner that might be␣
↪able to edit its entry
      'plObjectType'      => array('user'),

      // simplePlugin can generate the ACL list for us
      'plProvidedAcls'    => parent::generatePlProvidedAcls(self::getAttributesInfo())
    );
  }

  // The main function : information about attributes
  static function getAttributesInfo ()
  {
    return array(
    );
  }
}
```

With this code you'll have an empty plugin, just adding the "demoPlugin" objectClass. The *plInfo* static function must provide informations about your plugin. Please fill **plShortName** and **plDescription** with something meaningful (and **plTitle** as well if your plugin have its own page). See *plInfo* for more details about other fields

3.2.4 Attributes

You might have noticed the empty getAttributesInfo method. This is where the magic happens. You should fill this function with an array of sections containing attributes. Available attribute types are BooleanAttribute, IntAttribute, FloatAttribute, StringAttribute, SelectAttribute, PasswordAttribute... The names are pretty clear about what these attributes are. There are also three special kind of attributes, SetAttribute, ArrayAttribute and CompositeAttribute. SetAttribute and ArrayAttribute might both be used for multi-valuated attribute. Array will allow several identical values while Set won't. A composite attribute is a unique LDAP attribute composed of several displayed attributes. You'll see one in the following example.

For more information about each type of attribute, see *Attributes Types*

3.2.5 Example

```
<?php
// The main function : information about attributes
static function getAttributesInfo ()
{
  return array(
    // Attributes are grouped by section
    'section1' => array(
      'name'  => _('Hair Information'),
      'attrs' => array(
        new SetAttribute(                 // This attribute is multi-valuated
          new SelectAttribute (
            _('Color'),                     // Label of the attribute
            _('Color of the hair'),         // Description
            'hairColor',                    // LDAP name
            TRUE,                           // Mandatory
            array('blond','black','brown'), // [SelectAttribute] Choices
            "", // We don't set any default value, it will be the first one
            array('Blond','Black','Brown')  // [SelectAttribute] Output choices
          )
        ),
        new FloatAttribute  (
          _('Length'),                    // Label
          _('Length of the hair in cm'),  // Description
          'hairLength',                   // LDAP name
          FALSE,                          // Not mandatory
          0,                              // [FloatAttribute] Minimum value
          FALSE,                          // [FloatAttribute] No maximum value
          10                              // [FloatAttribute] Default value
        ),
      )
    ),
    'section2' => array(
      'name'  => _('Bicycle'),
      'attrs' => array(
        new StringAttribute (
          _('Brand'),                     // Label
          _('Brand of the bicycle'),      // Description
          'bicycleBrand',                 // LDAP name
          TRUE,                           // Mandatory
          'GreatBicycleBrand'             // Default value
        ),
        new BooleanAttribute (
```

(continues on next page)

(continued from previous page)

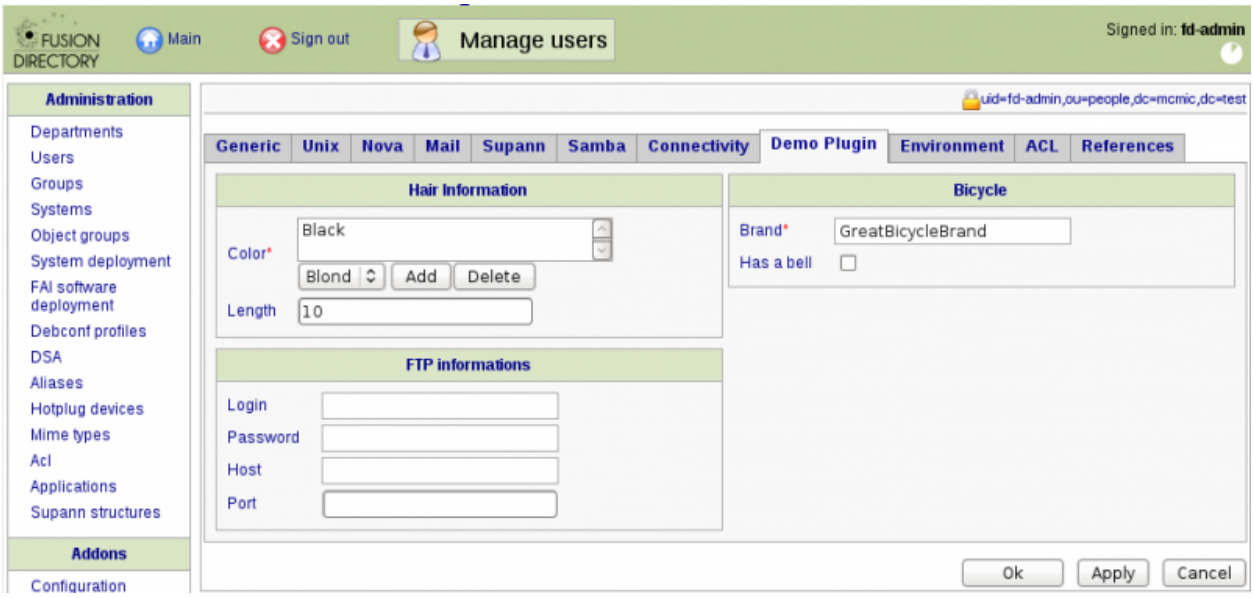
```
          _('Has a bell'),                    // Label
          _('Does the bicycle have a bell'),  // Description
          'bicycleBell',                      // LDAP name
          FALSE,                              // Not mandatory
          FALSE                               // Default value
        ),
      )
    ),
    'ftp' => array(
      'name'  => _('FTP informations'),
      'attrs' => array(
        new CompositeAttribute (
          _('Informations for ftp login'),
          'ftpLoginInfo',
          array(
            new StringAttribute (_('Login'),    _('Login for FTP'),     'ftpLogin'),
            new StringAttribute (_('Password'), _('Password for FTP'),  'ftpPassword
↪'),
            new StringAttribute (_('Host'),     _('Host for FTP'),      'ftpHost'),
            new IntAttribute    (_('Port'),     _('Port for FTP'),      'ftpPort',␣
↪FALSE, 0, FALSE, 21),
          ),
          'ftp://%[^@:]:%[^@:]@%[^@:]:%d',    // scanf format
          'ftp://%s:%s@%s:%d'                 // printf format
        )
      )
    ),
  );
}
```

As you can see, attribute constructor take 5 arguments being label, description, ldap name, whether this attribute is mandatory or not, default value. Some attributes takes other arguments before and after the default value. For each section you might also specify keys 'icon' with a section icon path, or 'class' with an array of css class this section should have. (Only useful class for now is 'fullwidth' which means your section will fill the whole page width)

3.2.6 Displaying the plugin in FusionDirectory

- Put the plugin code into a directory FusionDirectory is reading (see above).
- Run `fusiondirectory-setup --update-cache` as root.
- Log out, log in.

A tab should now shows in user edition mode, with the attributes we specified:



3.2.7 Displaying the plugin in the "My account" menu

You may also want the plugin to show in the "My Account" menu, if your plugin is for users and you've set plModifySelf to TRUE. For this, you need your plugin to have a main.inc PHP file. Just put this in it:

