Mandriva Management Console Documentation

Release 3.1.1

Jean-Philippe Braun <jpbraun@mandriva.com>, Jean Parpaillon <

October 21, 2014

Contents

Mandriva Management Console (MMC) is a framework used in Mandriva Directory Server and Pulse 2 projects that provides plugins for MMC.

If you plan to install *MDS* plugins or Pulse 2 plugins you first need to install and configure MMC (see section *Core*).

CHAPTER 1

Installation and configuration

1.1 Core

1.1.1 Introduction

The MMC (**Mandriva Management Console**) is made of two parts:

- An agent running on the machine to manage. We call it « MMC agent ». The agent exports to the network several plugins that allow to manage the machine. Of course, there can be multiple agents running on the network. The agent and its plugins are written in Python.
- A web interface, that talks to the agent(s) using XML-RPC. The interface is written in PHP, and use the scriptaculous and prototype frameworks to provide an AJAX experience across all major browsers including Internet Explorer 6.

In this document, we will first explain how to install and configure the MMC agent and the base plugins, and then how to install the web interface.

The MMC core provides 3 plugins:

- **base** : a plugin for managing users and groups in LDAP
- **ppolicy** : a plugin for managing user password policies
- **audit** : a framework for recording all operations done in the MMC interface

Note: Other plugins are available in the *Mandriva Directory Server* and *Pulse 2* projects.

These installations instructions are generic: this means they should work on most Linux distributions.

If you have any installation issues, please use the MDS users mailing list.

1.1.2 Installation

How to install the MMC (Mandriva Management Console) on a Linux distribution

Repositories configuration and packages installation

Mandriva users are lucky

... because Mandriva RPM packages for the MDS and the MMC are available.

Packages for Mandriva 2010.0, 2010.2, 2011.0 and Cooker are available on Mandriva official repositories. You will find an official mirror using the Mandriva mirror finder module.

You can also add the repositories with the following command:

```
urpmi.addmedia --distrib --mirrorlist '$MIRRORLIST'
```

To install the MMC base packages, just type:

```
# urpmi mmc-agent mmc-web-base python-mmc-base
```

Debian packages

For Debian Squeeze, add this in your sources.list:

```
deb http://mds.mandriva.org/pub/mds/debian squeeze main
```

For Debian Wheezy:

```
deb http://mds.mandriva.org/pub/mds/debian wheezy main
```

To install MMC base packages, just type:

```
# apt-get update
# apt-get install mmc-agent mmc-web-base python-mmc-base
```

Other packages

We also provide packages for other distribution trough OpenBuildSystem here :

- MMC core
- MDS plugins

Note: CentOS DAG repository

For some packages, you will need to add the DAG repository to yum. Create a file named `/etc/yum.repos.d/DAG.repo` containing:

```
# DAG Repository for RedHat Enterprise 4 / CentOS 4
[dag]
name=DAG Repository
baseurl = http://apt.sw.be/redhat/el$releasever/en/$basearch/dag
gpgkey=http://dag.wieers.com/packages/RPM-GPG-KEY.dag.txt
gpgcheck=1
enabled=0
```

Packages naming conventions

Here are the packages naming conventions:

- mmc-agent: the MMC agent package
- python-mmc-PLUGIN: MMC agent plugin
- mmc-web-PLUGIN: web interface plugin

Note: Sample configuration files

All MMC related sample configuration files are available in the python-mmc-base package, in the directory `/usr/share/doc/python-mmc-base/contrib/` or on our repository.

You will find there OpenLDAP, SAMBA and Postfix configuration files and also OpenLDAP schemas.

Installation from source tarball

Note: If you are using packages you can skip this part

Pre-requisites

This python modules are needed for MMC to run :

- python-twisted-web
- python-ldap
- pylibacl
- pyopenssl
- python-gobject

The MMC web interface is written in PHP4. Basically, you just need to install an Apache 2 server with PHP5 support.

The XML-RPC module of PHP is needed too.

Installation

Get the current tarball at this URL: ftp://mds.mandriva.org/pub/mmc-core/sources/current/

```
# tar xzvf mmc-core-x.y.z.tar.gz
# cd mmc-core-x.y.z
# ./configure --sysconfdir=/etc --localstatedir=/var
# make
# make install
# tar xzvf mds-x.y.z.tar.gz

If you want also MDS modules:

# cd mds-x.y.z
# ./configure --sysconfdir=/etc --localstatedir=/var
# make
# make install
```

The default $PREFIX for installation is `/usr/local`. You can change it on the `./configure` line by adding the option `--prefix=/usr` for example.

Here are how the files are installed:

- `$PREFIX/sbin/mmc-agent`: the MMC agent
- `$PREFIX/lib/mmc/`: helpers for some MMC plugins
- `/etc/mmc/`: all MMC configuration files. There files are sample files you will need to edit.

- `/etc/init.d/mmc-agent`: MMC agent init script
- `$PREFIX/lib/pythonX.Y/site-packages/mmc`: MMC Python libraries and plugins.
- `$PREFIX/lib/pythonX.Y/site-packages/mmc/plugins/`: MMC Python plugins
- `$PREFIX/share/mmc/`: all MMC web interface related files (PHP, images, ...l)
- `$PREFIX/share/mmc/modules/`: MMC web interface plugins
- `/etc/mmc/mmc.ini`: MMC web configuration file

LDAP server configuration

MMC currently supports OpenLDAP.

One LDAP schema called MMC schema is mandatory. This schema and others are available in the `/usr/share/doc/mmc/contrib/base/` directory provided by the python-mmc-base package.

Mandriva

The OpenLDAP configuration can be easily done using the `openldap-mandriva-dit-package`.

```
# urpmi openldap-mandriva-dit
...
# /usr/share/openldap/scripts/mandriva-dit-setup.sh
Please enter your DNS domain name [localdomain]:
mandriva.com
Administrator account
The administrator account for this directory is
uid=LDAP Admin,ou=System Accounts,dc=mandriva,dc=com
Please choose a password for this account:
New password: [type password]
Re-enter new password: [type password]
Summary
=======
Domain:        mandriva.com
LDAP suffix:   dc=mandriva,dc=com
Administrator: uid=LDAP Admin,ou=System Accounts,dc=mandriva,dc=com
Confirm? (Y/n)
Y
config file testing succeeded
Stopping ldap service
Finished, starting ldap service
Running /usr/bin/db_recover on /var/lib/ldap
remove /var/lib/ldap/alock
Starting slapd (ldap + ldaps): [  OK  ]
```

And you're done, the LDAP directory has been populated and the LDAP service has been started.

Some tweaks needs to be done to the LDAP configuration so that the LDAP service suits to the MDS.

First, copy the MMC LDAP schema you need to the LDAP schemas directory.

```
# cp /usr/share/doc/mmc/contrib/base/mmc.schema /etc/openldap/schema/
```

Then, add these line to the file `/etc/openldap/schema/local.schema`:

```
include /etc/openldap/schema/mmc.schema
```

Then, to avoid LDAP schemas conflicts, comment or remove these lines at the beginning of the file /etc/openldap/slapd.conf:

```
#include /usr/share/openldap/schema/misc.schema
#include /usr/share/openldap/schema/kolab.schema
#include /usr/share/openldap/schema/dnszone.schema
#include /usr/share/openldap/schema/dhcp.schema
```

Last, comment or remove these lines at the end of the file /etc/openldap/mandriva-dit-access.conf:

```
#access to dn.one="ou=People,dc=mandriva,dc=com"
#        attrs=@inetLocalMailRecipient,mail
#        by group.exact="cn=MTA Admins,ou=System Groups,dc=mandriva,dc=com" write
#        by * read
```

To check that the LDAP service configuration is right, run slaptest:

```
# slaptest
config file testing succeeded
```

Now you can restart the LDAP service:

```
# service ldap restart
Checking config file /etc/openldap/slapd.conf: [  OK  ]
Stopping slapd:                                [  OK  ]
Starting slapd (ldap + ldaps):                 [  OK  ]
```

Debian

When installing the slapd package, debconf allows you to configure the root DN of your LDAP directory, set the LDAP manager password and populate the directory. By default debconf will not ask you to configure the root DN, you can run dpkg-reconfigure for this. If you choose "mandriva.com" as your domain, the LDAP DN suffix will be "dc=mandriva,dc=com".

```
# dpkg-reconfigure slapd
```

After that you only need to include the mmc.schema in slapd configuration and you are done.

Note: Debian Squeeze and later

Debian's OpenLDAP uses its own database for storing its configuration. So there is no more slapd.conf. You can use the mmc-add-schema script to load new schema in the OpenLDAP configuration database:

```
# mmc-add-schema /usr/share/doc/mmc/contrib/base/mmc.schema /etc/ldap/schema/
Adding schema for inclusion: mmc... ok
```

You can also write a regular slapd.conf file like before, and issue the followind commands to convert the file in the new format:

```
# /etc/init.d/slapd stop
# rm -rf /etc/ldap/slapd.d/*
# slaptest -f /path/to/slapd.conf -F /etc/ldap/slapd.d
# chown -R openldap.openldap /etc/ldap/slapd.d
# /etc/init.d/slapd start
```

Other distributions

Note: OpenLDAP example configuration

You will find an example of OpenLDAP configuration in the directory `agent/contrib/ldap/` of the mmc-core tarball.

Note: Already existing directory

If you already have an OpenLDAP directory, all you need to do is to include the mmc.schema file.

Get the file `mmc.schema` from the `/usr/share/doc/mmc/contrib/base` directory, and copy it to `/etc/openldap/schema/` (or maybe `/etc/ldap/schema/`).

Include this schema in the OpenLDAP configuration, in `/etc/ldap/slapd.conf` (or maybe `/etc/openldap/slapd.conf`):

```
include /etc/openldap/schema/mmc.schema
```

This schema must be included after the `inetorgperson.schema` file.

In the OpenLDAP configuration file, we also define the LDAP DN suffix, the LDAP manager (rootdn) and its password (rootpw):

```
suffix          "dc=mandriva,dc=com"
rootdn          "cn=admin,dc=mandriva,dc=com"
rootpw          {SSHA}gqNR92aL44vUg8aoQ9wcZYzvUxMqU6/8
```

The SSHA password is computed using the slappasswd command:

```
# slappasswd -s secret
{SSHA}gqNR92aL44vUg8aoQ9wcZYzvUxMqU6/8
```

Once the OpenLDAP server is configured, the base LDAP directory architecture must be created. Create a file called `/tmp/ldap-init.ldif` containing:

```
dn: dc=mandriva,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
dc: mandriva
o: mandriva
dn: cn=admin,dc=mandriva,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP Administrator
userPassword: gqNR92aL44vUg8aoQ9wcZYzvUxMqU6/8
```

The userPassword field must be filled with the output of the slappasswd command. Now we inject the LDIF file into the directory:

```
# /etc/init.d/ldap stop
# slapadd -l /tmp/ldap-init.ldif
# chown -R ldap.ldap /var/lib/ldap (use the openldap user for your distribution)
# /etc/init.d/ldap start
```

Note: LDAP suffix

In this example, the LDAP suffix is dc=mandriva,dc=com. Of course, you can choose another suffix.

Note: Changing the OpenLDAP manager password

You can't change this password using the MMC interface. You must use this command line:

```
$ ldappasswd -s NewPassword -D "cn=admin,dc=mandriva,dc=com" -w OldPassword -x cn=admin,dc=mandriva,
```

NSS LDAP configuration

To use LDAP users and groups, the OS needs to know where to look in LDAP.

To do this, `/etc/nsswitch.conf` and `/etc/ldap.conf` (`/etc/libnss-ldap.conf` for Debian based distros) should be configured.

Note: On Debian install the package `libnss-ldap`

Your `/etc/nsswitch.conf` should look like this:

```
passwd:     files ldap
shadow:     files ldap
group:      files ldap
hosts:      files dns
bootparams: files
ethers:     files
netmasks:   files
networks:   files
protocols:  files
rpc:        files
services:   files
netgroup:   files
publickey:  files
automount:  files
aliases:    files
```

Your `/etc/ldap.conf`:

Note: On Debian wheezy the configuration is located in

/etc/libnss-ldap.conf

```
host 127.0.0.1
base dc=mandriva,dc=com
```

1.1.3 MMC configuration

Web interface configuration

For a full documentation of the `/etc/mmc/mmc.ini` file see *MMC web configuration file*.

What you may change in this file is:

- « login » and « password »: these are the credentials to connect to the MMC agents on your network (the same credentials as in `/etc/mmc/agent/config.ini`)
- « url » option of the [server_x]: the URL to connect to the MMC agent.

To connect to the MMC web interface using an URL like http://IP/mmc, we add an alias to Apache 2:

```
# cp /etc/mmc/apache/mmc.conf /etc/httpd/conf.d/mmc.conf
```

or on Debian:

```
# cp /etc/mmc/apache/mmc.conf /etc/apache2/conf.d/mmc.conf
```

Then don't forget to reload the Apache service.

Now you should be able to see the MMC login screen at this URL: http://IP/mmc

---

Note: PHP configuration notes

The directive magic_quotes_gpc must be enabled in Apache PHP configuration, either in the global PHP configuration file, either in the `mmc.conf` file with this line:

```
php_flag magic_quotes_gpc on
```

The MMC web interface is not compatible with php-eaccelerator. Please uninstall it else you won't be able to connect to the MMC.

MMC agent configuration

For a full description of the MMC agent configuration file see *MMC agent configuration file*.

With the default configuration file we provide (`/etc/mmc/agent/config.ini`), the MMC agent listen locally to incoming XMLRPC over HTTPS connections on port 7080.

MMC « base » plugin configuration

For a full description of the MMC base plugin configuration file see *MMC base plugin configuration file*.

The main part of the configuration (`/etc/mmc/plugins/base.ini`) is to set the LDAP server to connect to, and the credentials to use to write into the LDAP. Check the following options:

- `ldapurl` : usually `ldap://127.0.0.1:389`
- `baseDN` : the rootdn of your LDAP directory
- `baseUsersDN` : DN to the `ou` containing LDAP users (eg: `ou=People, %(baseDN)`)
- `baseGroupsDN` : DN to the `ou` containing LDAP groups (eg: `ou=Group, %(baseDN)`)
- `rootName` : DN of the LDAP administrator
- `password` : password of the LDAP administrator

The `defaultUserGroup` option must be set to an existing group in the LDAP. You will have to create it using the MMC web interface if this group does not exist.

You need to create the directory specified in the `destpath` option.

About firewalling

The MMC web interface communicate with the MMC agent using the TCP port 7080 on localhost (default configuration). Please check that your firewall configuration doesn't block this port.

About SE Linux

The MMC web interface opens a socket to communicate with the MMC agent using XML-RPC.

On SE Linux enabled systems (e.g. Fedora Core 6), by default Apache can't open socket per policy. So you need to fix or disable your SE linux configuration to make it works.

1.1.4 Audit framework

Note: The configuration of the audit framework is optionnal

The MMC audit framework allows to record all users operations made through the MMC agent, and so the MMC web interface. These operations are all loggued: LDAP modifications, all filesystem related modifications, and service management (stop, start, ...)

The Python SQLAlchemy library version 0.5.x/0.6.x is required for the audit framework. The Python / MySQL bindings are also needed. On Debian install the following packages:

```
apt-get install python-mysqldb python-sqlalchemy
```

The audit framework is configured in the `base.ini` configuration file, and is disabled by default. To enable it, uncomment the audit section. It should look like:

```
[audit]
method = database
dbhost = 127.0.0.1
port = 3306
dbdriver = mysql
dbuser = audit
dbpassword = audit
dbname = audit
```

The **mmc-helper** tool will allow you to create the dabatase and to populate it with the audit tables easily.

