# Developper Documentation

These documentations are dedicated to people who want to contribute code to FusionDirectory

## Developping plugins

Configuring your client and your project on the forge

Writing a plugin for FusionDirectory using Simple Plugin

- Simple Plugin Attributes
- Going further with Simple Plugin
- FusionDirectory configuration backend
- FusionDirectory Main Menu

FusionDirectory coding guidelines

FusionDirectory Header and Copyright guidelines

FusionDirectory plugins organization

FusionDirectory version numbering

FusionDirectory API Stable

FusionDirectory API Dev

FusionDirectory schema naming convention

FusionDirectory schema number attribution

FusionDirectory schema attribute and object class naming

FusionDirectory oid declaration at iana

FusionDirectory theme system

## Argonaut

Follow this link

## Finding Fixed version of FusionDirectory that will become a .x release

## FusionDirectory 1.0.15 fixes

### Debian

Do like https://documentation.fusiondirectory.org/en/documentation/admin_installation/schema_installation but change the repository

```
# fusiondirectory repository jessie
deb
http://integration.opensides.be/repos/fixes-releases/debian/fusiondirectory-
115-fixes-jessie/ jessie main

# fusiondirectory repository wheezy
deb
http://integration.opensides.be/repos/fixes-releases/debian/fusiondirectory-
115-fixes-wheezy/ wheezy main
```

### RHEL 6

Do like https://documentation.fusiondirectory.org/en/dl_install#centos-6rhel-6 but change the baseurl

```
baseurl=http://integration.opensides.be/repos/fixes-releases/rpm/rhel/6/fusi
ondirectory-115-fixes/RPMS/
```

### RHEL 7

Do like https://documentation.fusiondirectory.org/en/dl_install#centos-7rhel-7 but change the baseurl

```
baseurl=http://integration.opensides.be/repos/fixes-releases/rpm/rhel/7/fusi
ondirectory-115-fixes/RPMS/
```

### SLES

Do like https://documentation.fusiondirectory.org/en/documentation/admin_installation_sles#install-fusiondirectory-repository but change the url

```
http://integration.opensides.be/repos/fixes-releases/rpm/SLES/fusiondirector
y-115-fixes/RPMS/
```

# Writing a plugin for FusionDirectory using Simple Plugin

Simple Plugin is the new way of writing plugin for FusionDirectory. If you're starting the development of a plugin now, you have to use it.

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

# 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).

## Icons

If your plugin packs some icons, they need to be placed in the default icon theme: *{fd-directory}/html/themes/default/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/default/icons/16/apps/apache.png* and is used in the code as *geticon.php?context=applications&icon=apache&size=16*

# 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

# 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 Simple Plugin Attributes

## Example

```
// 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 (
          _('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
        )
      )
    ),
  );
}
```

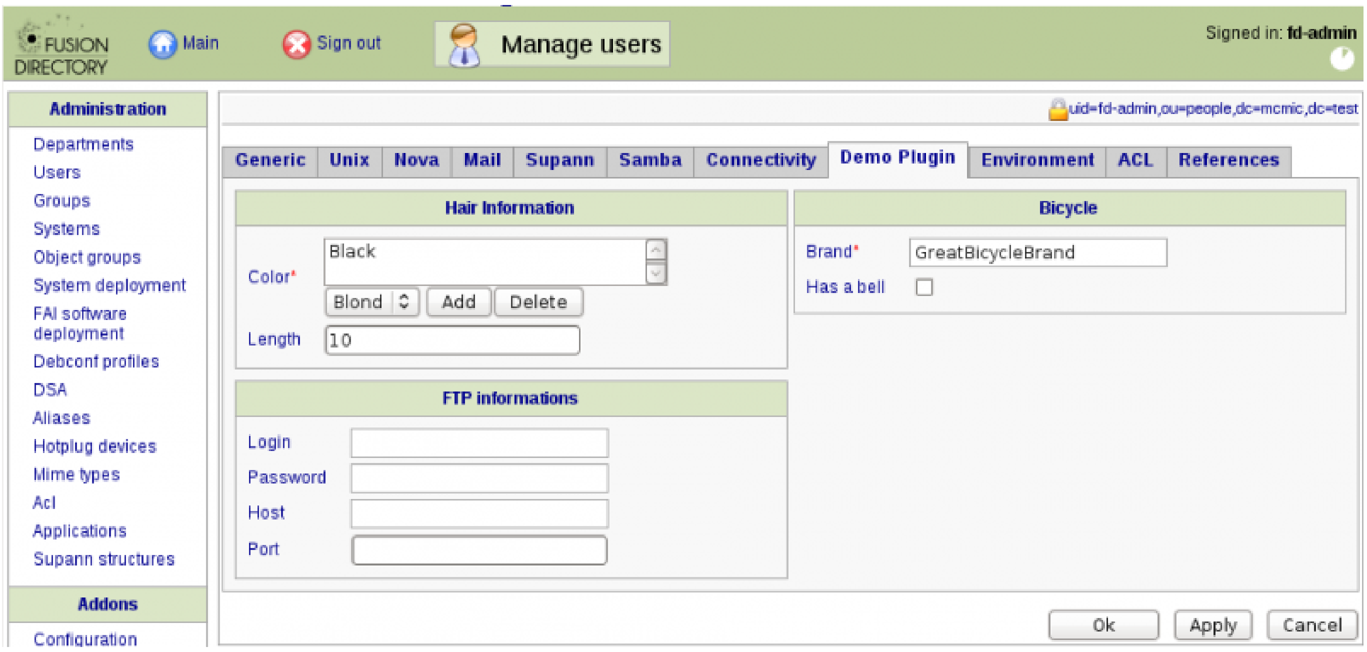
As you can see, attribute constructor take 5 arguments being label, description, ldap name, whether this attribute is mandatory or not, default value. Some attributes takes other arguments before and after the default value.
For each section you might also specify keys 'icon' with a section icon path, 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)