```
<?php
    simplePlugin::mainInc('demoPlugin', $ui->dn);
?>
```

3.3 Going further with Simple Plugin

Here, we'll see what Simple Plugin functions you can inherit in order to adjust the behavior of your plugin.

Because sometimes, you don't just want to edit LDAP fields.

3.3.1 simplePlugin special attributes

There are some attributes that you can set in your class or in your constructor that will allow you to do more things:

- Set **displayHeader** to TRUE if you need this tab to be deactivable.
- Set **mainTab** to TRUE if this plugin is the main tab.
- Set **preInitAttributes** to an array of attributes names to be sure they'll be initialized before the others (it's used by workstationGeneric for the network attribute)

custom template

By default simplePlugin does a template for you, but if you want to add some elements to the template, or just render the sections in a different order, or that kind of things, here's what to do: Change **templatePath** value to your custom template path (usually in the constructor, using get_template_path).

In your template, you'll be able to use the **$sections** array that contains each section render. For instance:

```
<h1>Hello world!</h1>
<div class="plugin_sections">
    {$sections.section1}
    {$sections.Mysection}
</div>

<input name="{$hiddenPostedInput}" value="1" type="hidden"/>

<!-- Place cursor -->
<script language="JavaScript" type="text/javascript">
  <!-- // First input field on page
    focus_field('{$focusedField}');
  -->
</script>
```

You need to add the **hidden** input at the end in order for the POST analysis to work. The script is needed if you want the auto-focusing of first field to work.

3.3.2 simplePlugin attributes values and methods

In all of these methods, you can access the attributes by using $this->attributesAccess as follows: $this->attributesAccess['attributeldapname']

Don't forget to look at the documentation of the Attribute classes to know who to use them. For instance they offer a **setDisabled** method if you need to disable some of them, **hasChanged** will allow you to know if an attribute has been modified, etc... You can also easily access their value using **$this->attributeldapname**. Be aware that this is not a real class attribute, accessing it will call the **getValue** and **setValue** methods of the attribute. That means you can't create reference to it or call method that needs references like the array ones (array_push, ...). The [] operator for arrays do not work either.

execute

This method is the one that render the plugin. You can change it, doing something that look like that:

```
<?php
function execute()
{
  $smarty = get_smarty();
  parent::execute();
  // your code goes here
  if ($this->displayPlugin) {
    return $this->header.$smarty->fetch($this->templatePath);
  } else {
    return $this->header;
  }
}
```

You can fetch any template but usually **$this->templatePath** is used, just remember to add **$this->header** at the beginning if you activated the display header feature.

Please avoid doing heavy things in the execute function as it is just the render function, it's not supposed to compute anything.

save_object

This function analyse the POST informations. You can inherit it as follows:

```
<?php
function save_object()
{
  parent::save_object();
  if (isset($_POST[get_class($this)."_posted"])) {
    // your code goes here
  }
}
```

ldap_save

This function saves the informations into the LDAP. You can inherit it and do some additionnal LDAP modifications when saving:

```
<?php
function ldap_save()
{
  $errors = parent::ldap_save();

  if (!empty($errors)) {
    return $errors;
  }

  // your code goes here

  return $errors;
}
```

prepare_save

prepare_save will fill the attribute **$this->attrs**, which is an array of what will be written into the LDAP. Your code should modify **$this->attrs** as ldap_save will write it into the LDAP.

```
<?php
function prepare_save()
{
  parent::prepare_save();

  // your code goes here
}
```

__construct

Of course, there is always the possibility to have your own constructor, just remember to call the parent one at the end. The simple plugin constructor have a 3rd optional parameter which is the attributes information. If you don't give it, the **getAttributesInfo** static function will be used. So you can do the following:

```php
<?php
function __construct($dn = NULL, $object = NULL)
{
  $attributesInfo = self::getAttributesInfo();
  // some modifications on $attributesInfo
  parent::__construct($dn, $object, $attributesInfo);
}
```

An other method, often simpler, is to modify your attributes after being constructed. You can't do that for all modifications but for common cases like SelectAttribute choices modification, it's what you should do:

```php
<?php
function __construct($dn = NULL, $object = NULL)
{
  parent::__construct($dn, $object);