To create the MySQL database:

```
# mmc-helper audit create
-- Execute the following lines into the MySQL client
CREATE DATABASE audit DEFAULT CHARSET utf8;
GRANT ALL PRIVILEGES ON audit.* TO 'audit'@localhost IDENTIFIED BY
'audit';
FLUSH PRIVILEGES;
```

Just execute the printed SQL statement in a MySQL client and the database will be created. Note that the `base.ini` is read to set the audit database name, user and password in the SQL statements.

On most Linux distribution, the "root" user has administrative access to the local MySQL server. So this one liner will often be enough:

```
# mmc-helper audit create | mysql
```

Once created, the audit database tables must be initialized with this command:

```
# mmc-helper audit init
INFO:root:Creating audit tables as requested
INFO:root:Using database schema version 2
INFO:root:Done
```

At the next start, the MMC agent will connect to the audit database and record operations.

1.1.5 Dashboard plugin

Note: The configuration of the dashboard plugin is optionnal

Installation

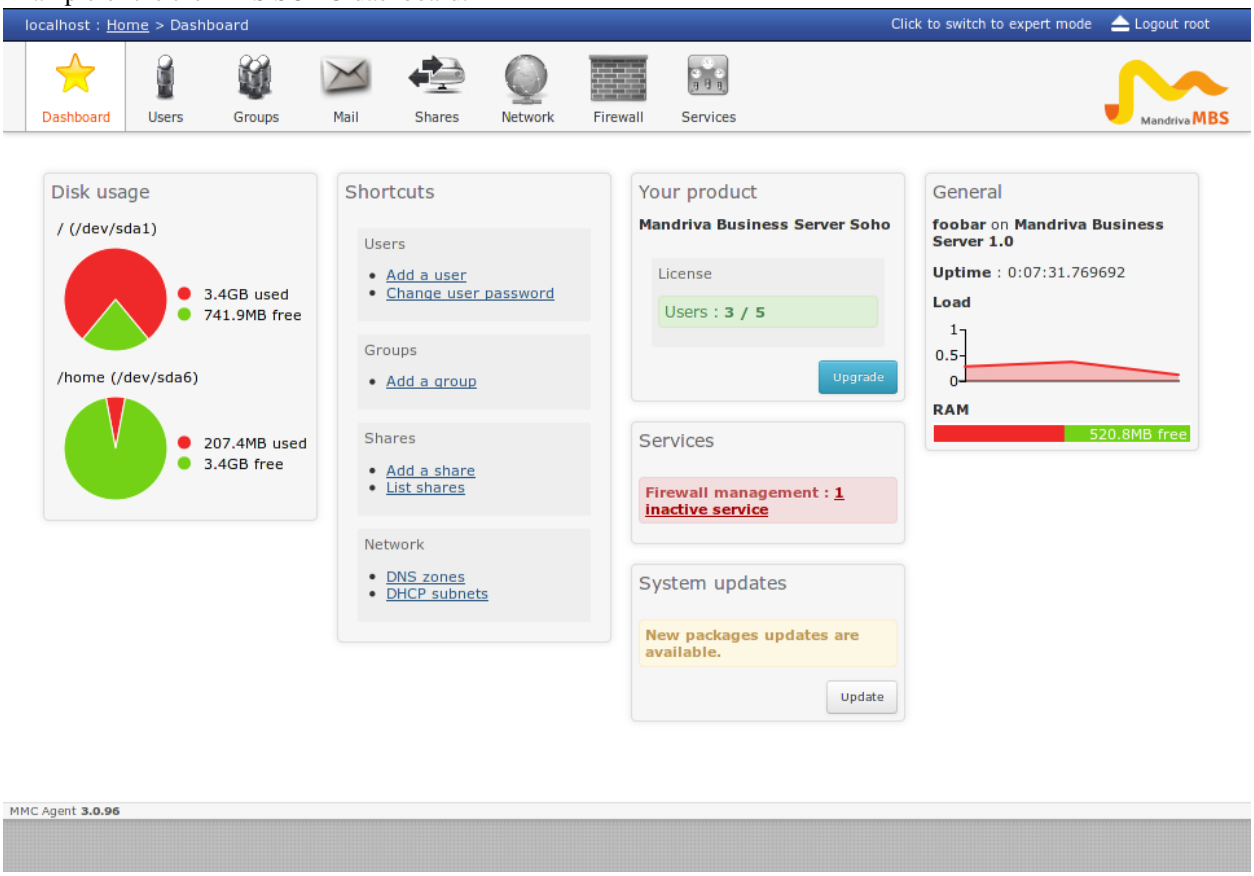
Install the packages `python-mmc-dashboard` and `mmc-web-dashboard`. Restart the `mmc-agent` service.

MMC « dashboard » plugin

The dashboard plugin will replace the legacy MMC-CORE home page with a page that can display panels from different MMC plugins.

Every MMC plugin can register its panels to the dashboard.

Example of the the MBS SOHO dashboard:



MMC « dashboard » plugin configuration

Like every MMC plugin the configuration can be found in `/etc/mmc/plugins/dashboard.ini`

The `disabled_panels` option can contain a list of panels that will be disabled. For example, to disable the shortcut and general panel:

```
disabled_panels = shortcut general
```

1.1.6 Password policy plugin

Note: The configuration of the ppolicy plugin is optionnal

Installation

Install the packages `python-mmc-ppolicy` and `mmc-web-ppolicy`.

OpenLDAP configuration for password policies

On Mandriva, if you used the mandriva-dit setup scripts, the password policy configuration is already done. If not, here are some instructions:

You must add this to your OpenLDAP `slapd.conf` configuration file:

```
# Include password policy schema
include /path/to/openldap/schema/ppolicy.schema
...
# Load the ppolicy module
moduleload  ppolicy
...
# Add the overlay ppolicy to your OpenLDAP database
database  bdb
suffix    "dc=mandriva,dc=com"
...
overlay ppolicy
ppolicy_default "cn=default,ou=Password Policies,dc=mandriva,dc=com"
```

Beware that the ppolicy_default value must match the options "ppolicyDN" and "ppolicyDefault" you set into the `ppolicy.ini` file.

MMC « ppolicy » plugin configuration

For a full description of the MMC ppolicy plugin configuration file see *MMC ppolicy (Password Policy) plugin configuration file*.

The only thing you'll have to modify in the configuration file is the "ppolicyDN" option if needed. The OU parent must be an existing DN. If the OU or the default password policy object doesn't exist, the MMC agent will create them when it starts.

Password Policy checker module

This module has only been built and tested on Mandriva and Debian. It is installed as `/usr/lib/openldap/mmc-check-password.so`.

If password quality checking is enabled on the password policy, OpenLDAP calls this module to check password quality when a user password is changed using the LDAP Password Modify Extended operation. MDS will change user passwords with this operation if you set "passwordscheme = passmod" in the `base.ini` configuration file.

To check a password, `mmc-check-password.so` will launch the command `/usr/bin/mmc-password-helper`. The password will pass the quality checks if it contains at least one number, one upper case character, one lower case character and one special character (like #, $, etc.). The password must not contains the same character twice. If python-cracklib is available, a cracklib check is also done.

The mmc-password-helper tool

This tool allows to check a password from the command line. For example:

```
% echo foo | mmc-password-helper -c
% echo $?
1
# Exit code is set to 1 if the password fails quality checks, else 0
# Use -v for more
# echo foo | mmc-password-helper -c -v
the password must be 8 or longer
% echo $?
1
```

The tool also generates good passwords:

```
% mmc-password-helper -n
1NjY0MD:
# Use -l to change the length (default is 8)
% mmc-password-helper -n -l 12
2ND=3OTcwMjY
% mmc-password-helper -n | mmc-password-helper -c
% echo $?
0
# Generated password will always succeed quality checks :)
```

Using password policies with SAMBA

If the SAMBA module is installed you can benefit of the LDAP password policies when a user changes his password from any Windows machine in the domain or via the MMC web interface.

Since SAMBA can't handle multiple password policies the MMC won't set any SAMBA password policies in the SAMBA domain ldap entry. But when SAMBA will try to change the user password in the LDAP, standard LDAP password policies applies.

The OpenLDAP password policies applies when the user password is changed with the "passmod" LDAP operation and when the user running the "passmod" is not the OpenLDAP rootdn.

If the MMC is binded to OpenLDAP with the rootdn as the administrator you will be able to change passwords from the MMC interface without any password policy checks. However, password poclicy is applied on the "change user password page" for normal users.

Note: Password synchronization

Usually the password synchronisation between the SAMBA password and the LDAP password is done by SAMBA itself. When a user changes his password SAMBA updates the sambaNTPassword attribute and run the "passmod" LDAP operation to change the userPassword attribute. This synchronization is done when `ldap sync password = yes` is set in SAMBA configuration. The problem with this method is that if the password does not pass the password policy check, the SAMBA password will be updated (as it is not changed by a "passmod" operation) but the userPassword attribute won't.

The second method to synchronize the password is to set `ldap sync password = only` in SAMBA configuration. In this case, SAMBA will only run the "passmod" LDAP operation when the user changes his password and won't update the `sambaNTPassword` attribute of the user. To update this attribute the OpenLDAP overlay `smbk5pwd` must be used. This overlay will intercept "passmod" operations and update the SAMBA password automatically only if the `userPassword` attribute has been updated successfully.

In conclusion, in order to use LDAP password policies with SAMBA you have to make sure that:

- SAMBA is not binded to OpenLDAP with the rootdn
- The `password scheme` option is set to "passmod" in `/etc/mmc/plugins/base.ini`
- Prefer using the `ldap sync password = only` method with the `smbk5pwd` overlay to make sure that passwords are always in sync (Shares -> General options -> Expert mode -> LDAP password sync)

The configuration of the smbk5pwd overlay is pretty forward. In your slapd.conf just add :

```
moduleload    smbk5pwd
[ ... ]
overlay smbk5pwd
smbk5pwd-enable samba
overlay ppolicy
ppolicy_default "cn=default,ou=Password Policies,dc=mandriva,dc=com"
[ ... ]
```

Note: The overlays order is important. Overlays will be called in the reverse order that they are defined. So ppolicy check must be done before smbk5pwd synchronization.

SAMBA domain policies

The SAMBA domain policies attributes are synchronized with the default OpenLDAP password policies by the MMC:

- pwdMinLength -> sambaMinPwdLength
- pwdMaxAge -> sambaMaxPwdAge
- pwdMinAge -> sambaMinPwdAge
- pwdInHistory -> sambaPwdHistoryLength
- pwdMaxFailure -> sambaLockoutThreshold
- pwdLockoutDuration -> sambaLockoutDuration

1.1.7 Services plugin

Note: The configuration of the services plugin is optionnal

Installation

Install the packages `python-mmc-services` and `mmc-web-services`.

> **Warning:** This plugin requires systemd.
> If systemd is not available the plugin won't be loaded.

MMC « services » plugin

This plugin allows the administrator to interact with the system services installed on the server. The plugin uses systemd DBUS interface to interact with services.

Currently you can start, stop, restart and reload services. You can also check any service log from the MMC interface.

MMC « services » plugin configuration

Like every MMC plugin the configuration can be found in `/etc/mmc/plugins/services.ini`

The plugin is disabled by default so you need to set `disable` to 0.

The plugin uses `journalctl` to display services logs in the interface. Check that the path to journalctl is correct for your system.

The `blacklist` option is used to hide any services in the interface. We don't display the OpenLDAP service because restarting it from the MMC is not reliable since the MMC depends on it.

1.1.8 Configuration files

MMC agent configuration file

This document explains the content of the MMC agent configuration file.

Introduction

The MMC agent is a XML-RPC server that exports to the network the API provided by the MMC python plugins.

Its configuration file is `/etc/mmc/agent/config.ini`. This file must be readable only by root, as it contains the login and password required to connect to the MMC agent.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

`/etc/mmc/agent/config.ini` available sections:

Section name	Description	Optional
main	MMC agent main option	no
daemon	MMC agent daemon option	no

All the other sections (loggers, handlers, ...) are related to Python language logging framework. See the Python documentation for more informations.

Section « main »

Available options for the "main" section:

Option name	Description	Optional	Default value
host	IP where the MMC agent XML-RPC server listens to incoming connections	No	
port	TCP/IP port where the MMC agent XML-RPC server listens to incoming connections	No	
login	login to connect to the MMC agent XML-RPC server	No	mmc
password	password to connect to the MMC agent XML-RPC server	No	s3cr3t
enablessl	Enable TLS/SSL for XMLRPC communication. If disabled, the XMLRPC traffic is not encrypted.	yes	0
verifypeer	If SSL is enabled and verifypeer is enabled, the XML-RPC client that connects to the MMC agent XML-RPC server must provide a valid certificate, else the connection will be closed.	yes	0
localcert	If verifypeer = 1, the file should contain the private key and the public certificate. This option was previously called privkey	If verifypeer = 1, yes	
cacert	Path to the file (PEM format) containing the public certificate of the Certificate Authority that produced the certificate defined by the localcert option. If verifypeer = 1, the certificate provided by the XML-RPC client will be validated by this CA.	If verifypeer = 1, yes	
session-timeout	Session timeout in seconds. When a user authenticates to the MMC agent, a user session in created. This session is destroyed automatically when no call is done before the session timeout is reach.	Yes	900
multithreading	Multi-threading support. If enabled, each incoming XML-RPC request is processed in a new thread.	Yes	1
maxthreads	If multi-threading is enabled, this setting defines the size of the pool of threads serving XML-RPC requests.	Yes	20
session-timeout	RPC Session timeout in seconds. If unset default to Twisted hardcoded 900 seconds.	yes	900

If `host=127.0.0.1`, the MMC agent will only listen to local incoming connections. You can use `host=0.0.0.0` to make it listen to all available network interfaces.

To connect to the MMC agent, the client (for example the MMC web interface) must do a HTTP Basic authentication, using the configured login and password.

You must change the login and password in the configuration file, because if you keep using the default configuration, anybody can connect to your MMC agent. MMC agent issue a warning if you use the default login and password.

Section « daemon »

This section defines some MMC agent daemon settings.

Available options for the "daemon" section

Option name	Description	Optional	Default value
user	System user under which the MMC agent service is running	yes	root
group	System group under which the MMC agent service is running	yes	root
umask	umask used by the MMC agent when creating files (log files for example)	yes	0777
pidfile	Path to the file containing the PID of the MMC agent	No	

If the MMC agent is configured to run as non-root, it drops its root privileges to the defined user and group id using the "seteuid" system call. This is done as soon as the configuration file is read.

Sections related to the Python logging module

See http://docs.python.org/lib/logging-config-fileformat.html.

In the default MMC agent configuration, two handlers are configured:

```
[handler_hand01]
class=FileHandler
level=INFO
formatter=form01
args=("/var/log/mmc/mmc-agent.log",)

[handler_hand02]
class=StreamHandler
level=DEBUG
args=(sys.stderr,)
```

The handler `hand01` records all logs emitted by the MMC agent (and its activated plugins) in the file `/var/log/mmc/mmc-agent.log`.

The handler `hand02` is used by the MMC agent only when it starts to display startup messages, then it is closed.

How to enable full debug in MMC agent

Just set `level=DEBUG` in hand01 handler (see previous section), and restart the MMC agent.

All the remote function calls and responses are now recorded in MMC log file.

MMC base plugin configuration file

This document explains the content of the MMC base plugin configuration file.

Introduction

The « base » plugin is the main plugin of the MMC Python API. It features base operations like LDAP management (users, groups, etc), user authentication and provisioning.

The plugin configuration file is `/etc/mmc/plugins/base.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Obfuscated password support in configuration files

All the passwords contained in MMC-related configuration files can be obfuscated using a base64 encoding. This is not a security feature, but at least somebody won't be able to read accidentally a password.

To obfuscate a password, for example the word "secret", you can use the Python interpreter:

```
$ python -c 'print "secret".encode("base64")'
c2VjcmV0
```

The base64-encoded password is the word "c2VjcmV0". Now to use it in a configuration file:

```
[section]
password = {base64}c2VjcmV0
```

The {base64} prefix warns the configuration parser that the following word is a base64-encoded word, and so needs to be decoded before being used.