# 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 show in user edition mode, with the attributes we specified:



Hair Information

| | |
|---|---|
| Color* | Black |
| | Blond · Add · Delete |
| Length | 10 |

Bicycle

| | |
|---|---|
| Brand* | GreatBicycleBrand |
| Has a bell | ☐ |

FTP informations

| | |
|---|---|
| Login | |
| Password | |
| Host | |
| Port | |

## 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);
?>
```

## Going further

Going further with Simple Plugin

# 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.

# 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.

# 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:

```
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:

```
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:

```
function ldap_save($cleanup = TRUE)
{
  parent::ldap_save($cleanup);
```

```
  // your code goes here
}
```

## 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.

```
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 4th 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:

```
function __construct(&$config, $dn = NULL, $object = NULL)
{
  $attributesInfo = self::getAttributesInfo();
  // some modifications on $attributesInfo
  parent::__construct($config, $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:

```
function __construct(&$config, $dn = NULL, $object = NULL)
{
  parent::__construct($config, $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.

## 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).

## 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.

# Configuration back-end

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

## Programming a plugin for the FusionDirectory config backend

You need to create a simplePlugin inheriting class that will have the objectType **'configuration'** if you want a whole config 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.

## Ldap storage for the configuration backend

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

```
$this->config->get_cfg_value('option_name', default_value)
```

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

## Directory hirearchy for a FusionDirectory config backend plugin

## FusionDirectory Main menu

Real sections and fake subsections (subsections are indexing indication to support future extension)

### Users and groups

#### 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

### 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

### Configuration

#### Configuration - 0-9

- Configuration - 0
- (Password recovery - 1)
- GPG server info - 5

#### Import/export - 10-29

- LDAP import/export - 10
- OPSI import - 15

### Reporting

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

# FusionDirectory coding guidelines

## 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.

## Indentation and line length

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

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

```
set expandtab
set shiftwidth=2
set softtabstop=2
set tabstop=2
```

The line length should not exceed 80 characters. There is one exception for i18n strings that must not be split for gettext.

## 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.

## Comments

Avoid perl style comments using "#". Always use "//" for single line comments and /* */ blocks for multi line comments.

```
/*
 * This is a long comment...
 * ... which should look like this.
 */

// Short comment
```

## File format

Use

```
UTF-8, LF
```

instead of

```
CR LF
```

## White spaces

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

```
# Methods
foo($parameter);

# Arrays
$b = $value[0];

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

For vars declaration, place values on the same column

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

Always use spaces after comma to separate arguments:

```
function foo ($param1, $param2)
```

Always use single spaces to split logical and mathematical operations:

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

## Braces

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

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

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

Switches are formatted like this:

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

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

Always use braces for single line blocks:

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

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

```
function bar ()
{
  ...
}
```

## Casing

Always use camel casing with lowercase characters in the beginning for multi-word identifiers:

```
function checkForValidity ()
{
  $testSucceeded = false;
  ...
}
```

## Naming

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

```
// 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.

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

Use uppercase for constants/defines and _ if possible:

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

## PHP specific

Use return without parenthesis

```
return TRUE; // good

return(TRUE); // bad
```

Open and close tags:

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

### HTML

Do not include HTML code inside of your PHP file.