  $array = array('node1','node2'); // some dummy array
  // After simplePlugin constructor, you must access attributes by their ldap name
  $this->attributesAccess['myattributeLdapName']->setChoices($array);
}
```

is_this_account

This method is used to check if an object has your plugin tab activated or not. By default it will just return TRUE if the objectClasses of your tab are present and FALSE otherwise, it is usually correct. If you need an other behaviour, you will have to override it.

function is_this_account($attrs)

Even if the method is not static, it's not supposed to use the object attributes and should only use the information in the attrs parameter to tell if the LDAP node has this tab activated or not.

3.3.3 Section templates

We've seen that you can use a specific template for your plugin instead of the default one, and that sections are pre-rendered in a sections array. Here, we'll see how to use a specific template for a section, in order to modify its organization. It's quite easy to do, all you have to do is adding a 'template' key to the section array in getAttributesInfo:

```
'my_section' => array(
  'name'  => _('Great Section'),
  'attrs' => array(
    new StringAttribute (_('Something'), _('This attribute does nothing'), 'someThing
↪', FALSE, 'DefaultValue'),
    // other attributes...
  ),
  'template' => get_template_path('my_section_template.tpl', TRUE, dirname(__FILE__))
),
```

You need to use get_template_path as above in order to get an absolute path for the tpl file. In this template file, you need to copy simpleplugin_section.tpl, the default template. Please don't touch the fieldset, legend and table, just replace the foreach by what you want. You need to use the attributes array, which contain for each attribute, indexed by its ldap name, its label and its input html code. For instance, for the above section, doing the following would have the same result than the default template:

```
<fieldset id="{$sectionId}" class="plugin_section{$sectionClasses}">
  <legend>{$section}</legend>
  <table>
    <tr>
      <td title="{$attributes.someThing.description}"><label for="someThing">{eval␣
↪var=$attributes.someThing.label}</label></td>
      <td>{eval var=$attributes.someThing.input}</td>
    </tr>
  </table>
</fieldset>
```

You need to use 'eval' for label and HTML input as it contains some smarty code too (for ACL check for instance).

3.3.4 Managed attributes

In some case you want some attributes to be enabled/disabled depending on a checkbox or select state. For this, you can use the **setManagedAttributes** method as follow:

```
$this->attributesAccess['boolean']->setManagedAttributes (
  array(
    'disable' => array (
      FALSE => array (
        'attribute1',
        'attribute2',
      )
    )
  )
);
```

'disable' means that the attributes will be disabled but still saved into the LDAP. you can use 'erase' instead if you want those to be remove from the LDAP. FALSE means that when the value is FALSE, they'll be disabled. You can also use this method with selectattributes:

```
$this->attributesAccess['select']->setManagedAttributes (
  array(
    'multiplevalues' => array ('darkcolors' => array('blue','black')),
    'erase' => array (
      'darkcolors' => array (
        'attribute1',
        'attribute2',
      ),
      'yellow' => array (
        'attribute3',
        'attribute4',
      ),
    )
  )
);
```

Note the **multiplevalues** special key in order to specify several values that disable the same attributes.

3.4 plInfo

plInfo is a static function that must be present on all plugin classes that want to appear in FusionDirectory in one of these ways:

- In FusionDirectory main menu: use plSection
- In an object tabs: use plObjectType
- In ACL settings: use plProvidedAcls

This static method returns an array containing keys from the following table. Some may be safely omitted. Do not fill a key if you intend to use the default value for it. Use translated strings (use the _() function) for all displayable strings.

Table 1: plInfo keys

key	value	used in	default
plShortName	A short name for this class	Used in main menu and ACL summaries or such	**mandatory**
plTitle	A short title for this class	Used in FD page header at top of the page if this plugin have its own page	plShortName value
plDescription	A short description	Used as tooltip in the menus and description in ACL management	**mandatory**
plIcon	A big icon (48x48)	Used for main menu and header at top of the page	no icon
plSmallIcon	A small icon (16x16)	Used in listings	empty icon
plSelfModify	TRUE for user tabs, omitted for others	Define if it should appear in «my account» menu section for logged in users	FALSE
plPriority	an integer (safe to omit most of the time)	Defines tab order and menu items order	no prority (at the end, in no specific order)
plDepends	Array of tabs this tab depends on	Listed tabs will need to be activated before this one can be used	empty
plCategory	Array of ACL categories	Should be omitted if you've listed objectTypes	categories of objectTypes and managed objects
plObjectType	Array of objectTypes	See full documentation below	empty
plSection	menu section key or array (see below)	See full documentation below	empty
plProvidedAcls	Array of acls	See full documentation below	empty
plForeignKeys	Array of foreign keys	See full documentation below	empty
plManages	Array of managed objectTypes (only for management classes)	Used to create links to objects of these types	empty

3.4.1 plSection

plSection can be used if your plugin should appear in the menu. Default menu sections are "*accounts*", "*systems*", "*conf*", "*reporting*" and "*personal*". Usually the **plSection** is set on the management class if there is any. No need to set any **plSection** on plugins with objectType *user* and selfModify TRUE, they'll appear in the 'My account' section anyway.

You can also create a new menu section in this attribute using the following syntax:

```
<?php
array('mysection' => array('name' => _('My section'), 'priority' => 100))
```

Replace *mysection* with a lowercase id for your section and *My section* with the name to display in the menu.

The existing sections are:

Table 2: Menu sections

key	name	priority
accounts	Users and groups	0
systems	Systems	10
conf	Configuration	20
reporting	Reporting	30
personal	My account	40

So you can for instance use a priority between 0 and 10 create a section between *accounts* and *systems*.

3.4.2 plObjectType

plObjectType is used to know which object type should have this plugin in its tabs. If this tab is the main tab of a new objectType, **plObjectType** must contain the definition for this object type.

ObjectType definition is an array containing the following keys:

Table 3: ObjectType properties

key	value	default
name	Displayable name for this object type	**mandatory**
description	Displayable description for this object type	**mandatory**
filter	LDAP filter to find objects of this type	**mandatory**
mainAttr	LDAP attribute to use in dn	cn
nameAttr	LDAP attribute to use in object links	*mainAttr*
tabClass	PHP class to use for tab handling	simpleTabs
icon	Small icon (16x16)	no icon
ou	RDN for the LDAP branch to store these objets in	empty string
aclCategory	The ACL category this objectType is in	*key*

aclCategory should be the name of an existing ACL category. Most of the time omit this and a category will automagically be created for you.

For instance, this is the plObjectType of the user class:

```
<?php
'plObjectType'  => array(
  'user' => array(
```

(continues on next page)

(continued from previous page)

```
    'description' => _('Users'),
    'name'        => _('User'),
    'filter'      => 'objectClass=gosaAccount',
    'mainAttr'    => 'cn',
    'icon'        => 'geticon.php?context=types&amp;icon=user&amp;size=16',
    'ou'          => get_ou('userRDN'),
  )
),
```

3.4.3 plForeignKeys

plForeignKeys is to be used if some of your fields are foreign keys to fields of other objects. For instance the manager field in a department is a foreign key on the dn of a user.

The syntax for this is:

```
<?php
'plForeignKeys'  => array(
  'myfield' => array(
    array('class', 'hisfield', 'filter'),
  )
)
```

But you can omit *filter* most of the time (defaults to '*myfield*=%oldvalue%') and *hisfield* if it is the *dn*, and if there is only one field you are referring to you can omit the array, so for our department example this gives us:

```
<?php
'plForeignKeys'  => array(
  'manager' => 'user'
)
```

Which is pretty straight forward.

Declaring a foreignKey ensure you that:

- If the referred field is modified through FD your object will be updated as well
- If the referred object is deleted your field will be emptied if possible (or the specific value referring the object will be removed in case of multi-value attributes)
- Your objects will appear in the references tab of referenced objects

3.4.4 plCategory

ACL categories will be filled automagically if you use either **plManages** or **plObjectType**. This is the recommanded way to go. If you do need to specify ACL categories, you can create an acl category by specifying a descriptive array for it:

```
<?php
'plCategory' => array(
    'acl' => array(
        'description'  => _('ACL'),
        'objectClass'  => array('gosaAcl','gosaRole')
    )
 ),
```

An ACL category only contains a description and a list of LDAP objectClasses (for some historical reason)

3.5 Attributes Types

Here is a detailed description about each available attribute type for simple plugin.

3.5.1 StringAttribute

The most simple class, allows to handle an LDAP attribute which is a string. Specific constructor parameter : as last parameter you can pass a pattern that the string must match in order to be valid

There are some attributes that are just like this one but with specific check for validity : MailAttribute, HostNameAttribute, IPv4Attribute, IPv6Attribute, MacAddressAttribute, IPAttribute (accepts both v4 and v6)

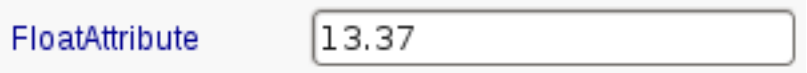
3.5.2 PasswordAttribute

Same thing as StringAttribute but input form is a hidden password input (html password input type). No specific constructor parameters

3.5.3 IntAttribute

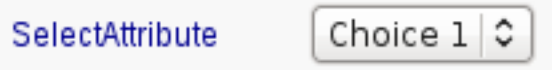
Allow to handle an int. Specific constructor parameters : min and max. Use “FALSE” to disable either one of them. “intval” will be used on user input in order to convert it. html5 “number” input type is displayed.

3.5.4 FloatAttribute



The same that IntAttribute but for floats. “floatval” will be used.

3.5.5 SelectAttribute



Specific constructor attributes : an array containing the available choices. An html select is displayed.

3.5.6 BooleanAttribute

or

Allow to handle booleans. No specific constructor parameters. A checkbox is displayed.

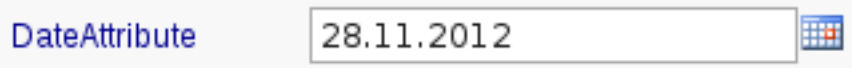
3.5.7 ObjectClassBooleanAttribute

Special kind of boolean that adds or removes object classes from the object.

3.5.8 FileAttribute

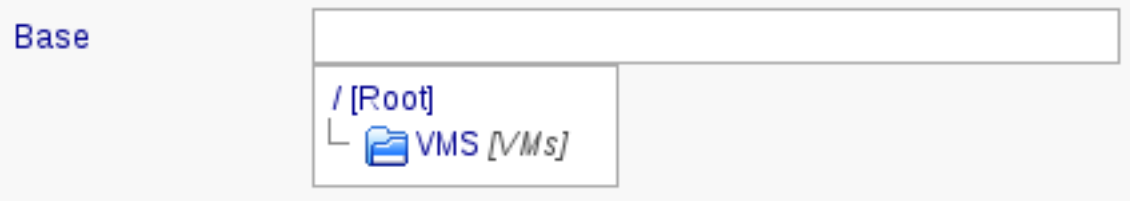
Allow the user to upload a file, store the file content in the LDAP. If you need to do something else with the uploaded file, you'll have to inherit this class and the readFile function.

3.5.9 DateAttribute



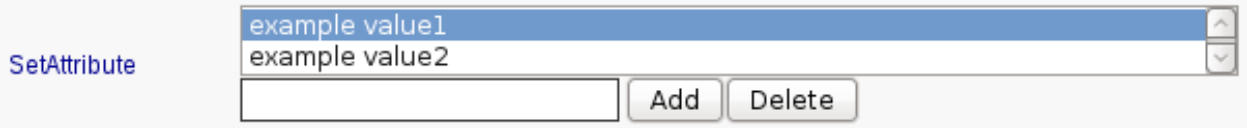
Show a text input with a calendar in order for the user to choose a date. You need to pass the wanted date format in LDAP in its constructor

3.5.10 BaseSelectorAttribute



Allow the user to select the base of the object. Usually in the main tab of most objects.

3.5.11 ArrayAttribute and SetAttribute



Allow to handle a multi-valuated attribute. The constructor takes only two parameters:

- An attribute, which is one of the above.
- An array of default values.

A multiple select will be used for displaying values, with remove and add buttons. SetAttribute is the same, but does not allow several identical values.

3.5.12 CompositeAttribute

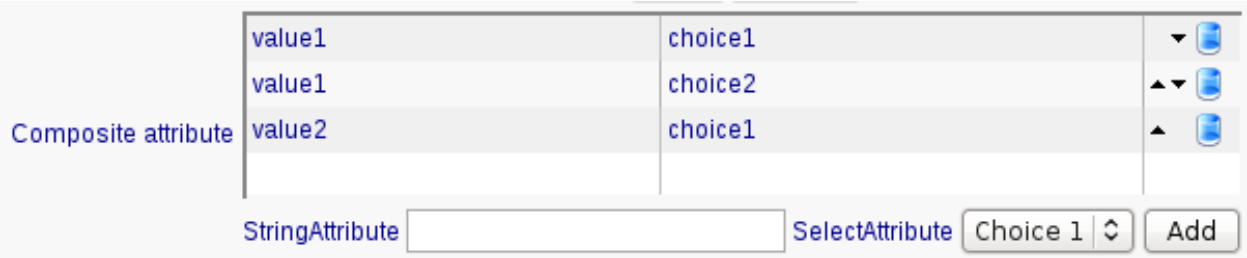
Allow to handle several UI attributes which are stored as only one LDAP field. For instance let's say you store an FTP connection URL in an LDAP field as "ftp://user:password@host:port" but you want to display 4 inputs for the 4 parts. That would look like :