Configuration file sections

Here are all the base.ini available sections:

Section name	Description	Optional
ldap	LDAP access definition	no
backup-tools	Backup tools configuration	no
audit	MMC audit framework configuration	yes
hooks	Hooks for scripts that interacts with the MMC	yes
userdefault	Attributes and Objectclass values that are added or deleted when adding a new user into the LDAP	yes
authentication	Defines how a user is authenticated when logging into the MMC web interface. For example, another LDAP server can be use to perform the authentication.	yes
provisioning	User accounts can be created or updated automatically when logging in to the MMC web interface.	yes
subscription	This section is used to declare what has been subscribed, and to give some important information to the end user.	yes

Section « ldap »

This section defines how the main LDAP is accessed, where are located the users organization units, etc.

Available options for the "ldap" section

Option name	Description	Optional	Default value
host (deprecated by ldapurl)	IP address or hostname of the LDAP server	no	
ldapurl	LDAP URL to connect to the LDAP server, for example: ldap://127.0.0.1:389. If ldapurl starts with "ldaps://", use LDAP over SSL on the LDAPS port. LDAPS is deprecated, and you should use StartTLS. If ldapverifypeer = demand, always use the server hostname instead of its IP address in the LDAP URL. This hostname must match the CN field of the server certificate.	no	
network_timeout	Network timeout in seconds for LDAP operations. No default timeout set.	yes	
start_tls	TLS connection parameters when LDAPS is not used. "off": never use TLS. "start_tls": use the LDAPv3 StartTLS extended operation (recommended).	yes	off
ldapverifypeer	If start_tls != off or LDAPS, specify check to perform on server certificate. "never": don't ask certificate. "demand": request certificate. If none or bad certificate provided, stop the connection (recommended).	yes	demand
cacertdir	Client certicates to use (default are empty) for LDAPS or TLS connections. For example: /etc/ssl/certs	yes	
cacert	Certificate Authority file. For example: /etc/mmc/certs/demoCA/cacert.pem	yes	
localcert	Local SSL certificate file. For example: /etc/mmc/certs/client.cert	yes	
localkey	Local SSL public key. For example: /etc/mmc/certs/client.key	yes	
ciphersuites	Accepted ciphers from the LDAP server.	yes	TLSv1:!NULL
ldapdebuglevel	set this to 255 to debug LDAP connection problems. Details of all LDAP operations will be written to stdout	yes	0
baseDN	LDAP base Distinguished Name (DN)	no	
rootName	LDAP administrator DN	no	
password	LDAP administrator password	no	
baseUsersDN	LDAP organisational unit DN where the users are located	no	
baseGroupsDN	LDAP organisational unit DN where the groups are located	no	
gpoDN	LDAP organisational unit DN where the GPO are located	yes	ou=System,baseDN
userHomeAction	If set to 1, create and delete user directory when creating/deleting one	no	
defaultUserGroup	When creating an user, set this group as the primary user group	yes	
skelDir	Use the specified directory when creating a user home directory	yes	/etc/skel
defaultHomeDir	Use this directory as a base directory when creating a user without specifying a home directory. If the creater user is called "foo", his/her homeDirectory will be "defaultHomeDir/foo"	yes	/home
defaultShellEnable	the default shell for enabled users	no	/bin/bash
defaultShellDisable	the default shell for disabled users	no	/bin/false
authorizedHomeDir	a list of directory where user home directory can be put	yes	defaultHomeDir
uidStart	starting uid number for user accounts	yes	10000
gidStart	starting gid number for groups	yes	10000
logViewModule	enable/disable the logview module	yes	no
logfile	LDAP log file path	no	
passwordscheme	LDAP user password scheme. Possible values are "ssha", "crypt" and "passmod". "passmod" uses the LDAP Password Modify Extended Operations to change password. The password encryption is done by the LDAP server.	no	passmod

Section « backup-tools »

This section defines where are located the backup tools. The backup tools are used when backuping a home directory or a SAMBA share from the MMC.

Available options for the "backup-tools" section:

Option name	Description	Optional
path	Where are located the executable needed by the backup tools	no
destpath	Where the backup are located once done	no

Section « audit »

This section defines the audit framework behaviour. By default the audit framework is disabled.

Available options for the "audit" section:

Option name	Description	Optional
method	Method used to record all audit data. Only the "database" method is supported.	no
dbhost	Host to connect to the SGBD that stores the audit database	no
dbdriver	Database driver to use. "mysql" and "postgres" drivers are supported.	no
dbport	Port to connect to the SGBD that stores the audit database.	no
dbuser	User login to connect to the SGBD that stores the audit database.	no
dbpassword	User password to connect to the SGBD that stores the audit database.	no
dbname	Name of the audit database.	no

Section « hooks »

The hooks system allow you to run external script when doing some operations with the MMC.

The script will be run as root user, with as only argument the path to a temporary file containing the full LDIF export of the LDAP user.

For the « adduser » and « changeuserpassword » hooks, the LDIF file will contain the userPassword attribute in cleartext.

For the « usertoken » hook the userPassword attribute will contain the authentication token for the user. This token is valid for 15 minutes. Using this token a link can be send to the user (email, sms...) so that he can login in the MMC interface and change his password trough the "Reset password page". The link is in the form: `http://SERVER/mmc/token.php?token=<TOKEN>`.

The executable bit must be set on the script to run. The temporary LDIF file is removed once the script has been executed.

Available options for the "hooks" section:

Option name	Description	Optional
adduser	path to the script launched when a user has been added into the LDAP directory	yes
changeuserpassword	path to the script launched when a user has been changed into the LDAP directory	yes
deluser	path to the script launched when a user is going to be removed from the LDAP directory	yes
usertoken	path to the script launched when an authentication token has been created for a user	yes

Here is a hook example written in BASH for « adduser »:

```
#!/bin/sh
# Get the uid of the new user
VALUE=`cat $1 | grep ^uid: | sed "s/uid: //"`
# Log new user event
echo "New user $VALUE created" >> /var/log/newuser.log
exit 0
```

The same hook, written in Python:

```
#!/usr/bin/env python
import sys
# ldif is a Python package of the python-ldap extension
import ldif
LOGFILE = "/var/log/newuser.log"

class MyParser(ldif.LDIFParser):

    def handle(self, dn, entry):
        uid = entry["uid"][0]
        f = file(LOGFILE, "a")
        f.write("New user %s created\\n" % uid)
        f.close()

parser = MyParser(file(sys.argv[1]))
parser.parse()
```

Section « userdefault »

This section allows to set default attributes to a user, or remove them, only at user creation.

Each option of this section is corresponding to the attribute you want to act on.

If you want to remove the « displayName » attribute of each newly added user:

```
[userdefault]
displayName = DELETE
```

Substitution is done on the value of an option if a string between '%' is found. For example, if you want that all new user have a default email address containing their uid:

```
[userdefault]
mail = %uid%@mandriva.com
```

If you want to add a value to a multi-valuated LDAP attribute, do this:

```
[userdefault]
objectClass = +corporateUser
```

Since version 1.1.0, you can add modifiers that interact with the substitution. This modifiers are put between square bracket at the beginning os the string to substitute.

Available modifiers for substitution

modifier character	Description
/	Remove diacritics (accents mark) from the substitution string
_	Set substitution string to lower case
|	Set substitution string to upper case

For example, you want that all new created users have a default mail address made this way: « firstname.lastname@mandriva.com ». But your user firstname/lastname have accent marks, that are forbidden for email address. You do it like this:

```
[userdefault]
mail = [_/]%givenName%.%sn%@mandriva.com
```

User authentication

The default configuration authenticates users using the LDAP directory specified in the [ldap] section.

But it is also possible to set up authentication using an external LDAP server.

Section « authentication » This optional section tells the MMC agent authentication manager how to authenticate a user. Each Python plugin can register "authenticator" objects to the authentication manager, that then can be used to authenticate users.

The authentication manager tries each authenticator with the supplied login and password until one successfully authenticate the user.

Please note that the user won't be able to log in to the MMC web interface if she/he doesn't have an account in the LDAP directory configured in the [ldap] section of the base plugin. The provisioning system will allow you to automatically create this account.

The base plugin registers two authenticators:

- baseldap: this authenticator uses the LDAP directory configured in the [ldap] section of the base plugin to authenticate the user,
- externalldap: this authenticator uses an external LDAP directory to authenticate the user.

Available options for the "authentication" section

Option name	Description	Optional	Default value
method	space-separated list of authenticators to try to authenticate a user	yes	baseldap

The default configuration is:

```
[authentication]
method = baseldap
```

authentication_baseldap This section defines some configuration directives for the baseldap authenticator.

Available options for the "authentication_baseldap" section:

Option name	Description	Optional	Default value
authonly	space-separated list of login that will be authentified using this authenticator. Others will be skipped.	yes	

For example, to make the "baseldap" authenticator only authenticate the virtual MMC "root" user:

```
[authentication_baseldap]
authonly = root
```

authentication_externalldap This section defines some configuration directives for the baseldap authenticator.

Available options for the "authentication_externalldap" section:

Option name	Description	Optional	Default value
exclude	space-separated list of login that won't be authenticated using this authenticator.	yes	
authonly	If set, only the logins from the specified space-separated list of login will be authenticated using this authenticator, other login will be skipped.	yes	
mandatory	Set whether this authenticator is mandatory. If it is mandatory and can't be validated during the mmc-agent activation phase, the mmc-agent exits with an error.	yes	True
network_timeout	LDAP connection timeout in seconds. If the LDAP connection failed after this timeout, we try the next LDAP server in the list or give up if it the last.	yes	
ldapurl	LDAP URL of the LDAP directory to connect to to authenticate user. You can specify multiple LDAP URLs, separated by spaces. Each LDAP server is tried until one successfully accepts a connection.	no	
suffix	DN of the LDAP directory where to search users	no	
bindname	DN of the LDAP directory account that must be used to bind to the LDAP directory and to perform the user search. If empty, an anonymous bind is done.	no	
bindpasswd	Password of the LDAP directory account given by the bindname option. Not needed if bindname is empty.	no	
filter	LDAP filter to use to search the user in the LDAP directory	yes	objectClass=*
attr	Name of the LDAP attribute that will allow to match a user entry with a LDAP search	no	

For example, to authenticate a user using an Active Directory:

```
[authentication_externalldap]
exclude = root
ldapurl = ldap://192.168.0.1:389
suffix = cn=Users,dc=adroot,dc=com
bindname = cn=Administrator, cn=Users, dc=adroot, dc=com
bindpasswd = s3cr3t
filter = objectClass=*
attr = cn
```

User provisioning

This feature allows to automatically create a user account if it does not already exist in the LDAP directory configured in the [ldap] section of the base plugin.

User provisioning is needed for example if an external LDAP is used to authenticate users. The users won't be able to log in to the MMC web interface even if their login and password are rights, because the local LDAP doesn't store thir accounts.

Section « provisioning » This optional section tells the MMC agent provisioning manager how to provision a user account. Each Python plugin can register "provisioner" objects to the provisioning manager, that then can be used to provision users.

When a user logs in to the MMC web interface, the authenticator manager authenticates this user. If the authentication succeed, then the provisioning manager runs each provisioner.

The authenticator object that successfully authenticates the user must pass to the provisioning manager the user informations, so that the provisioners have data to create or update the user entry.

Available options for the "provisioning" section

Option name	Description	Optional	Default value
method	space-separated list of provisioners	yes	

For example, this configuration tells to use the "externalldap" provisioner to create the user account:

```
[provisioning]
method = externalldap
```

provisioning_external This section defines some configuration directives for the externalldap authenticator.

Available options for the "authentication_externalldap" section:

Option name	Description	Optional	Default value
exclude	space-separated list of login that won't be provisioned by this provisioner.	yes	
ldap_uid	name of the external LDAP field that is corresponding to the local uid field. The uid LDAP attribute is the user login.	no	
ldap_givenName	name of the external LDAP field that is corresponding to the local givenName field	no	
ldap_sn	name of the external LDAP field that is corresponding to the local sn (SurName) field	no	
profile_attr	The ACLs fields of the user that logs in can be filled according to the value of an attribute from the external LDAP. This option should contain the field name.	yes	
profile_acl_<profilename>	The ACLs field of the user that logs in with the profile <profilename>.	yes	
profile_group_mapping	If enabled, users with the same profile will be put in the same users group.	yes	False
profile_group_prefix	If profile_group_mapping is enabled, the created groups name will be prefixed with the given string.	yes	

To create a user account, the MMC agent needs the user's login, password, given name and surname. That's why the `ldap_uid'È, ''ldap_givenName` and `ldap_sn` options are mandatory.

Here is a simple example of an authenticators and provisioners chain that authenticates users using an Active Directory, and create accounts:

```
[authentication]
method = baseldap externalldap

[authentication_externalldap]
exclude = root
ldapurl = ldap://192.168.0.1:389
suffix = cn=Users,dc=adroot,dc=com
bindname = cn=Administrator, cn=Users, dc=adroot, dc=com
bindpasswd = s3cr3t
filter = objectClass=*
attr = cn

[provisioning]
method = externalldap

[provisioning_externalldap]
exclude = root
ldap_uid = cn
```

```
ldap_givenName = sn
ldap_sn = sn
```

Subscription informations

This section contains all the information needed when the version is not a community one. It allow for example to send mail to the administrator directly from the GUI when something went wrong.

Available options for the "subscription" section:

Option name	Description	Optional	Default value
product_name	A combination of "MDS" and "Pulse 2" to describe the product	yes	MDS
vendor_name	The vendor's name	yes	Mandriva
vendor_mail	The vendor's email address	yes	sales@mandriva.com
customer_name	The customer's name	yes	
customer_mail	The customer's email address	yes	
comment	A comment on the customer	yes	
users	The number of users included in the subscription. 0 is for infinite.	yes	0
computers	The number of computers included in the subscription. 0 is for infinite.	yes	0
support_mail	The support's email address	yes	customer@customercare.mandriva.com
support_phone	The support's phone number	yes	0810 LINBOX
support_comment	A comment about the support	yes	

MMC ppolicy (Password Policy) plugin configuration file

This document explains the content of the MMC ppolicy (Password Policy) plugin configuration file

Introduction

The « ppolicy » plugin allows to set the default password policy to apply to all users contained into the LDAP directory, and to set a specific password policy to a user.

This plugin is disabled by default. Please be sure to understand how works password policy for LDAP before enabling it. Here are some related documentations:

- Internet-Draft: Password Policy for LDAP Directories
- Managing Password Policies in the Directory

The plugin configuration file is `/etc/mmc/plugins/ppolicy.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the ppolicy.ini available sections:

Section name	Description	Optional
main	global ppolicy plugin configuration	no
ppolicy		yes
ppolicyattributes		yes

Section « main »

This sections defines the global options of the mail plugin

Option name	Description	Optional	Default value
disable	Is this plugin disabled ?	Yes	1

Section « ppolicy »

This section defines the LDAP location of the password policies.

Option name	Description	Optional	Default value
ppolicyDN	DN of the LDAP OU where the default password policy will be stored	No	
ppolicyDefault	Name of the default password policy	No	

Section « ppolicyattributes »

This section defines the attributes and the values of the default LDAP password policy. The default policy will be initialized when the MMC agent starts if the default policy doesn't exist in the LDAP directory.

Of course the attribute name must match the LDAP password policy schema. Here is the default configuration we ship for this section:

```
# This options are used only once to create the default password
policy entry
# into the LDAP
[ppolicyattributes]
pwdAttribute = userPassword
pwdLockout = True
pwdMaxFailure = 5
pwdLockoutDuration = 900
# Password can be change if it not 7 days old
pwdMinAge = 25200
```

```
# Password expiration is 42 days
pwdMaxAge = 3628800
pwdMinLength = 8
pwdInHistory = 5
pwdMustChange = True
# To check password quality
pwdCheckModule = mmc-check-password.so
pwdCheckQuality = 2
```

MMC web configuration file

This document explains the content of the MMC web configuration file

Introduction

The MMC web interface communicates with MMC agents to manage LDAP directories, services and ressources.