Use smarty templating if possible.

### Code inclusion

Use

```
require_once
```

instead of

```
include.
```

## GNU Indent command

Using the following

```
indent -bap -bad -blf -br -cdw -ce -nut -npcs -saf -sai -saw -ts2
```

Might be a good start, but be careful as it breaks (at least) “<?php” tag and elseif keyword.

# 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

```
This code is part of FusionDirectory (http://www.fusiondirectory.org/)
Copyright (C) 2011  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,
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.
```

Behind the copyright you can also if you want put a copyright with your name, like this

```
 This code is part of FusionDirectory (http://www.fusiondirectory.org)
 Copyright (C) 2011 Alejandro Escanero Blanco (aescanero@gmail.com)
```

# FusionDirectory plugins organization

The directories in a FusionDirectory plugins are standardized, it looks like this :

```
drwxr-xr-x  3 benoit benoit  80 19 oct 18:56 addons
drwxr-xr-x  4 benoit benoit 104 19 oct 18:56 admin
drwxr-xr-x  2 benoit benoit  80 19 oct 18:56 config
drwxr-xr-x  2 benoit benoit  80 19 oct 18:56 contrib
drwxr-xr-x  3 benoit benoit 104 19 oct 18:56 html
drwxr-xr-x 14 benoit benoit 368 19 oct 18:56 locale
drwxr-xr-x  3 benoit benoit  80 19 oct 18:56 personal
```

**addons** = This dir is used if the plugin have an addons

**admin** = This dir is the main dir for all plugins going into the admin menu

**config** = This dir is the ldap configuration dir, used if the plugin need to store option in ldap

**contrib** = This dir is used to put all the contributed files like schema, docs, manpages etc..

**html** = This dir is used to put all the images

**locale** = This dir is used for localization of the plugin

**personal** = This dir is used when plugin is to be used to manage user properties

## Subdirectories in each of the main directories

**contrib** :

opendlap = Schemas for the openldap server

docs = Documentation how to use the plugin

**html** :

images = images for plugin (Only if they are not icons)

themes/breezy = icons for plugin

Newer plugin adds their icons as part of the default theme to use the icon theme specification and allow themes to change these icons.

For instance themes/breezy/icons/48/apps/myapp.png for a 48×48 icon for your application.

**locale** :

en = iso code for language

en = directory where to put the message file

# FusionDirectory Version Numbering

From now, version number should not exceed 3 digits, with the following rules:

- The third digit is incremented whenever a new version is released, bug fix or small features for instance.
- The second digit is incremented whenever a new version with notable new features is released.
- The first digit is incremented whenever a version include huge changes from last one or when second digit has exceeded 9

Note : the third digit can became a number if needed, meaning we could have a 1.1.11 version if 11 bugfixes version were needed for 1.1 before 1.2 could be released.

# Schema naming convention

Each plugin have minimum 2 schema, one schema for the config backend named

<plugin-name>-fd-conf.schema

and one for the plugin named

<plugin>-fd.schema

it can also have is own schemas if needed

# 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.

## 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***

## 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 | |

| Schema | Number attributed | |
|---|---|---|
| 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 | |
| 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 | |

| Schema | Number attributed | |
|---|---|---|
| **dhcp-fd-conf.schema** | 57 | |
| **spamassassin-fd.schema** | 58 | |
| **user-reminder-fd-conf.schema** | 59 | |
| **audit-fd.schema** | 60 | |
| **audit-fd-conf.schema** | 61 | |
| **demoplugin.schema** | 1337 | |
| **test-fd.schema** | 1338 | |

GOsa legacy Schema

| 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 |

# 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 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

DESC '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 )
```

# FusionDirectory Iana registration

PEN Assignment The Private Enterprise Number 38414 has been assigned to your organization. An email with this number has also been sent to the email address provided.

# Fusion directory theme system

A theme is defined by:

1. A folder in html/themes
2. A folder with the same name in ihtml/themes

The folder in html/themes contains:

1. A index.theme file following the Icon Theme Specification
2. The icons as described in the index.theme file, usually in an icons folder
3. Replacements for any css file

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

## 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: http://git.fusiondirectory.org/gitweb/?p=main/fusiondirectory.git;a=blob;f=html/themes/default/index.theme;h=186f91745acfd3dae98fc3f17dfecad410bde728;hb=refs/heads/master

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 default/
lrwxrwxrwx 1 root root   23 Jun 12  2014 gnome -> /usr/share/icons/gnome//
```

## Replacements of css and tpl files

Any file you see in html/themes/default or ihtml/themes/default 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/default/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.