```
<?php
new CompositeAttribute (
  _('Informations for ftp login'),
  'ftpLoginInfo',
  array(
    new StringAttribute (_('Login'),    _('Login for FTP'),     'ftpLogin'),
    new StringAttribute (_('Password'), _('Password for FTP'),  'ftpPassword'),
    new StringAttribute (_('Host'),     _('Host for FTP'),      'ftpHost'),
    new IntAttribute    (_('Port'),     _('Port for FTP'),      'ftpPort', FALSE, 0,␣
↪FALSE, 21),
  ),
  'ftp://%[^@:]:%[^@:]@%[^@:]:%d', // sscanf format
  'ftp://%s:%s@%s:%d'              // sprintf format
)
```

(If you need something else than scanf and printf for composition, you have to inherit CompositeAttribute in a new attribute class and write your own readValues and writeValues functions)

3.5.13 OrderedArrayAttribute

This is an OrdreredArrayAttribute of CompositeAttribute (itself composed of a String and a Select attribute)



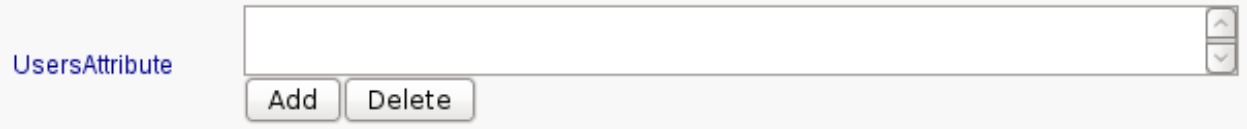
Like a SetAttribute, but shows values as a table with button for removing entries and changing order. It stores the order as "indice:value" in the LDAP. You can pass FALSE as second parameter to disable the ordering if you just want a SetAttribute that looks different.

3.5.14 UsersAttribute

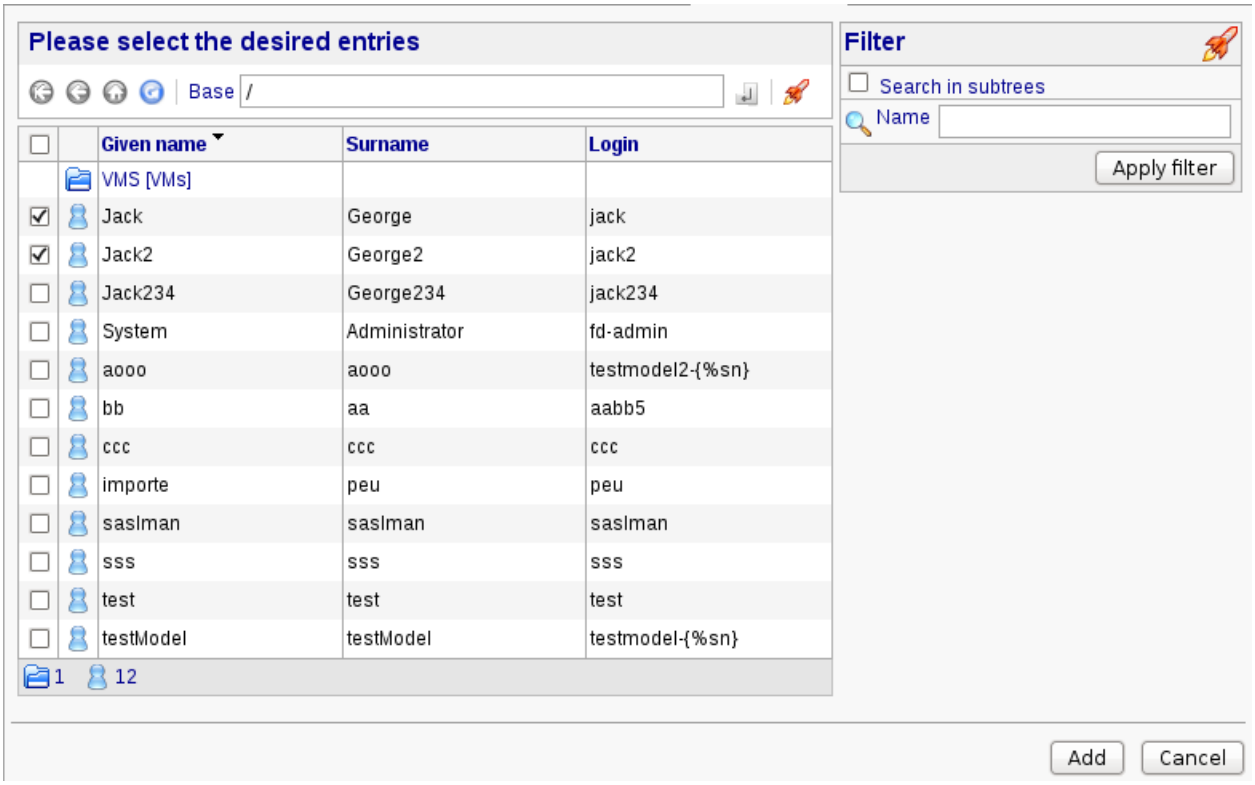
Allow the user to select a lists of users. Their dn are stored in the LDAP.

A dialog is available to add users:

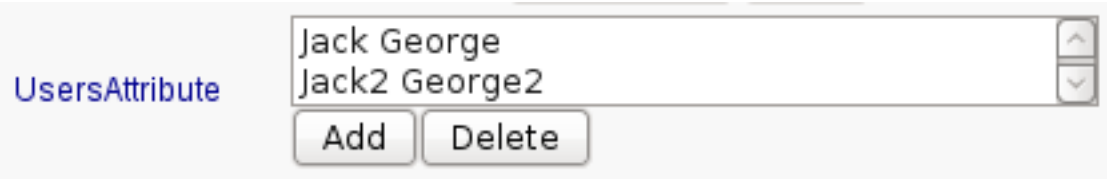
Before:



Selection:



After:



3.6 Configuration back-end

If your plugin needs to have some configuration stored into the LDAP and to appear in the configuration page accessible from the menu, you need to create another class for the configuration backend, inside your plugin.

3.6.1 Class for the configuration

You need to create a simplePlugin inheriting class that will have the objectType **'configuration'** if you want a whole tab for your plugin or simply **'smallConfig'** if you have only one or two sections that can be displayed with the other plugins in the plugins tab of the configuration.

3.6.2 LDAP storage for the configuration

To store your configuration options into the LDAP backend you will need to write your own schema. The options needs to have a name which starts by the prefix 'fd'.

They will be accessible in the PHP code using:

```
<?php
$config->get_cfg_value('option_name', default_value)
```

With *option_name* being the option name **without** the 'fd' prefix.

3.7 Menu sections

Here are the existing menu sections and examples of priorities they use, so that you can select which priority you give your menu entry.

3.7.1 Users and groups (accounts)

- Departments - 0-9
  - Departments - 0
- Users - 10-19
  - Users - 10
- Groups - 20-39
  - Groups - 20
  - (Object groups - 21)
  - NIS Netgroups - 25
  - Aliases - 26
  - ACL roles - 27
  - ACL assignments - 28
- Other - 40-99
  - Supann structures - 40
  - Sudo - 45
  - EJBCA - 50
  - DSA - 55
  - Password Policies - 57
  - Applications - 60

3.7.2 Systems (systems)

- Systems - 0-19
  - Systems - 0

  - DNS - 1
  - (DHCP - 2)
  - Auto fs - 5
- Deployment - 20-39
  - FAI - 20
  - Debconf - 21
  - Repository management - 22
  - OPSI - 25
  - Deployment queue - 30
- Other - 40-99
  - Samba domains - 40
  - SOGo - 50

3.7.3 Configuration (conf)

- Configuration - 0-9
  - Configuration - 0
  - (Password recovery - 1)
  - GPG server info - 5
- Import/export - 10-29
  - LDAP import/export - 10
  - OPSI import - 15

3.7.4 Reporting (reporting)

- Dashboard - 0
- Debug help - 1
- Inventory objects - 5
- Audit

CHAPTER 4

API

FusionDirectory

4.1 FusionDirectory API

FusionDirectory API is called simplePlugin and is documented automatically at each build.

FusionDirectory API

4.2 Fusiondirectory WebService

FusionDirectory WebService plugin exposes a JSONRPC webservice you can use if you want to access LDAP content through FusionDirectory system. This way, you ensure that things like foreign keys are kept consistent, and you have a nicer API than the low-level LDAP one.

It is a standard JSONRPC server served on HTTPS protocol.

Note that you can allow HTTP in plugin configuration, but please avoid doing so except for testing purposes.

The webservice methods are detailed here.

Basically you first need to call **login** to get a session ticket you'll use in the other method calls you make. If you have several LDAP configured you might call **listLdaps** first to list them and specify which one to use as first parameter of **login** (otherwise just pass NULL as first parameter).

Then you can use **ls** to list objects of a given type (list types with **listTypes** first if needed). **getfields** method will give you the fields of a given type (and tab) and **setfields** will allow you to change the value of these fields.

```
<?php

/* You can find this file in FusionDirectory include directory if argonaut plugin is␣
↪installed */
require_once('jsonRPCClient.php');
```

(continues on next page)

(continued from previous page)

```
/* Connection information. Fill peer_name with the name matching the certificate. */
$host       = 'https://localhost/fusiondirectory/jsonrpc.php';
$ca_file    = '/etc/ssl/certs/fd.pem';
$login      = 'fd-admin';
$password   = 'adminpwd';

/* DN of an existing user we can display and modify */
$userdn     = 'uid=bilbo,dc=opensides,dc=be';

$ssl_options = array(
  'cafile'            => $ca_file,
  'peer_name'         => 'localhost',
  'verify_peer'       => TRUE,
  'verify_peer_name'  => TRUE,
);

$http_options = array(
  'timeout' => 10
);

try {
  /* We create the connection object */
  $client = new jsonRPCClient($host, $http_options, $ssl_options);

  /* Then we need to login. Here we log in the default LDAP */
  $session_id = $client->login(NULL, $login, $password);