Its configuration file is `/etc/mmc/mmc.ini`. This file must be readable only by the Apache web server, as it contains the login and password required to connect to MMC agents.

Like all MMC related configuration files, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

`/etc/mmc/mmc.ini` available sections:

Section name	Description	Optional
global	MMC web interface global options	no
debug	debug options	no
logintitle	Login page title	yes
server_x	MMC agent XMLRPC server connection options	no

Section « global »

Available options for the « global » section:

Option name	Description	Optional	Default value
backend	Which RPC backend to use. Only xmlrpc backend is available.	no	
login	credential to authenticate with the MMC agent	no	
password	credential to authenticate with the MMC agent	no	
root	Root URL where the MMC web pages are installed	no	
rootfsmodules	Filesystem path where the MMC web modules are installed	no	
maxperpage	Number of items (users, groups, ...) in displayed lists on the web interface	no	
community	It's a yes or no flag, it set the fact the installed version is a community one or not	yes	yes

Section « debug »

For debugging purpose only. The XML-RPC calls and results will be displayed on the MMC web interface.

Option name	Description	Optional	Default value
level	Wanted debug level. 0 to disable debug. 1 to enable debug.	No	

Section « logintitle »

This section allows to customize the title of the login box of the MMC web interface login page. By default, there is no title.

A title can be defined for each supported locales, like this:

```
localename = Title_for_this_locale
```

The title string must be encoded in UTF-8.

For example:

```
[logintitle]
; Default page title for English and non-translated languages
C = Welcome
; French title
fr_FR = Bienvenue
; Spanish title
es_ES = Bienvenido
```

Section « server_x »

You can set multiple sections called « server_01 », « server_02 » ... to specify a list of MMC agents to connect to.

On the MMC login web page, all the specified MMC agents will be displayed, and you will be able to select the one you want to be connected to.

Available options for the « server_x » sections:

Option name	Description	Optional	Default value
description	Label to display on the MMC login web page	no	
url	How to connect the XMLRPC server of this MMC agent	no	
forgotPassword	Show a "forgot password" link on the login page (See the usertoken hook in the base module configuration)	yes	no
timeout	Timeout in seconds for all socket I/O operations. Beware that timeout on a SSL socket only works with PHP >= 5.2.1.	yes	300
verifypeer	If verifypeer is enabled, the TLS protocol is used, and the XML-RPC server must provide a valid certificate.	yes	0
localcert	If verifypeer = 1, path to the file (PEM format) containing the private key and the public certificate used to authenticate with the MMC agent	no if verifypeer = 1	
cacert	Path to the file (PEM format) containing the public certificate of the Certificate Authority that produced the certificate defined by the localcert option. The certificate provided by the MMC agent will be validated by this CA.	no if verifypeer = 1	

For example, to define a local MMC agent:

```
[server_01]
description = Local MMC agent
url = http://127.0.0.1:7080
```

To use SSL between the web interface and the MMC agent (SSL must be enabled on the MMC agent):

```
[server_01]
description = Local MMC agent
url = https://127.0.0.1:7080
```

To use TLS with certificate check:

```
[server_01]
description = MMC agent
url = https://10.0.0.1:7080
verifypeer = 1
cacert = /etc/mmc/certs/demoCA/cacert.pem
localcert = /etc/mmc/certs/client.pem
```

1.1.9 Using MMC

Controlling mmc-agent

To start and stop the MMC agent, use the `/etc/init.d/mmc-agent` script:

```
# /etc/init.d/mmc-agent stop
# /etc/init.d/mmc-agent start
```

The MMC agent must be started to use the MMC web interface.

When the MMC agent is started, all startup log messages are written to stderr and `/var/log/mmc/mmc-agent.log`.

Here is what is written (for example) if there is no error:

```
# /etc/init.d/mmc-agent start
Starting Mandriva Management Console XML-RPC Agent: mmc-agent starting...
Plugin base loaded, API version: 4:0:0 build(82)
Plugin mail loaded, API version: 3:0:1 build(78)
Plugin samba loaded, API version: 3:0:2 build(78)
Plugin proxy loaded, API version: 1:0:0 build(78)
Daemon PID 13943
done.
```

If there is an error:

```
# /etc/init.d/mmc-agent start
Starting Mandriva Management Console XML-RPC Agent: mmc-agent starting...
Can't bind to LDAP: invalid credentials.
Plugin base not loaded.
MMC agent can't run without the base plugin. Exiting.
failed.
```

The base plugin can't bind to LDAP, because the credentials we used to connect to the LDAP server are wrong. As the base plugin must be activated to use the MMC agent, the MMC agent exits.

```
# /etc/init.d/mmc-agent start
Starting Mandriva Management Console XML-RPC Agent: mmc-agent starting...
Plugin base loaded, API version: 4:0:0 build(82)
Plugin mail loaded, API version: 3:0:1 build(78)
Samba schema are not included in LDAP directory
Plugin samba not loaded.
Plugin proxy loaded, API version: 1:0:0 build(78)
Daemon PID 14010
done.
```

In this example, the SAMBA schema has not been detected in the LDAP directory, so the SAMBA plugin is not started. But this plugin is not mandatory, so the MMC agent doesn't exit.

Administrator login to the MMC web interface

You can always login to the MMC web interface using the login « root » with the LDAP administrator password.

After you installed the MMC, this is the only user you can use to log in, because the LDAP directory entry is empty.

MMC agent and Python plugins inter-dependencies

When the MMC agent starts, it looks for all the installed plugins, and tries to activate them. Each plugin has a self-test function to check if it can be activated or not. For example, if the « base » plugin can't contact the LDAP, it won't be activated. It the SAMBA schema is not available in the LDAP, the « samba » plugin won't start.

The MMC agent always tries to enable the plugin « base » first. The MMC agent won't start if the plugin « base » can't be activated.

A MMC web module won't show in the web interface if the corresponding Python plugin is not loaded by the contacted MMC agent.

For example, you installed the SAMBA web module, but the SAMBA Python plugin of the MMC agent the web interface is connected to has not been activated. This will be detected and automatically the SAMBA management module of the web interface won't be displayed.

How to disable a plugin

In the .ini file corresponding to the plugin (in `/etc/mmc/plugins/`) , set « disable = 1 » in the main section.

1.2 Mandriva Directory Server

1.2.1 Introduction

The **Mandriva Directory Server** (MDS) provides different modules running on top of the **Mandriva Management Console**.

MDS is composed of the following plugins:

- **samba**: The « samba » plugin allows the MMC to add/remove SAMBA attributes to users and groups, to manage SAMBA share, etc.
- **network**: The « network » plugin allows the MMC Python API to manage DNS zones and hosts, DHCP subnet and hosts, into a LDAP. Patched version of ISC BIND (with LDAP sdb backend) and ISC DHCP (with LDAP configuration file backend) are needed. PowerDNS support is also available.
- **mail**: The « mail » plugin allows the MMC to add/remove mail delivery management attributes to users and groups, and mail virtual domains, mail aliases, etc. Zarafa support is also available.
- **sshlpk**: The « sshlpk » plugin allows the MMC to manage lists of SSH public keys on users.
- **userquota**: The « userquota » plugin allows the MMC to set filesystem quotas to users. The plugin provides LDAP attributes for storing quota information. The plugin allows also to store network quotas in the LDAP directory for external tools.
- **shorewall**: The « shorewall » plugin provides an interface to configure shorewall rules and policies from the MMC. Shorewall is wrapper around iptables [1].

Before installing MDS plugins, you have to install the Mandriva Management Console (see *Installation*).

1.2.2 Mail plugin

Installation

Install the packages `python-mmc-mail` and `mmc-web-mail`.

LDAP directory configuration

You need to import our mail schema into the LDAP directory. The schema file is provided by the `python-mmc-base` package in `/usr/share/doc/mmc/contrib/mail/mail.schema`.

Once this schema is imported, you will be able to manage mail delivery attributes thanks to the MMC.

Note: To include the schema on Debian:

```
mmc-add-schema /usr/share/doc/mmc/contrib/mail/mail.schema /etc/ldap/schema/
```

---

[1] http://shorewall.net/

Postfix/LDAP configuration

Example Postfix configuration files are included into the mds tarball and packages in */usr/share/doc/mmc/contrib/mail/postfix/*.

We provide two kinds of configuration:

- no-virtual-domain: the mail domain is fixed in the « mydestination » option in *main.cf* (you can't manage mail domains in the MMC - default mode)
- with-virtual-domains: mails are delivered to all mail domains created thanks to the MMC (you can add/remove mail domains from the MMC)

Copy all configuration files in */etc/postfix* and replace LDAP configuration values and domain name with your settings. In all *ldap-*.cf* files fix the search_base option. In *main.cf* fix the domain name in *myhostname* and *mydestination*.

NSS LDAP configuration

NSS LDAP configuration is needed to deliver mails with the right UIDs/GIDs.

See *NSS LDAP configuration*.

MMC « mail » plugin configuration

For a full description of the MMC mail plugin configuration file see *MMC mail plugin configuration file*.

This plugin won't be activated if your LDAP directory does not include a special mail schema.

To enable virtual domains set *vDomainSupport* to 1. To enable virtual aliases set *vAliasesSupport* to 1. To enable Zarafa support set *zarafa* to 1.

1.2.3 Network plugin

Introduction

This plugin allows to store in a LDAP directory:

- DNS zones declarations and related DNS records as needed for a standard LAN;
- DHCP server configuration with DHCP subnet, dynamic pool and static host declarations.

The MMC web interface allows to easily manage the DNS and DHCP services.

The network plugin relies on patched version of ISC DHCP 3 and ISC BIND 9:

- ISC BIND: a patch featuring a LDAP sdb backend must be applied to your BIND installation. With this patch BIND will be able to read DNS zone declarations from a LDAP directory. This patch is available there. The stable release of this patch (version 1.0) works fine.
- ISC DHCP: the patch on this page allows to store into a LDAP the DHCP service configuration (instead of `/etc/dhcp3/dhcpd.conf`).

Installation

Install the packages `python-mmc-network` and `mmc-web-network`.

Debian packages for patched versions of BIND

We provide Debian Lenny packages for the LDAP patched version of BIND. This packages work on Squeeze too.

Configure your APT repository as in the *Debian packages* section. And add in /etc/apt/preferences.d/pining :

```
Package: *
Pin: origin mds.mandriva.org
Pin-Priority: 1001
```

Then install the packages :

```
# apt-get update
# apt-get install bind9 isc-dhcp-server-ldap
```

DNS service configuration (ISC BIND)

When managing the DNS zones, the MMC agent will create files into the BIND configuration directory (located in `/etc/bind/`). These files must be included in the main BIND configuration file so that the corresponding zones are loaded from the LDAP directory.

All the DNS zones are defined in the file `named.conf.ldap`. This file must be included in the main BIND configuration file `named.conf`. Adding this line at the end of BIND `named.conf` should be sufficient:

```
include "/etc/bind/named.conf.ldap";
```

An example of `named.conf` filename for Debian based system is available at `/usr/share/doc/mmc/contrib/network/named.conf`.

Note: BIND and OpenLDAP services startup order

On most distributions, BIND is started before OpenLDAP during the boot sequence. If BIND/LDAP is used, BIND won't be able to connect to the LDAP directory, and won't start. So you may need to tweak your system boot scripts to fix this. The following command line should work on Debian based systems:

```
# update-rc.d -f slapd remove && update-rc.d slapd start 14 2 3 4 5 . stop 86 0 1 6 .
```

DHCP service configuration (ISC DHCP)

The DHCP server needs to know how to load its configuration from LDAP. Here is a typical `/etc/dhcp/dhcpd.conf`:

```
ldap-server "localhost";
ldap-port 389;
ldap-username "cn=admin, dc=mandriva, dc=com";
ldap-password "secret";
ldap-base-dn "dc=mandriva, dc=com";
ldap-method dynamic;
ldap-debug-file "/var/log/dhcp-ldap-startup.log";
```

The dhcpd service will try to find an LDAP entry for the machine hostname. If the entry name is different, you can set in `dhcpd.conf`:

```
ldap-dhcp-server-cn "DHCP_SERVER_NAME";
```

An example of `dhcpd.conf` filename is available in the directory `/usr/share/doc/mmc/contrib/network/`.

LDAP Schemas

Two new LDAP schemas must be imported into your LDAP directory: dnszone.schema and dhcp.schema.

Both are available in the directory `/usr/share/doc/mmc/contrib/network/`.

To speed up LDAP search, you can index these attributes: zoneName, relativeDomainName, dhcpHWAddress, dhcpClassData.

For OpenLDAP `slapd.conf` configuration file, you will add:

```
index zoneName,relativeDomainName eq
index dhcpHWAddress,dhcpClassData eq
```

MMC « network » plugin configuration

For a full description of the MMC network plugin configuration file see *MMC network plugin configuration file*.

You should verify that the paths to directories and init scripts are right.

MMC « network » plugin initialization

For the DHCP service only, the MMC network plugin needs to create into the LDAP directory two objects:

- the container called “DHCP config” (objectClass dhcpService), where all the DHCP service configuration will be stored
- the primary server (objectClass dhcpServer) that links to the DHCP service configuration. The hostname of the machine running the MMC network plugin will be use to name this entry.

The first start of the MMC network plugin should look like:

```
...
Created OU ou=DHCP,dc=mandriva,dc=com
Created DHCP config object
The server 'your_server_hostname' has been set as the primary DHCP server
Plugin network loaded ...
...
```

DHCP failover configuration

The DHCP failover can be done directly from the MMC interface on the page “Network -> Network services management”.

The primary DHCP server name is by default the hostname of the server where the mmc-agent is running. You can override this by setting the “hostname” option in `/etc/mmc/plugins/network.ini`

To configure DHCP failover you need at least the name of your secondary DHCP server and the IP addresses of the two DHCP servers. In expert mode you can set any parameter of the failover configuration.

The secondary ISC dhcpd configuration is almost the same as the primary DHCP:

```
ldap-server "LDAP_SERVER_IP";
ldap-port 389;
ldap-username "cn=admin, dc=mandriva, dc=com";
ldap-password "secret";
ldap-base-dn "dc=mandriva, dc=com";
ldap-dhcp-server-cn "SECONDARY_DHCP_SERVER_NAME";
```

```
ldap-method dynamic;
ldap-debug-file "/var/log/dhcp-ldap-startup.log";
```

1.2.4 SAMBA plugin

This document explains how to install the SAMBA plugin for MMC and its related configuration.

Installation

Install the packages `python-mmc-samba, mmc-web-samba` and `samba`.

SAMBA configuration for MMC

This section explains how to configure SAMBA with a LDAP directory so that it works with the MMC. Basically, you need to do a classic SAMBA/LDAP setup, SAMBA running as a PDC.

Note: Configuration files

A `slapd.conf` for OpenLDAP and a `smb.conf` for SAMBA can be found in */usr/share/doc/mmc/contrib/samba*.

Please use these files as templates for your own configuration.

If you aren't familiar with SAMBA/LDAP installation, read the SAMBA LDAP HOWTO. SAMBA LDAP setup is not easy.

LDAP directory configuration

You need to import the SAMBA schema into the LDAP directory. The schema file is provided by the `python-mmc-samba` package in `/usr/share/doc/mmc/contrib/samba/samba.schema`. But you can also use the schema provided by the SAMBA project.

SAMBA configuration

Stop samba before modifying its configuration:

```
# /etc/init.d/samba stop
Or according to your distribution:
# /etc/init.d/smb stop
```

In `/etc/samba/smb.conf`, you need to modify the « workgroup », « ldap admin dn » and « ldap suffix » to suit your configuration.

SAMBA also needs the credentials of the LDAP manager to write into the LDAP:

```
# smbpasswd -w secret
Setting stored password for "cn=admin,dc=mandriva,dc=com" in secrets.tdb
```

Now, SAMBA needs to create the SID for your workgroup:

```
# net getlocalsid MANDRIVA
SID for domain MANDRIVA is: S-1-5-21-128599351-419866736-2079179792
```

Use slapcat to check that the SID has really been recorded into the LDAP. You should find an entry like this:

```
# slapcat | grep sambaDomainName
dn: sambaDomainName=MANDRIVA,dc=mandriva,dc=com
...
```

Now you can start SAMBA:

```
# /etc/init.d/samba start
```

Populating the LDAP directory for SAMBA

The LDAP directory needs to be populated so that SAMBA can use it. We use the **smbldap-populate** command from the *smbldap-tools* package. This command populates the LDAP with the OUs (Organizational Unit), users and groups needed by SAMBA.

Note: On Debian do first:

cp /usr/share/doc/smbldap-tools/examples/smbldap_bind.conf /etc/smbldap-tools/ cp /usr/share/doc/smbldap-tools/examples/smbldap.conf.gz /etc/smbldap-tools/ gunzip /etc/smbldap-tools/smbldap.conf.gz

---

Now the smbldap-tools conf file need to be edited. Put this in `/etc/smbldap-tools/smbldap_bind.conf`:

```
slaveDN="cn=admin,dc=mandriva,dc=com"
slavePw="secret"
masterDN="cn=admin,dc=mandriva,dc=com"
masterPw="secret"
```

`smbldap_bind.conf` defines how to connect to and write to the LDAP server.

Then edit `smbldap.conf` and set those fields:

```
SID="S-1-5-21-128599351-419866736-2079179792"
sambaDomain="MANDRIVA"
ldapTLS="0"
suffix="dc=mandriva,dc=com"
sambaUnixIdPooldn="sambaDomainName=MANDRIVA,${suffix}"
#defaultMaxPasswordAge="45"
userSmbHome=""
userProfile=""
userHomeDrive=""
```

Now the directory can be populated. Type:

```
# smbldap-populate -m 512 -a administrator
```

A user called « administrator » will be created, and a prompt will ask you to give its password. Thanks to the « -m 512 » option, this user will belong to the « Domain Admins » group.

User password expiration

By default, the maximum password age of a SAMBA user is 42 days. Then the user will need to change his/her password.

If you don't want password to expire, type:

```
# pdbedit -P "maximum password age" -C 0
```

If you want to check your current password expiration policy:

```
# pdbedit -P "maximum password age"
```

Giving privileges to SAMBA users and groups

If « enable privileges = yes » is set on your `smb.conf`, you can give privileges to SAMBA users and groups.

For example, to give to "Domain Admins" users the right to join a machine to the domain:

```
# net -U administrator rpc rights grant 'DOMAIN\Domain Admins' SeMachineAccountPrivilege
Password:
Successfully granted rights.
```

Notice that you must replace « DOMAIN » by your SAMBA domain name in the command line.

Note: Users that can give privileges

Only users that belong to the "Domain Admins" group can use the **net rpc rights grant** command to assign privileges.

About SE Linux

The default SE Linux configuration may not allow SAMBA to launch the script defined in "add machine script", and so you won't be able to join a machine to the SAMBA domain.

MMC « base » plugin configuration

By default, you want your new user to belong to the « Domain Users » group.

You just need to set the « defaultUserGroup » option to « Domain Users » in `/etc/mmc/plugins/base.ini`.

MMC « SAMBA » plugin configuration

For a full description of the MMC SAMBA plugin configuration file see *MMC SAMBA plugin configuration file*.

You shouldn't need to edit the configuration file (`/etc/mmc/plugins/samba.ini`). This plugin won't be activated if your LDAP directory does not include the SAMBA schema, and well-known RIDs.

ACLs must be enabled on your filesystem. The SAMBA plugin needs them to set the ACLs when creating shares, and SAMBA will be able to map NTFS ACLs to the POSIX ACLs.

If you use XFS, ACLs are enabled by default. For ext3, you need to enable ACLs in `/etc/fstab`.

1.2.5 Shorewall plugin

Installation

Install the packages `python-mmc-shorewall` and `mmc-web-shorewall`.

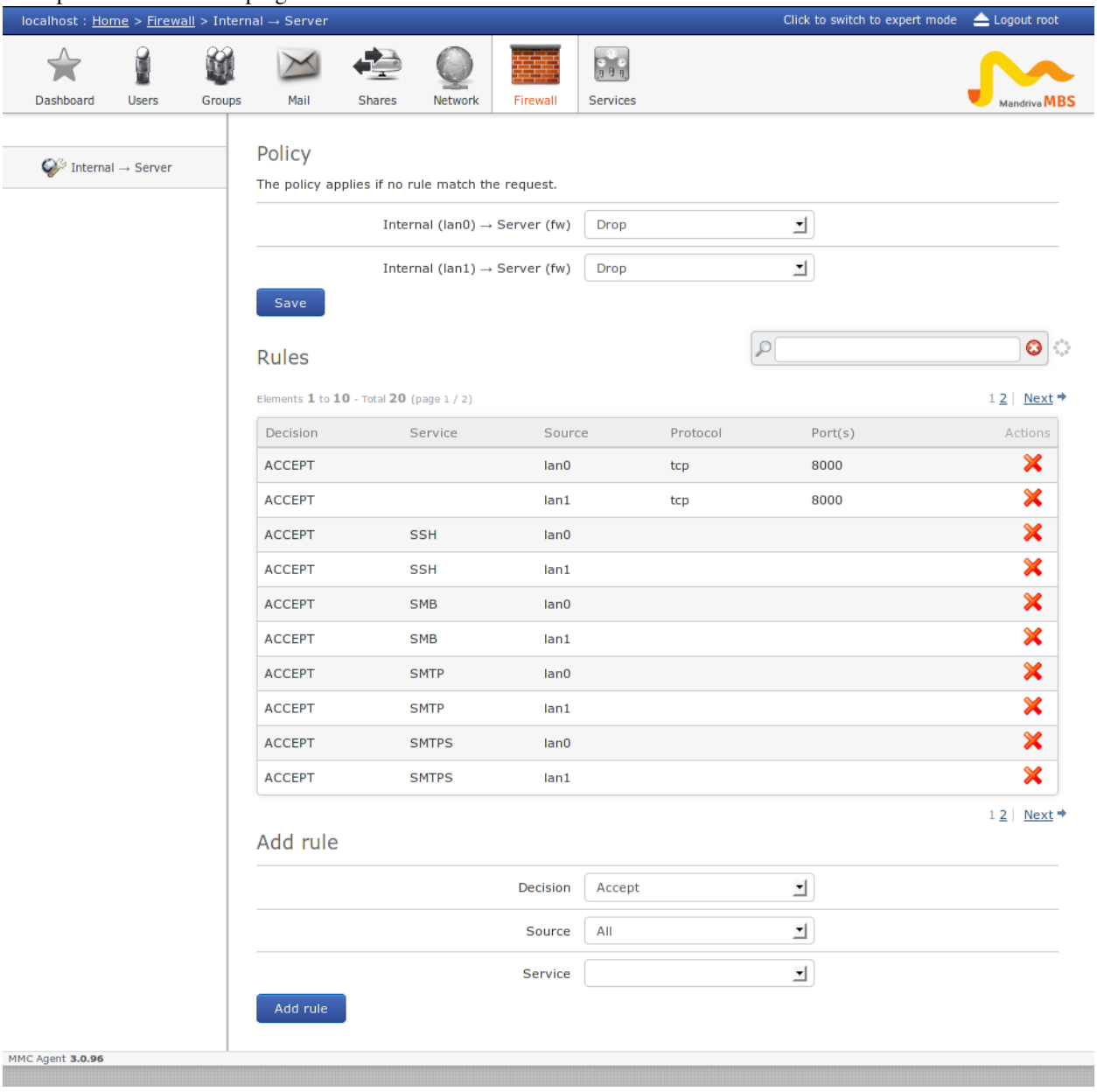
MMC « shorewall » plugin

THe shorewall plugin will manage the files in `/etc/shorewall`. The plugin is designed to manage internal and external interfaces. An external interface is generally connected to an insecure network (Internet), and internal interface is connected to a known/controlled network.

Typically if your server is installed in a datacenter and have a public interfaces to the Internet, they are external interfaces. If your server is in your local network you have only internal interfaces. A server acting as a gateway has generally one public interface and one interface interface.

Once your interfaces are defined as 'internal' or 'external' all the firewall configuration can be done from the MMC interface. Depending on your interfaces configuration you will be able to access more or less features. For example, if you have one internal and one external interface you will be able to create a NAT rule for your internal network.

Example of the shorewall plugin with two 'internal' interfaces:



MMC « shorewall » plugin configuration

Like every MMC plugin the configuration can be found in `/etc/mmc/plugins/shorewall.ini`

The plugin will assume that an interface is internal or external because of its zone name. By default if your zone begins by 'lan' (lan0, lanA ...) the interface will be considered as 'internal'. If the zone name begins by 'wan', the interface is considered as 'external'.

You can change theses names by changing the `external_zones_names` and `internal_zones_names` options.

Shorewall configuration

The initial shorewall configuration should be done manually. Check the shorewall docs for more information about shorewall configuratioon.

Example with a gateway (two interfaces, one internal, one external).

In `/etc/shorewall/interfaces` declare your network interfaces and associated zones:

```
lan0     eth0
wan0     eth1
```

In `/etc/shorewall/zones` declare your zones types:

```
fw      firewall
lan0    ipv4
wan0    ipv4
```

In `/etc/shorewall/policy` define the default policy between your zones:

```
fw      all     ACCEPT  # server -> anywhere
lan0    fw      DROP    # lan0 -> server
wan0    fw      DROP    # wan0 -> server
all     all     DROP    # catch-all rule
```

Finally, be sure that the file `/etc/shorewall/rules` exists

1.2.6 Squid plugin

Installation

Install the packages `python-mmc-squid` and `mmc-web-squid`.

LDAP directory configuration

Two groups will be created automatically in the LDAP tree when the mmc-agent starts with the squid plugin enabled:

- InternetMaster: the group with total privilegies to access any site and downloads at any time
- InternetFiltered: is the group with Internet and extensions filtred by a list of keywords and domains

The group names and their description can be changed in the configuration file of the plugin: *MMC squid plugin configuration file*.

Squid configuration

Please use the provided squid configuration available in `/usr/share/doc/mmc/contrib/squid/`.

The configuration of the `squid.conf` file was customized to provide LDAP authentication for the users. Copy the configuration file to `/etc/squid` or `/etc/squid3/` (on Debian).

1.2.7 SSH public keys plugin

Installation

Install the packages `python-mmc-sshlpk` and `mmc-web-sshlpk`.

LDAP directory configuration

You need to import the sshlpk schema into the LDAP directory. The schema file is provided by the `python-mmc-sshlpk` package in `/usr/share/doc/mmc/contrib/sshlpk/openssh-lpk.schema`.

Once this schema is imported, you will be able to manage ssh attributes thanks to the MMC.

Note: On Debian, run:

mmc-add-schema /usr/share/doc/mmc/contrib/sshlpk/openssh-lpk.schema /etc/ldap/schema

---

MMC « sshlpk » plugin configuration

For a full description of the MMC sshlpk plugin configuration file see *MMC sshlpk plugin configuration file*.

This plugin won't be activated if your LDAP directory does not include the sshlpk schema.

1.2.8 Userquota plugin

Installation

Install the packages `python-mmc-userquota` and `mmc-web-userquota`.

LDAP directory configuration

You need to import the quota schema into the LDAP directory. The schema file is provided by the `python-mmc-userquota` package in `/usr/share/doc/mmc/contrib/userquota/quota.schema`.

Once this schema is imported, you will be able to manage quota attributes thanks to the MMC.

Note: On Debian, run:

mmc-add-schema /usr/share/doc/mmc/contrib/userquota/quota.schema /etc/ldap/schema

---

Enabling filesystem quotas on your server

If you are using an ext3 or XFS filesystem you should add the "usrquota" option on the mountpoint(s) where you want to manage quotas in /etc/fstab.

If you want to manage quota on / with an XFS filesystem you need also to pass the kernel option `rootflags=usrquota`. You'll need to modify your GRUB configuration for this.

If you are using an XFS filesystem, you must remount manually the partition after adding the "usrquota" option on the mountpoints in /etc/fstab. On ext3 filesystems, you can remount the filesystem dynamicaly with the usrquota option using the following command:

```
mount -o remount,usrquota /path/to/mount/point
```

On ext filesystems you have to create quota files on your mountpoints :

```
quotacheck -cum /path/to/mount/point
```

This is not needed on XFS.

Enable the quotas on all mountpoints with:

```
quotaon -au
```

Check that the quotas are enabled with:

```
quotaon -aup
```

MMC « userquota » plugin configuration

In the diskquota section of `/etc/mmc/plugins/usrquota.ini` you need to specify the list of devices where you want to apply user quotas in the option `devicemap`.

The devicemap option use the following format :

```
device1:blocksize:displayname,device2:blocksize:displayname,...
```

The device is the unix name of the partition (eg: "/dev/sda1").

Note: Use the device name reported by the `quotaon -aup` command

The displayname is a string representing the device (eg: "Homes"). The quota blocksize value is 1024 on Linux x86.

For a full description of the MMC userquota plugin configuration file see *MMC userquota plugin configuration file*.

This plugin won't be activated if your LDAP directory does not include the quota schema or the quotas are not enabled on any mountpoints.

1.2.9 Configuration files

MMC mail plugin configuration file

This document explains the content of the MMC mail plugin configuration file.

Introduction

The « mail » plugin allows the MMC to add/remove mail delivery management attributes to users and groups, and mail virtual domains, etc. It uses the « base » plugin for all its related LDAP operations.

The plugin configuration file is `/etc/mmc/plugins/mail.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the mail.ini available sections:

Section name	Description	Optional
main	global mail plugin configuration	no
userdefault	Attributes and Objectclass values that are added or deleted when adding mail attributes to a user	yes
mapping	Map mail.schema attributes to other existing LDAP attributes	yes

Section « main »

This sections defines the global options of the mail plugin

Available options for the « main » section:

Option name	Description	Optional	Default value
disable	Is this plugin disabled ?	Yes	1
vDomainSupport	Is virtual domain management enabled ?	Yes	0
vDomainDN	Organizational Unit where virtual mail domains will be stored	Yes if vDomainSupport is disabled	ou=mailDomains, %(baseDN)s
vAliasesSupport	Is virtual aliases management enabled ?	Yes	0
vAliasesDN	Organizational Unit where virtual aliases will be stored	Yes if vAliasesSupport is disabled	ou=mailAliases, %(baseDN)s
zarafa	Is Zarafa LDAP fields support enables ?	Yes	0

Section « userdefault »

When adding the mail attributes to a user, you may want to change the value of the attributes that are added. Please look at the *MMC base plugin configuration file* for a look at how this section works.

The mailbox field of this section is very important to set because it determines the paths where the mails are delivered to users.

If the mails are delivered by Postfix, use this:

```
[userdefault]
mailbox = %homeDirectory%/Maildir/
```

If you use Dovecot as the delivery agent:

```
[userdefault]
mailbox = maildir:%homeDirectory%/Maildir/
```

Zarafa support

The zarafa.schema file must be included into the LDAP directory.

If Zarafa support is enabled, the "zarafa-user" object class will be automatically added to users if the administrator gives them mail access thanks to the MMC web interface.