  /* Once we have a session ID, we can ask for the list of users */
  $users  = $client->ls($session_id, 'user');

  foreach ($users as $dn => $user) {
    echo "$user ($dn)\n";
  }

  /* We can get a user's fields */
  /* Doing the same thing with NULL as dn would create a user, if you provide all␣
↪needed fields. */
  $fields = $client->getfields($session_id, 'user', $userdn);

  /* Change a value. We can pass an array for each tab, main one is 'user'.
    The array for each tab contains attribute ids and their new value.
    Attribute ids can be found in getfields, they are used as keys in the 'attrs'␣
↪array of each section.
  */
  $result = $client->setfields($session_id, 'user', $userdn,
  array(
    'user' => array('description' => 'Modified by webservice')
    )
  );

  if (isset($result['errors'])) {
    foreach($result['errors'] as $error) {
      print "Error: $error\n";
    }
  }
```

(continues on next page)

(continued from previous page)

```
} catch (jsonRPCClient_RequestErrorException $e) {
  die($e->getMessage());

} catch (jsonRPCClient_NetworkErrorException $e) {
  die($e->getMessage());
}
```

CHAPTER 5

LDAP schemas

If your plugin needs a new schema or stores data in config back-end, the following rules apply

5.1 Schema naming convention

Each plugin can have 2 schemas, one schema for the config backend named:

<plugin-name>-fd-conf.schema

and one for the plugin named:

<plugin>-fd.schema

5.2 LDAP number rules

This page is here to help you choose ID numbers for your attributeTypes objectClasses in your schemas. FusionDirectory project has '1.3.6.1.4.1.38414' prefix. There are three number after this prefix :

1. The first one should be the one attributed to your schema
2. The second one should start by 1 for attributeTypes and 2 for objectClasses
3. The third one should be incremented for each attributeType or objectClass

The important thing is the first one, the two others are up to you, these are just advices and rules we use in FD schemas.

5.2.1 Example

- attribueType 1.3.6.1.4.1.38414.**42.1.1**
- attribueType 1.3.6.1.4.1.38414.**42.1.2**
- attribueType 1.3.6.1.4.1.38414.**42.1.3**

- objectClass 1.3.6.1.4.1.38414.**42.2.1**
- objectClass 1.3.6.1.4.1.38414.**42.2.2**

Or, if you have two groups of attributeTypes that are somehow related:

- attribueType 1.3.6.1.4.1.38414.**42.10.1**
- attribueType 1.3.6.1.4.1.38414.**42.10.2**
- attribueType 1.3.6.1.4.1.38414.**42.11.1**
- attribueType 1.3.6.1.4.1.38414.**42.11.2**
- objectClass 1.3.6.1.4.1.38414.**42.2.1**
- objectClass 1.3.6.1.4.1.38414.**42.2.2**

5.2.2 Attribution

Schema	Number attributed
recovery-fd.schema	1
argonaut-fd.schema	2
quota-fd.schema	3
debconf-fd.schema	4
zimbra-fd.schema	5
goto.schema	6
puppet-fd.schema	7
core-fd-conf.schema	8
samba-fd-conf.schema	9
mail-fd.schema, mail-fd-conf.schema	10
alias-fd.schema, alias-fd-conf.schema	11
sympa-fd.schema	12
dsa-fd-conf.schema	13
cyrus-fd.schema	14
systems-fd.schema	16
supann-fd-conf.schema	17
system-fd-conf.schema	18
asterisk-fd-conf.schema	19
opsi-fd.schema	20
opsi-fd-conf.schema	21
netgroup-fd-conf.schema	22
sudo-fd-conf.schema	23
fax-fd-conf.schema	24
fai-fd-conf.schema	25
nagios-fd-conf.schema	26
board-fd-conf.schema	27
health-fd.schema	28 *reserved for Harmo*
ipmi-fd.schema	29
weblink-fd.schema	30
dovecot-fd.schema	31
sogo-fd-conf.schema	32
repository-fd.schema	33
repository-fd-conf.schema	34

Continued on next page

Table 1 – continued from previous page

Schema	Number attributed
gpg-fd.schema	35
ipmi-fd-conf.schema	36
desktop-fd-conf.schema	37
template-fd.schema	38
inventory-fd.schema	39
fusioninventory-fd.schema	40
fusioninventory-fd-conf.schema	41
voip-fd.schema	42
dns-fd-conf.schema	43
webservice-fd-conf.schema	44
ppolicy-fd-conf.schema	45
applications-fd.schema	46
ejbca-fd-conf.schema	47
personal-fd.schema	48
ejbca-fd.schema	49
personal-fd-conf.schema	50
dns-fd.schema	51
community-fd.schema	52
community-fd-conf.schema	53
subcontracting-fd.schema	54
newsletter-fd-conf.schema	55
newsletter-fd.schema	56
dhcp-fd-conf.schema	57
spamassassin-fd.schema	58
user-reminder-fd-conf.schema	59
audit-fd.schema	60
audit-fd-conf.schema	61
core-fd.schema	62
renater-partage-fd.schema	63
sympa-fd-conf.schema	64
sinaps-fd-conf.schema	65
supann-ext-fd.schema	66
public-forms-fd.schema	67
public-forms-fd-conf.schema	68
invitations-fd.schema	69
invitations-fd-conf.schema	70
seafile-fd.schema	71
seafile-fd-conf.schema	72 (temporary waiting for confirmation)

demoplugin.schema	1337
test-fd.schema	1338

GOsa legacy Schemas

Schema	Number attributed
core-fd.schema	1.3.6.1.4.1.10098.1.1.12
fai.schema	1.3.6.1.4.1.10098.1.1.5
mail-fd.schema	1.3.6.1.4.1.10098.1.1.12
proxy-fd.schema	1.3.6.1.4.1.10098.1.1.12
service-fd.schema	1.3.6.1.4.1.10098.1.1.9
system-fd.schema	1.3.6.1.4.1.10098.1.1.11

5.3 LDAP naming rules

When naming your LDAP objectClass and/or attributes, please follow these rules:

- Two first letter of your attribute or objectClass shoud be fd
- After that each letter that start a new word should be in uppercase
- Choose a meaningful name, that says what the attribute does
- If possible choose a first word that is common for all attributes of your objectClass
- Fill the attribute description in your LDAP schema

Also, remember to use the *LDAP number rules*

For plugins configuration objectClass, the following scheme shoud be used: fdNamePluginConf (for instance fdSystemsPluginConf)

Examples:

fdCyrusConnect

The description field should always start with “FusionDirectory - “

Example:

attributetype (1.3.6.1.4.1.38414.2.10.2 NAME ‘fdArgonautProtocol’
DESC ‘FusionDirectory - Argonaut, protocol.’
EQUALITY caseExactIA5Match
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
SINGLE-VALUE)

5.4 FusionDirectory reserved oid

PEN Assignment The Private Enterprise Number 38414 has been assigned to your organization.

CHAPTER 6

Themes

6.1 FusionDirectory theme system

A theme is defined by:

- A folder in *html/themes*
- A folder with the same name in *ihtml/themes*

The folder in *html/themes* contains:

- An index.theme file following the Icon Theme Specification
- The icons as described in the index.theme file, usually in an icons folder
- Replacements for any css file

The folder in ihtml should have the same name and should contain replacements for any template file.

6.1.1 Icon theme file

Here is the minimal index.theme file to inherit another icon theme.