The following fields are also available:

- Administrator of Zarafa (zarafaAdmin LDAP field)
- Shared store (zarafaSharedStoreOnly LDAP field)
- Zarafa account (zarafaAccount)
- Zarafa send as user list (zarafaSendAsPrivilege)

When you edit a group, you will also be able to set the "zarafa-group" object class to it.

Section « mapping »

When using an existing LDAP your mail attributes may not have the same name than the attributes of our mail.schema. The MDS mail plugin support attribute mapping so that you can use your LDAP without modification.

The following attributes can be mapped to other values: mailalias, maildrop, mailenable, mailbox, mailuserquota, mailhost

If your are using the zarafaAliases to store users aliases write:

```
[mapping]
mailalias = zarafaAliases
```

MMC network plugin configuration file

This document explains the content of the MMC network plugin configuration file.

Introduction

The « network » plugin allows the MMC Python API to manage DNS zones and hosts, DHCP subnet and hosts, into a LDAP. Patched version of ISC BIND (with LDAP sdb backend) and ISC DHCP (with LDAP configuration file backend) are needed. PowerDNS support is also available.

The plugin configuration file is `/etc/mmc/plugins/network.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the network.ini available sections:

Section name	Description	Optional
main	global network plugin configuration	yes
dns	DNS related configuration	no
dhcp	DHCP related configuration	no

Section « main »

This sections defines the global options of the network plugin.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Is the plugin disabled	yes	no

Section « dns »

This section defines where DNS needed files, directories and LDAP entities are located.

When the plugin starts for the first time, it creates:

- the directory `bindroot/named.ldap`. This directory will contains all zones definitions
- the file `bindroot/named.conf.ldap`. This file will include all the zone definitions stored into `bindroot/named.ldap/`

Available options for the "dns" section:

Option name	Description	Optional	Default value
type	DNS server type: “bind” or “pdns” (PowerDNS)	yes	bind
dn	LDAP DN where the DNS zones are stored	no	
logfile	path to BIND log file	no	
pidfile	path to BIND pid file	no	
init	BIND init script	no	
bindchroot-confpath	path to the named.ldap directory inside the BIND chroot. Don’t set it if BIND is not into a chroot.	no	
bindroot	path to the BIND configuration file directory	no	
bindgroup	gid which BIND is running (“bind” or “named”)	no	
dnsreader	LDAP user DN to use to read zone info	yes	
dnsreader-password	password of the user specified in dnsreader	not if dnsreader is set	

Here is an example for BIND on a Mandriva Corporate Server 4:

```
[dns]
type = bind
dn = ou=DNS,dc=mandriva,dc=com
pidfile = /var/lib/named/var/run/named.pid
init = /etc/rc.d/init.d/named
logfile = /var/log/messages
bindroot = /var/lib/named/etc/
bindchrootconfpath = /etc
bindgroup = named
dnsreader = uid=DNS Reader,ou=System Accounts,dc=mandriva,dc=com
dnsreaderpassword = s3cr3t
```

Section « dhcp »

This section defines where DHCP related files and LDAP entities are located.

Available options for the “backup-tools” section:

Option name	Description	Optional	Comment
dn	LDAP DN where the DHCP server configuration is stored	no	
pidfile	path to DHCP server pidfile	no	
init	path to DHCP service init script	no	
logfile	path to DHCP service log file	no	
leases	path to DHCP service leases file	no	
hostname	name of the DHCP server to user	no	Set manually the master DHCP hostname in the LDAP. If not set, DHCP name will be the local hostname. If set, you can configure the “ldap-dhcp-server-cn” option in dhcpd.conf to match this setting

MMC SAMBA plugin configuration file

This document explains the content of the MMC SAMBA plugin configuration file.

Introduction

The « samba » plugin allows the MMC to add/remove SAMBA attributes to users and groups, to manage SAMBA share, etc. It uses the « base » plugin for all its related LDAP operations.

The plugin configuration file is `/etc/mmc/plugins/samba.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the samba.ini available sections:

Section name	Description	Optional
main	global SAMBA plugin configuration	yes
hooks	Hooks for scripts that interacts with the MMC	yes
userdefault	Attributes and Objectclass values that are added or deleted when adding a new user into the LDAP	yes

Section « main »

This section defines the global options of the SAMBA plugin.

Available options for the “main” section:

Option name	Description	Optional	Default value
baseComputersDN	LDAP organisational unit DN where the SAMBA computer accounts are located	no	
sambaConfFile	Main SAMBA configuration file path	yes	/etc/samba/smb.conf
sambaInitScript	System SAMBA initialization script	yes	/etc/init.d/samba
sambaAvSo	VFS shared library location for anti-virus check on shares (scannedonly, vscan-clamav...). If this file is present, we can enable anti-virus check when creating a SAMBA share. This results to an option on the share : vfs object = libname (without .so suffix)	yes	/usr/lib/samba/vfs/vscan-clamav.so
defaultSharesPath	Directory where the SAMBA shares are created, if no path is specified	no	
authorizedSharePaths	Comma-separated list of directories where SAMBA shares are allowed to be created.	yes	The value of defaultSharesPath

Section « hooks »

The hooks system allow you to run external script when doing some operations with the MMC.

The script will be run as root user, with as only argument the full LDIF of the LDAP user. For the « addsmbattr » and « changeuserpasswd » hook, the LDIF file will contains the userPassword attributes in cleartext.

Available options for the "hooks" section:

Option name	Description	Optional
addsmbattr	path to the script launched when the SAMBA LDAP attributes has been added to a user	yes
changesambaattributes	path to the script launched when the SAMBA LDAP attributes has been changed on a user	yes
changeuserpasswd	path to the script launched when the SAMBA password of a user is changed	yes

Section « userdefault »

When adding the SAMBA attributes to a user, you may want to change the value of the attribute that are added. Please look at the *MMC base plugin configuration file* for a look at how this section works.

For example, if you want to delete the sambaPwdMustChange attribute of a user entry:

```
sambaPwdMustChange = DELETE
```

MMC squid plugin configuration file

This document explains the content of the MMC squid plugin configuration file.

Introduction

The plugin allows control of internet content filters, manipulating squid files directly and use the LDAP base to authentication of users.

The plugin configuration file is `/etc/mmc/plugins/squid.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the squid.ini available sections:

Section name	Description	Optional
main	global mail plugin configuration	no
squid	paths and names of LDAP access groups	no

Section « main »

This sections defines the global options of the squid plugin

Available options for the « main » section:

Option name	Description	Optional	Default value
disable	Is this plugin disabled ?	Yes	1

Section « squid »

Available options for the « main » section:

Option name	Description	Optional	Default value
squidBinary	path to the squid binary	No	/usr/sbin/squid3
squidInit	the path of the squid init script	No	/etc/init.d/squid3
squidPid	the path of squid pid file	No	/var/run/squid3.pid
sargBinary	the path of the sarg binary	Yes	/usr/bin/sarg
groupMaster	the name of the group that have full access	No	InternetMaster
groupMasterDesc	the group description	No	Full Internet access
groupFiltered	the name of the group that have a filtered access	No	InternetFiltered
groupFiltered-Desc	the group description	No	Filtered Internet access
squidRules	the path where will be stored rule files	No	/etc/squid/rules/
blacklist	path to the blacklist file	No	%(squidRules)s/blacklist.txt
whitelist	path to the whitelist file	No	%(squidRules)s/whitelist.txt
blacklist_ext	path to the extensions blacklist file	No	%(squidRules)s/blacklist_ext.txt
timeranges	path to the timeranges file	No	%(squidRules)s/timeranges.txt
machines	path to the machines file	No	%(squidRules)s/machines.txt

MMC sshlpk plugin configuration file

This document explains the content of the MMC sshlpk plugin configuration file.

Introduction

The « sshlpk » plugin allows the MMC to manage lists of SSH public keys on users. It uses the « base » plugin for all its related LDAP operations.

The plugin configuration file is `/etc/mmc/plugins/sshlpk.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the sshlpk.ini available sections:

Section name	Description	Optional
main	global sshlpk plugin configuration	no
hooks	hooks for scripts that interacts with the MMC	yes

Section « main »

This sections defines the global options of the sshlpk plugin

Available options for the “main” section:

Option name	Description	Optional	Default value
disable	Define if the plugin is disabled or not	no	no

Section « hooks »

The hooks system allow you to run external script when doing some operations with the MMC.

The script will be run as root user, with as only argument the full LDIF of the LDAP user.

Available options for the “hooks” section:

Option name	Description	Optional
updatesshkeys	path to the script launched when the user’s SSH public keys are updated	yes

MMC userquota plugin configuration file

This document explains the content of the MMC userquota plugin configuration file.

Introduction

The « userquota » plugin allows the MMC to set filesystem quotas to users. The plugin provides LDAP attributes for storing quota information. The plugin allows also to store network quotas in the LDAP directory for external tools. It uses the « base » plugin for all its related LDAP operations.

The plugin configuration file is `/etc/mmc/plugins/userquota.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Here are all the userquota.ini available sections:

Section name	Description	Optional
main	global userquota plugin configuration	no
diskquota	filesystem quota configuration	yes
networkquota	network quota configuration	yes

Section « main »

This sections defines the global options of the mail plugin

Available options for the « main » section:

Option name	Description	Optional	Default value
disable	Is this plugin disabled ?	no	yes

Section « diskquota »

Available options for the « diskquota » section:

Option name	Description	Optional	Default value
enable	Is this plugin enabled ?	No	0
devicemap	The definition of the filesystems using quotas	No	/dev/sda1:1024:Root
softquotablocks	Coef used to calculate the soft blocks limit	No	0.95
softquotainodes	Coef used to calculate the soft inodes limit	No	0.95
inodesperblock	Coef used to calculate the inodes limit from the blocks limit	No	1.60
setquotascript	Command template for applying quotas on filesystem	No	/usr/sbin/setquota $uid $softblocks $blocks $softinodes $inodes $devicepath
delquotascript	Command template for removing quotas on filesystem	No	/usr/sbin/setquota $uid 0 0 0 0 $devicepath
runquotascript	Script for setting quotas	No	/bin/sh

The soft limits of the quotas are calculated using the softquotablocks and softquotainodes coefs. The inode limit is calculated using the inodesperblock coef.

The inode limits protects the filesystem if some user create to much hardlinks as a hardlink use one inode but no block on the filesystem.

The setquotascript and delquotascript options define the commands templates used to apply or remove quotas on the filesystem. The runquotascript is the name of a shell script which contain the quota commands to be run on the system. If it is set to /bin/sh, then quotas will be applied on the local system. Check the applyquotas.sh example script to see how you can apply quotas on a different server. This is useful if your mmc-agent does not run on your file server.

Section « networkquota »

Available options for the « networkquota » section:

Option name	Description	Optional	Default value
enable	Is this plugin enabled ?	No	0
networkmap	The definition of networks using quotas	No	Internet:0.0.0.0/0:any

This section define the networks on which you want to use quotas. This allows you to store differents quotas values for differents network/protocol pair. This plugin will update the ldap records for network quotas for each user, but does not attempt to apply these quotas to a firewall, as this will be different for most people.

The networkmap option must be formatted with the following format :

```
displayName:network:protocol,...
----------------------
Internet:0.0.0.0/0:any,Local:192.168.0.0/24:any
```

1.3 Pulse 2

1.3.1 Introduction

Pulse 2 is an Open Source tool that simplifies application deployment, inventory, and maintenance of an IT network. It provides useful features to create rescue disk images to restore a unique computer or image to be deployed across the whole computers network. Remote application deployment and updates. Software and hardware inventory, remote diagnostic and control.

Pulse2 helps organizations with a range of a few computers to 100 000+ heterogeneous to inventory, maintain, update and take full control on their IT assets. It's support for heterogeneous platforms includes MS Windows, GNU/Linux (Mandriva, Redhat, Debian, Ubuntu., etc.), Mac OSX, HP-UX, IBM AIX and Solaris systems.

Pulse 2 is an easy-to-use, safe and flexible solution that allows you:

- Supervise large scale facilities through the use of a single Web interface console.
- Create and deploy hard disk images of your computers (new imaging module).
- Deploy new software and security updates on all your IT assets.
- Perform software and hardware inventory.
- Do remote diagnostics and remote management.

1.3.2 Installation

Installing the development version

In order to install Pulse2 and it's plugins you first need to install and configure *MMC*.

This how to will guide you through the installation and configuration of Pulse2 development environment

Installation form source code

Pre-requisites Debian:

```
# apt-get install git-core build-essential autogen autoconf libtool gettext
python-sqlalchemy python-mysqldb python-ldap python-openssl
python-twisted-web nsis xsltproc docbook-xsl
```

Centos:

```
# yum install git-core
```

Mandriva:

```
# urpmi git-core
```

Get the source code The development source code is managed in github https://github.com/mandriva-management-console/mmc.

Clone the github repository:

```
$ git clone git://github.com/mandriva-management-console/mmc.git
```

To compile and install all modules run:

```
$ $ cd pulse2/
$ ./autogen.sh
$ ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
$ make
# make install
```

You can update by running the following command:

```
$ git pull origin master
```

To keep your configuration files intact you may change the configure line to:

```
$ ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --disable-conf
```

The option `--disable-conf` will disable configuration files installation.

OpenLDAP

Debian:

```
# apt-get install slapd ldap-utils
```

Debconf will ask only ldap root password by default to more granular configuration use:

```
# dpkg-reconfigure slapd
```

Using dpkg-reconfigure debconf will ask you for:

- Omit OpenLDAP server configuration?: Choose <No>.
- DNS domain name: Enter you domain name.
- Organization name: Enter organization name.
- Admin password: Enter a password and confirm in next screen.
- Database backend to use: choose HDB.
- Do you want the database to be removed when slapd is purged: Choose <No>.
- Allow LDAPv2 protocol: Choose <NO>.

Centos:

```
# yum install
```

Mandriva:

```
# urpmi
```

OpenLDAP Schema

Old versions The mmc schema is needed to set ACLs on users in the MMC web interface:

```
# cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema /etc/ldap/schema/
```

Then in /etc/ldap/slapd.conf include the schema:

```
include /etc/ldap/schema/mmc.schema
```

New versions #TODO: Talk more about openldap changes in config and schema new storage.

Copy mmc schema to your current directory:

```
$ cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema .
```

Create a file mmc.conf with:

```
include     mmc.schema
```

Create a folder schemas:

```
$ mkdir schemas
```

Convert mcc.schema to ldif:

```
$ slaptest -f mmc.conf -F schemas/
```

Edit mmc schema, remove {0} from dn:, cn: and add cn=schema,cn=config to dn

```
dn: cn=mmc,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: mmc
```

Remove the following lines at the bottom of that file:

```
structuralObjectClass: olcSchemaConfig
entryUUID: 0ec2fe60-1381-1031-8f21-f92982aeda45
creatorsName: cn=config
createTimestamp: 20120405153755Z
entryCSN: 20120405153755.316520Z#000000#000#000000
modifiersName: cn=config
modifyTimestamp: 20120405153755Z
```

Add schema to ldap:

```
# ldapadd -Y EXTERNAL -H ldapi:/// -f schemas/cn\=config/cn\=schema/cn\=\{0\}mmc.ldif
```

Restart the slapd daemon.

MySQL

Debian:

```
# apt-get install mysql-server
```

Debconf will ask mysql root password.