```
[Icon Theme]
Name=MyTheme
Comment=Example from documentation
Inherits=oxygen
```

For an example of a more complex index.theme file look at the one of the default theme

All main icon themes should be working, you can activate them by using a symlink in the right folder. For instance on Debian if I want gnome icon theme:

```
$ ls -l /usr/share/fusiondirectory/html/themes/
drwxr-xr-x 4 root root 4096 Mar 16 10:24 breezy/
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x 4 root root 4096 Mar 16 10:24 legacy/
lrwxrwxrwx 1 root root   23 Jun 12  2014 gnome -> /usr/share/icons/gnome/
```

6.1.2 Replacements of css and tpl files

Any file you see in *html/themes/breezy* or *ihtml/themes/breezy* can be overridden by placing in your theme a file with the same name (css goes in *html*, tpl in *ihtml*).

Note that the file *html/themes/breezy/theme.css* is empty so that you can safely override it without losing anything from the default theme. Consider using it for theme making only small modifications.

CHAPTER 7

Translate

7.1 Translate FusionDirectory

We are using Transifex

- Create a transifex account
- Ask to be added to the language group you want to translate
- Start translating

CHAPTER 8

Release Policy

This section explain the release cycle and where contributions, fixes will be merged

8.1 FusionDirectory Version policy

8.1.1 Versioning

FusionDirectory can have 3 digits at maximum in a version : **X.Y.Z**

Z version increments (X.Y.Z1 –> X.Y.Z2, for example 1.2.1 to 1.2.2) are minor bug fix only releases.

Y or **X** version increments are major releases (X.Y1.Z -> X.Y2.Z, for exemple 1.1 to 1.2) are major releases.

8.1.2 Major Release

- Can contain any type of bugfix, new features and code refactor.
- Can remove attributes or objectclasses from the schema only if they were declared OBSOLETE in the previous major release.
- Can put **OBSOLETE** attributes and classes which are no longer used by the code.
- Two 2 major releases are needed before removing OBSOLETE attributes and objectClass.
- Can provide migration scripts in fusiondirectory-setup if needed for those, and/or migration instructions in the documentation.
- Have to provide migration instruction from previous major release.

8.1.3 Minor release

Minor release are small releases containing only bugfix to the last major release. It should be numbered with 3 digits.

Minor release **cannot** contain :

- Schema changes
- New features
- Code refactor
- Poorly tested code
- Changes which may break existing plugins or themes for previous release (or scripts based on the webservice)

Minor release contain :

- bugfix : should fix a bug observed in a previous release, something which did not work as intended.

Exceptions can be made :

- New feature can be included if it does not require any schema change and does not interfere with existing features
- Code refactor can be included if it leads to a significant performance gain and is thoroughly tested
- New plugin may be added if it does not require schema change (but it can add new schemas as this is non-intrusive)

Minor release must be released as soon as possible when :

- Security breach is found in the last stable release
- Regression (a bug which was not there in previous releases) is found in the last stable release
- Major bug is found in the last stable release

8.2 FusionDirectory Life Cycle

A maintained version is a **major** version for which we release **minor** bug fix releases and communicate about security vulnerabilities.

Our general support policy is to maintain major releases until **3 months** after the next major version is released to give some time for upgrading.

We also provide a more conservative life-cycle through **ESR** versions.

The **ESR** tag is given to a major release once it has been thoroughly tested and proven reliable on real production systems.

ESR versions are maintained until **6 months** after the next **ESR** version is announced, allowing less frequent major upgrades, and a larger time-frame for switching between them.

CHAPTER 9

Distribution and PHP support

9.1 Distribution and PHP support Policy

Distribution OSes have different interpretations of what a ‘supported version’ is, here are the OS and PHP versions FusionDirectory support.

9.1.1 Server OSes

- Debian: stable and oldstable
- Ubuntu: the two latest LTS releases
- Enterprise Linux (RHEL, CentOS, . . .): two latest major releases

9.2 PHP versions

The version of PHP depend on the FusionDirectory version.

Fusiondirectory need at least PHP 5.4.

- Fusiondirectory 1.2.3 need PHP 5.4
- Fusiondirectory 1.3 need PHP 5.6
- Fusiondirectory 1.4 need PHP 7.0

CHAPTER 10

License

10.1 License

FusionDirectory is available under the GNU General Public License 2.0

FusionDirectory documentation is under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

orphan