Centos:

```
# yum install
```

Mandriva:

```
# urpmi
```

Apache HTTP server

Debian:

```
# apt-get install apache2 php5 php5-gd php5-xmlrpc
```

Centos:

```
# yum install
```

Mandriva:

```
# urpmi
```

Configuring apache2 and php Enable mmc web site:

```
# ln -s /etc/mmc/apache/mmc.conf /etc/apache2/sites-enabled/mmc.conf
```

Restart apache2:

```
# /etc/init.d/apache2 restart
```

Pulse setup

pulse2-setup will ask:

```
INFO     - Load defaults values from existing config
INPUT    - Enable audit module (Y/n): y
INPUT    - Enable inventory server (Y/n): y
INPUT    - Enable imaging server (Y/n): y
INPUT    - Enable package server (proxy) (Y/n): y
INPUT    - Server external IP address (default: 10.0.2.15): 172.16.0.4
INFO     - Run setup
INPUT    - Database host (default: localhost):
INPUT    - Database admin user (default: root):
INPUT    - Database admin password:
...
INPUT    - LDAP uri (default: ldap://127.0.0.1:389):
INPUT    - LDAP base DN (default: dc=mandriva, dc=com):
INPUT    - LDAP admin DN (default: cn=admin, dc=mandriva, dc=com):
INPUT    - LDAP admin password:
...
INPUT    - Wake-on-lan tool path (default: /usr/sbin/pulse2-wol):
```

DHCP Install

Debian:

```
# apt-get install isc-dhcp-server
```

DHCP Setup

The imaging module of Pulse 2 needs PXE functionalities, NFS and TFTP services. For PXE configure the DHCP server on the network to serve the Pulse2 PXE bootmenu.

For example with dhcp3-server in /etc/dhcp3/dhcpd.conf:

```
 subnet 192.168.0.0 netmask 255.255.255.0 {
     option broadcast-address 192.168.0.255; # broadcast address
     option domain-name "pulse2.test"; # domain name
     option domain-name-servers 192.168.0.2; # dns servers
     option routers 192.168.0.2; # default gateway
     pool {
         range 192.168.0.170 192.168.0.180;
         filename "/bootloader/pxe_boot";
        next-server 192.168.0.237;
     }
}
```

- filename and next-server are the relevant options to set.

You can find an example file for dhcp3 server in or repository.

Imaging client installation

Imaging client can run only on i386 compliant machines. It is not run directly on the server, but served through the network to i386 machines. For your convenience, prebuilt binaries are available, so that you can install it on a server which is not i386.

Once you have downloaded prebuilt binaries as pulse2-imaging-client-<version>_i386.tar.gz, simply run the following, as root: $ tar xfC pulse2-imaging-client-<version>_i386.tar.gz /

All files are extracted in /var/lib/pulse2/imaging/ dir.

As to serve the imaging client to the machines, you must then configure the following network services.

NFS setup

In */etc/exports* file, add the following lines:

```
/var/lib/pulse2/imaging/computers *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/masters *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/postinst *(async,ro,no_root_squash,subtree_check)
```

Then reload the new NFS configuration, as root.

Check the export list:

```
# showmount -e
Export list for imaging:
/var/lib/pulse2/imaging/masters *
/var/lib/pulse2/imaging/postinst *
/var/lib/pulse2/imaging/computers *
```

TFTP Setup

Bootloader and kernel are served to the client with TFTP protocol. We recommend using the atftpd server as it supports multicast..

You must configure the TFTP server to use as base directory:

```
/var/lib/pulse2/imaging
```

Note: don't use inetd.

Then check the configuration:

```
# atftp localhost
tftp> get /bootloader/pxe_boot
tftp> quit
# rm pxe_boot
```

DHCP Setup

The imaging module of Pulse 2 needs PXE functionalities, NFS and TFTP services. For PXE configure the DHCP server on the network to serve the Pulse2 PXE bootmenu.

For example with dhcp3-server in /etc/dhcp3/dhcpd.conf:

```
 subnet 192.168.0.0 netmask 255.255.255.0 {
     option broadcast-address 192.168.0.255; # broadcast address
     option domain-name "pulse2.test"; # domain name
     option domain-name-servers 192.168.0.2; # dns servers
     option routers 192.168.0.2; # default gateway
     pool {
         range 192.168.0.170 192.168.0.180;
         filename "/bootloader/pxe_boot";
        next-server 192.168.0.237;
     }
}
```

- filename and next-server are the relevant options to set.

You can find an example file for dhcp3 server in or repository.

Platform-specific installation instructions

How to install the Mandriva Management Console provided by a third-party Linux distributors.

In order to install Pulse2 and it's plugins you first need to install and configure *MMC*.

Note: Coming soon

Imaging client installation

Imaging client can run only on i386 compliant machines. It is not run directly on the server, but served through the network to i386 machines. For your convenience, prebuilt binaries are available, so that you can install it on a server which is not i386.

Once you have downloaded prebuilt binaries as pulse2-imaging-client-<version>_i386.tar.gz, simply run the following, as root: $ tar xfC pulse2-imaging-client-<version>_i386.tar.gz /

All files are extracted in /var/lib/pulse2/imaging/ dir.

As to serve the imaging client to the machines, you must then configure the following network services.

NFS setup

In */etc/exports* file, add the following lines:

```
/var/lib/pulse2/imaging/computers *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/masters *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/postinst *(async,ro,no_root_squash,subtree_check)
```

Then reload the new NFS configuration, as root.

Check the export list:

```
# showmount -e
Export list for imaging:
/var/lib/pulse2/imaging/masters *
/var/lib/pulse2/imaging/postinst *
/var/lib/pulse2/imaging/computers *
```

Pulse setup

pulse2-setup will ask:

```
INFO     - Load defaults values from existing config
INPUT    - Enable audit module (Y/n): y
INPUT    - Enable inventory server (Y/n): y
INPUT    - Enable imaging server (Y/n): y
INPUT    - Enable package server (proxy) (Y/n): y
INPUT    - Server external IP address (default: 10.0.2.15): 172.16.0.4
INFO     - Run setup
INPUT    - Database host (default: localhost):
INPUT    - Database admin user (default: root):
INPUT    - Database admin password:
...
INPUT    - LDAP uri (default: ldap://127.0.0.1:389):
INPUT    - LDAP base DN (default: dc=mandriva, dc=com):
INPUT    - LDAP admin DN (default: cn=admin, dc=mandriva, dc=com):
INPUT    - LDAP admin password:
...
INPUT    - Wake-on-lan tool path (default: /usr/sbin/pulse2-wol):
```

Installation from source tarball

In order to install Pulse2 and it's plugins you first need to install and configure *MMC*.

Get the current tarball at download page:

```
# tar xzvf pulse2-.x.y.x.tar.gz
# cd pulse2-x.y.z
# ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
# make
# make install
```

The *pulse2-setup* tool can then be used to provision databases, setup and check configuration files, etc. If you plan to use imaging service, please read the following section, as pulse2-setup does not handle with its configuration.

The default $PREFIX for installation is `/usr/local`. You can change it on the `./configure` line by adding the option `--prefix=/usr` for example.

configure options

The *configure* recognizes the following options to control how it operate:

- –help, -h: Print a summary of all of the options to configure, and exit.
- **–help=short –help=recursive: Print a summary of the options unique to this** package's configure, and exit. The short variant lists options used only in the top level, while the recursive variant lists options also present in any nested packages.
- **–version, -V: Print the version of Autoconf used to generate the configure** script, and exit.
- **–cache-file=FILE: Enable the cache: use and save the results of the tests** in FILE, traditionally config.cache. FILE defaults to /dev/null to disable caching.
- –config-cache, -C: Alias for –cache-file=config.cache.
- **–quiet, –silent, -q: Do not print messages saying which checks are** being made. To suppress all normal output, redirect it to /dev/null (any error messages will still be shown).
- **–srcdir=DIR: Look for the package's source code in directory DIR. Usually** configure can determine that directory automatically.
- **–prefix=DIR: Use DIR as the installation prefix. note Installation** Names for more details, including other options available for fine-tuning the installation locations.
- **–no-create, -n: Run the configure checks, but stop before creating any** output files.
- **–disable-conf: Do not install conf files. On a first install, you may not** use this option as configuration files are required.
- **–disable-conf-backup: Do not backup configuration files if they are** already installed. Default is to create backup files like *.~*N*~.
- –disable-wol: Do not build and install wake-on-lan tool.

OpenLDAP Schema

Old versions The mmc schema is needed to set ACLs on users in the MMC web interface:

```
# cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema /etc/ldap/schema/
```

Then in /etc/ldap/slapd.conf include the schema:

```
include /etc/ldap/schema/mmc.schema
```

New versions #TODO: Talk more about openldap changes in config and schema new storage.

Copy mmc schema to your current directory:

```
$ cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema .
```

Create a file mmc.conf with:

```
include    mmc.schema
```

Create a folder schemas:

```
$ mkdir schemas
```

Convert mcc.schema to ldif:

```
$ slaptest -f mmc.conf -F schemas/
```

Edit mmc schema, remove {0} from dn:, cn: and add cn=schema,cn=config to dn

```
dn: cn=mmc,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: mmc
```

Remove the following lines at the bottom of that file:

```
structuralObjectClass: olcSchemaConfig
entryUUID: 0ec2fe60-1381-1031-8f21-f92982aeda45
creatorsName: cn=config
createTimestamp: 20120405153755Z
entryCSN: 20120405153755.316520Z#000000#000#000000
modifiersName: cn=config
modifyTimestamp: 20120405153755Z
```

Add schema to ldap:

```
# ldapadd -Y EXTERNAL -H ldapi:/// -f schemas/cn\=config/cn\=schema/cn\=\{0\}mmc.ldif
```

Restart the slapd daemon.

Pulse setup

pulse2-setup will ask:

```
INFO     - Load defaults values from existing config
INPUT    - Enable audit module (Y/n): y
INPUT    - Enable inventory server (Y/n): y
INPUT    - Enable imaging server (Y/n): y
INPUT    - Enable package server (proxy) (Y/n): y
INPUT    - Server external IP address (default: 10.0.2.15): 172.16.0.4
INFO     - Run setup
INPUT    - Database host (default: localhost):
INPUT    - Database admin user (default: root):
INPUT    - Database admin password:
...
INPUT    - LDAP uri (default: ldap://127.0.0.1:389):
INPUT    - LDAP base DN (default: dc=mandriva, dc=com):
INPUT    - LDAP admin DN (default: cn=admin, dc=mandriva, dc=com):
INPUT    - LDAP admin password:
...
INPUT    - Wake-on-lan tool path (default: /usr/sbin/pulse2-wol):
```

Imaging client installation

Imaging client can run only on i386 compliant machines. It is not run directly on the server, but served through the network to i386 machines. For your convenience, prebuilt binaries are available, so that you can install it on a server which is not i386.

Once you have downloaded prebuilt binaries as pulse2-imaging-client-<version>_i386.tar.gz, simply run the following, as root: $ tar xfC pulse2-imaging-client-<version>_i386.tar.gz /

All files are extracted in /var/lib/pulse2/imaging/ dir.

As to serve the imaging client to the machines, you must then configure the following network services.

DHCP Setup

The imaging module of Pulse 2 needs PXE functionalities, NFS and TFTP services. For PXE configure the DHCP server on the network to serve the Pulse2 PXE bootmenu.

For example with dhcp3-server in /etc/dhcp3/dhcpd.conf:

```
 subnet 192.168.0.0 netmask 255.255.255.0 {
     option broadcast-address 192.168.0.255; # broadcast address
     option domain-name "pulse2.test"; # domain name
     option domain-name-servers 192.168.0.2; # dns servers
     option routers 192.168.0.2; # default gateway
     pool {
         range 192.168.0.170 192.168.0.180;
         filename "/bootloader/pxe_boot";
        next-server 192.168.0.237;
     }
}
```

- filename and next-server are the relevant options to set.

You can find an example file for dhcp3 server in or repository.

NFS setup

In */etc/exports* file, add the following lines:

```
/var/lib/pulse2/imaging/computers *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/masters *(async,rw,no_root_squash,subtree_check)
/var/lib/pulse2/imaging/postinst *(async,ro,no_root_squash,subtree_check)
```

Then reload the new NFS configuration, as root.

Check the export list:

```
# showmount -e
Export list for imaging:
/var/lib/pulse2/imaging/masters *
/var/lib/pulse2/imaging/postinst *
/var/lib/pulse2/imaging/computers *
```

TFTP Setup

Bootloader and kernel are served to the client with TFTP protocol. We recommend using the atftpd server as it supports multicast..

You must configure the TFTP server to use as base directory:

```
/var/lib/pulse2/imaging
```

Note: don't use inetd.

Then check the configuration:

```
# atftp localhost
tftp> get /bootloader/pxe_boot
```

```
tftp> quit
# rm pxe_boot
```

OpenLDAP Schema

Old versions

The mmc schema is needed to set ACLs on users in the MMC web interface:

```
# cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema /etc/ldap/schema/
```

Then in /etc/ldap/slapd.conf include the schema:

```
include /etc/ldap/schema/mmc.schema
```

New versions

#TODO: Talk more about openldap changes in config and schema new storage.

Copy mmc schema to your current directory:

```
$ cp /usr/share/doc/python-mmc-base/contrib/ldap/mmc.schema .
```

Create a file mmc.conf with:

```
include    mmc.schema
```

Create a folder schemas:

```
$ mkdir schemas
```

Convert mcc.schema to ldif:

```
$ slaptest -f mmc.conf -F schemas/
```

Edit mmc schema, remove {0} from dn:, cn: and add cn=schema,cn=config to dn

```
dn: cn=mmc,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: mmc
```

Remove the following lines at the bottom of that file:

```
structuralObjectClass: olcSchemaConfig
entryUUID: 0ec2fe60-1381-1031-8f21-f92982aeda45
creatorsName: cn=config
createTimestamp: 20120405153755Z
entryCSN: 20120405153755.316520Z#000000#000#000000
modifiersName: cn=config
modifyTimestamp: 20120405153755Z
```

Add schema to ldap:

```
# ldapadd -Y EXTERNAL -H ldapi:/// -f schemas/cn\=config/cn\=schema/cn\=\{0\}mmc.ldif
```

Restart the slapd daemon.

TFTP Setup

Bootloader and kernel are served to the client with TFTP protocol. We recommend using the atftpd server as it supports multicast..

You must configure the TFTP server to use as base directory:

```
/var/lib/pulse2/imaging
```

Note: don't use inetd.

Then check the configuration:

```
# atftp localhost
tftp> get /bootloader/pxe_boot
tftp> quit
# rm pxe_boot
```

Pulse2 quick install guide.

In order to install Pulse2 and it's plugins you first need to install and configure *MMC*.

Pre-requisites

The following tools are needed in order for Pulse2 to install and run properly:

- mmc-core framework
- python >= 2.5, with the following modules: * sqlalchemy * mysqldb
- OpenSSH client
- iputils (ping)
- perl, with the following modules: * syslog
- gettext
- NFS server (for imaging)
- 7z (for win32 client generation)
- NSIS (for building win32 agent pack, can be disabled)

Furthermore, the following services must be accessible, either on the local machine or through the network:

- MySQL
- DHCP server (for imaging purpose)

If you build Pulse2 from scm repository, you will also need the following tools (not needed if you use the .tar.gz archive):

- autoconf
- automake
- xsltproc
- docbook xsl

The MMC web interface is written in PHP4. Basically, you just need to install an Apache 2 server with PHP5 support. The XML-RPC module of PHP is needed too.

Packages naming conventions

Here are the packages naming conventions:

- pulse2-`SERVER`: the Pulse2 servers
- python-mmc-`PLUGIN`: MMC agent plugin
- mmc-web-`PLUGIN`: web interface plugin

Where `PLUGIN` can be one of dyngroup, glpi, imaging, inventory, msc, pkgs and pulse2.

Where `SERVER` can be one of common, imaging-server, inventory-server, launcher, package-server and scheduler.

Sample configuration files

All related sample configuration files are available or on our repository for Pulse2 plugins and servers.

Installation options

There are three easy options to install:

- Install an *official release from source tarball*.
- Install a version provided by your *operating system distribution*.
- Install the latest *development* version.

1.3.3 Configuration files

MMC dyngroup plugin configuration file

This document explains the content of the MMC dyngroup plugin configuration file.

Introduction

The « dyngroup » plugin is the MMC plugin in charge of creating, modifying and deleting groups of machines.

The plugin configuration file is `/etc/mmc/plugins/dyngroup.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now 3 sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	no
database	Needed options to connect to the database	no
querymanager	Describe how it react as a potential queriable plugin	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	yes	0
dyngroup_activate	Tell if the dynamic group part is activated or if there is only the static group part	yes	1
profiles_enable	Allow imaging fonctionnality on a profile, that is not available on static, and dynamic group	yes	0
default_module	Set the module that is going to be automatically selected is more than one dyngroup module is defined	yes	
max_elements_for_static_list	The maximum number of elements that have to be display in the static group creation list	yes	2000

« database » section This section defines the database options.

Available options for the "database" section:

Option name	Description	Optional	Default value
dbdriver	DB driver to use	no	mysql
dbhost	Host which hosts the DB	no	127.0.0.1
dbport	Port on which to connect to reach the DB	no	3306 (aka "default MySQL port")
dbname	DB name	no	dyngroup
dbuser	Username to give while conencting to the DB	no	mmc
dbpasswd	Password to give while connecting to the DB	no	mmc
dbdebug	Whenever log DB related exchanges	yes	ERROR
dbpoolrecycle	DB connection time-to-live	yes	60 (seconds)
dbpoolsize	The number of connections to keep open inside the connection pool	yes	5
dbsslenable	SSL connection to the database	yes	0
dbsslca	CA certificate for SSL connection	yes	
dbsslcert	Public key certificate for SSL connection	yes	
dbsslkey	Private key certificate for SSL connection	yes	

« querymanager » section This section define how this plugin react as a potential queriable plugin.

Available options for the "querymanager" section:

Option name	Description	Optional	Default value
activate	If queries on the group name are possible.	yes	1

MMC glpi plugin configuration file

This document explains the content of the MMC glpi plugin configuration file.

Introduction

The « glpi » plugin is the MMC plugin in charge of the glpi machine backend, it should only be used when invnetory is not used.

The plugin configuration file is `/etc/mmc/plugins/glpi.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now four sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	no
querymanager	Describe how it react as a potential queriable plugin	yes
authentication_glpi	Give the way to authenticate on glpi	yes
provisioning_glpi	Give the permissions that are going to be associated with users (based on permissions in glpi)	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the “main” section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	yes	0
dbdriver	DB driver to use	no	mysql
dbhost	Host which hosts the DB	no	127.0.0.1
dbport	Port on which to connect to reach the DB	no	3306 (aka "default MySQL port")
dbname	DB name	no	glpi
dbuser	Username to give while conencting to the DB	no	mmc
dbpasswd	Password to give while connecting to the DB	no	mmc
dbdebug	Whenever log DB related exchanges	yes	ERROR
dbpoolrecycle	DB connection time-to-live	yes	60 (seconds)
dbpoolsize	The number of connections to keep open inside the connection pool	yes	5
dbsslenable	SSL connection to the database	yes	0
dbsslca	CA certificate for SSL connection	yes	
dbsslcert	Public key certificate for SSL connection	yes	
dbsslkey	Private key certificate for SSL connection	yes	
localisation	Tells if the glpi entities are going to be used in pulse2	yes	
active_profiles	Tells which profiles are going to be used	yes	
filter_on	add a filter on the glpi_computers table when retrieving machines	yes	state==3

« querymanager » section This section define how this plugin react as a potential queriable plugin.

Available options for the "querymanager" section:

Option name	Description	Optional	Default value
activate	If queries on glpi inventory criterions are possible.	yes	True

« authentication_glpi » section This section define a way to authenticate thru glpi.

Available options for the "authentication_glpi" section:

Option name	Description	Optional	Default value
baseurl	glpi login page url yes http://glpi-server/glpi/		
doauth	Before provisioning, should we perform a GLPI authentication to create or update the user's informations in the GLPI database ?	yes	True

« provisioning_glpi » section This section define a way to do the user provisioning from glpi.

Available options for the "provisioning_glpi" section:

Option name	Description	Optional	Default value
exclude	users that are never going to be provisioned	yes	root
profile_acl_profileX	MMC web interface ACLs definition according to the user GLPI profile	yes	:##:base#main#default
profile_order	If the user belong to more than one profile, the first profile of this list will be used	yes	profile1 profile2 profile3

MMC imaging plugin configuration file

This document explains the content of the MMC imaging plugin configuration file.

Introduction

The « imaging » plugin is the MMC agent plugin that allows to manage all imaging related data.

The plugin configuration file is `/etc/mmc/plugins/imaging.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now three sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	no
database	Imaging database related options	no
imaging	Default values for the MMC web imaging plugin	no

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	no	0

« database » section This section defines how to connect to the imaging database.

Available options for the "database" section:

Option name	Description	Optional	Default value
dbdriver	DB driver to use	no	mysql
dbhost	Host which hosts the DB	no	127.0.0.1
dbport	Port on which to connect to reach the DB	yes	3306 (aka "default MySQL port")
dbname	DB name	no	imaging
dbuser	Username to use to connect to the DB	no	mmc
dbpasswd	Password to use to connect to the DB	no	mmc
dbpoolrecycle	This setting causes the pool to recycle connections after the given number of seconds has passed	yes	60
dbpoolsize	The number of connections to keep open inside the connection pool	yes	5
dbsslenable	SSL connection to the database	yes	0
dbsslca	CA certificate for SSL connection	yes	
dbsslcert	Public key certificate for SSL connection	yes	
dbsslkey	Private key certificate for SSL connection	yes	

« imaging » section This section defines default values to use in the MMC web interface in imaging related page.

Available options for the "imaging" section:

Option name	Description	Optional	Default value
web_def_date_fmt	Date format to use (see http://www.php.net/date for more informations)	yes	"%Y-%m-%d %H:%M:%S"
web_def_default_protocol	Network protocol to use for image restoration	yes	nfs
web_def_default_menu_name	Boot menu name	yes	Menu
web_def_default_timeout	Boot menu timeout in seconds	yes	60
web_def_default_background_uri	Boot menu background	yes	
web_def_default_message	Boot menu message	yes	Warning ! Your PC is being backed up or restored. Do not reboot !
web_def_kernel_parameters	Kernel parameters	yes	quiet
web_def_image_parameters	Image parameters	yes	

Pulse 2 Imaging Server configuration file

This document explains the content of the configuration file of the imaging server service from Pulse 2.

Introduction

The « imaging server » service is the Pulse 2 daemon in charge of managing backup folder on the server, based on the clients needs.

The service configuration file is `/etc/mmc/pulse2/imaging-server/imaging-server.ini`.

Like all Pulse 2 related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now four sections are available in this configuration file. The section describing the option can be duplicated if you need to pass more than one kind of option to the OCS inventory agent.

Section name	Description	Optional
main	Common imaging server configuration directives	*no*
daemon	Imaging server daemon related behaviors	*no*
helpers	Imaging server hooks	*no*
logger	Logging setting	*no*

« main » section This section is used to configure the imaging server services.

Available options for the "main" section:

Option name	Description	Optional	Type	Default value
adminpass	The password to be used when subscibing to this Pulse 2 server.	*no*	string	mandriva
base_folder	Where the images will be recorded	*no*	path	/var/lib/pulse2/imaging
host	The IP address on which the server will listen	*no*	string	0.0.0.0
net-boot_folder	Where the PXE elements will be taken from	*no*	path	/var/lib/tftpboot/pulse2
port	The port on which the server will listen.	*no*	int	1001
skel_folder	Where the original image template will be taken from	*no*	path	/usr/lib/pulse2/imaging/skel

« daemon » section This section sets the imaging service run-time options and privileges.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
group	The inventory service runs as this specified group.	*no*	string	root
pidfile	The inventory service store its PID in the given file.	*no*	path	/var/run/pulse2-imaging-server.pid
umask	The inventory service umask defines the right of the new files it creates (log files for example).	*no*	octal	0077
user	The inventory service runs as this specified user.	*no*	string	root

« helpers » section This section sets the imaging service hooks.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
client_add_path	The client_add script path	*no*	path	/usr/lib/pulse2/imaging/helpers/check_add_host
client_remove_path	The client_remove script path	*no*	path	/usr/lib/pulse2/imaging/helpers/check_remove_host
client_inventory_path	The client_inventory_path script path	*no*	path	/usr/lib/pulse2/imaging/helpers/info
menu_reset_path	The menu_reset_path script path	*no*	path	/usr/lib/pulse2/imaging/helpers/set_default
menu_update_path	The menu_update_path script path	*no*	path	/usr/lib/pulse2/imaging/helpers/update_menu
storage_create_path	The storage_create_path script path	*no*	path	/usr/lib/pulse2/imaging/helpers/create_config
storage_update_path	The storage_update_path script path	*no*	path	/usr/lib/pulse2/imaging/helpers/update_dir

« logger » section This section sets the logging system.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
log_file_path	The log path	*no*	path	/var/log/mmc/pulse2-imaging-server.log

MMC inventory plugin configuration file

This document explains the content of the MMC inventory plugin configuration file.

Introduction

The « inventory » plugin is the MMC plugin in charge displaying the content of the inventory database, and providing facilities for dynamic group creation.

The plugin configuration file is `/etc/mmc/plugins/inventory.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Available sections in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	no
inventory	Inventory related options	no
computer	Computers list's display content	no
expert_mode	Select which columns are only shown in expert mode	no
graph	Select which columns can be graphed	no
querymanager	Describe which part of the inventory is going to be queryable for the dyngroup plugin	yes
provisioning_inventory	Define the rules of provisioning of users from the inventory	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	no	0
software_filter	Allows to exclude softwares in the inventory software views according to their names, using a SQL expression. For example: %KB% allows to filter all softwares containing KB in their name. Multiple filters can be set using commas.	yes	

« inventory » section This section defines some global options.

Available options for the "inventory" section:

Option name	Description	Optional	Default value
dbdriver	DB driver to use	no	mysql
dbhost	Host which hosts the DB	no	127.0.0.1
dbport	Port on which to connect to reach the DB	yes	3306 (aka "default MySQL port")
dbname	DB name	no	inventory
dbuser	Username to give while conencting to the DB	no	mmc
dbpasswd	Password to give while connecting to the DB	no	mmc
dbpoolrecycle	This setting causes the pool to recycle connections after the given number of seconds has passed	yes	60
dbpoolsize	The number of connections to keep open inside the connection pool	yes	5
dbsslenable	SSL connection to the database	yes	0
dbsslca	CA certificate for SSL connection	yes	
dbsslcert	Public key certificate for SSL connection	yes	
dbsslkey	Private key certificate for SSL connection	yes	

« computer » section This section define what kind of informations will be displayed in computers list.

Available options for the "computer" section:

Option name	Description	Optional	Default value
content	List of additional parameters for the Computer object	yes	cn::Computer Name||displayName||Description
display	List of parameters that will be displayed in computers list	yes	“”

For exemple :

```
[computers]
content = Registry::Value::regdn::Path==DisplayName||Registry::Value::srvcomment::Path==srvcomment
display = cn::Computer Name||displayName::Description||srvcomment::Name||regdn::Display Name
```

« expert_mode » section This section defined columns that will be only displayed when in expert mode.

Available options for the “expert_mode” section:

Option name	Description	Optional	Default value
<Table name>	List of column in this Sql table that won’t be displayed in normal mode	yes	“”

« graph » section This section defined columns on which we will be able to draw graphs.

Available options for the “graph” section:

Option name	Description	Optional	Default value
<Table name>	List of column in this Sql table we will be able to draw	yes	“”

« querymanager » section This section defined columns that are going to be queryable to create groups from the dyngroup plugin.

Available options for the “querymanager” section:

Option name	Description	Optional	Default value
list	List of simple columns to query	yes	Entity/Label||Software/ProductName||Hardware/ProcessorType||Har
double	List of double columns to query (for exemple a software AND it’s version)	yes	Software/Products::Software/ProductName##Software/ProductVersio
halfstatic	List of columns to query with an hidden setted double columns (for exemple software KNOWING THAT version = 3)	yes	Registry/Value/displayname::Path##DisplayName

The separator to use between two entries is ||

List is a list of Table/Column that can be queryed as it.

Double is composed like that : NAME::Table1/Column1##Table2/Column2, knowing that name MUST start by the mysql table name plus the char ‘/’. It’s generaly used for having a AND on the same entry in a table (all machines having the software X and the version Y is not the same as all machines having the software X at the version Y)

Halfstatic is a list of Table/Column1/name complement::Column2##Value2, where the choices are on the column Column1 in the table Table where Column2 == Value2. The name complement is just to display purpose.

« provisioning_inventory » section This section defines some configuration directives for user provisioning with the inventory database. It allows to set access rights for users to the entities of the inventory database.

To enable the inventory provisioning system, you have to set this in `/etc/mmc/plugins/base.ini`:

```
[provisioning]
method = inventory
# Multiple provisining method can be used, for example:
# method = externalldap inventory
```

Available options for the "provisioning_inventory" section:

Option name	Description	Optional	Default value
exclude	space-separated list of login that won't be provisioned by this provisioner.	yes	
profile_attr	LDAP user attribute that is used to get the user profile	yes	
profile_entity_x	Space-separated list of entities assigned to the user profile "x". See the example below for more information	yes	

If the entity does not exist, it is created automatically in the database, as a child of the root entity (the root entity always exists).

For example:

```
[provisioning_inventory]
exclude = root
profile_attr = pulse2profile
profile_entity_admin = .
profile_entity_agent = entityA entityB
profile_entity_tech = %pulse2entity%
profile_entity_manager = plugin:network_to_entity
profile_entity_none =
profile_entity_default = entityA
```

In this example, the root user is never provisioned. The LDAP attribute used to get the user profile is called "pulse2profile".

The users with the "admin" profile are linked to the root entity, which is represented by the dot character. These users have access the root entity and all its sub-entities.

The users with the "agent" profile are linked to both entities "entityA" and "entityB" character. These users have access to entities "entityA" and "entityB", and all their sub-entities.

The users with the "tech" profile are linked to entities defined in the "pulse2entity" LDAP attribute values of these users.

The users with the "manager" profile are linked to entities computed by the "network_to_entity" provisioning plugin. See the next sub-section for more informations.

The users with the "none" profile are linked to no entity.

The users with no profile (the pulse2profile field is empty or don't exist) or with none of the profiles described in the configuration file are set to the "default" profile (be carefull, default is now a keyword).

« network_to_entity » plugin This plugin for the inventory provisioning system allows to link users to entities according to their IP when connecting to the MMC web interface.

The IP address of the user is determined by the Apache server running the MMC web interface thanks to the remote address of the HTTP connection. Then this IP address is forwarded to the MMC agent when authenticating and provisioning the user.

The IP address to entities mapping is done thanks to a rules file, similar to the one used by the inventory server to affect a computer inventory to an entity.

The rules file must be called `/etc/mmc/plugins/provisioning-inventory`. There is now to specify an alternate rules file.

Here is an example of rules file:

```
entityA              ip      match   ^192\\.168\\.1\\..*$
entityB,entityC      ip      match   ^192\\.168\\.0\\.19$
.                    ip      match   ^.*$
```

Each line of the rules file is processing starting from the top of the file, until one rule is valid. The user IP address is matched against a regular expression. If no rule match, the user is linked to no entity.

The first line links users connecting from the 192.168.1.0/24 to the entity called "entityA".

The second line links users connecting from the IP address 192.168.0.19 to the entities called "entityB" and "entityC".

The third line is a kind of catch-all rule.

Pulse 2 Inventory server configuration file

This document explains the content of the configuration file of the inventory server service from Pulse 2.

Introduction

The « inventory server » service is the Pulse 2 daemon in charge importing inventory sent from ocs inventory agents.

The service configuration file is `/etc/mmc/pulse2/inventory-server/inventory-server.ini`.

Like all Pulse 2 related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now four sections are available in this configuration file. The section describing the option can be duplicated if you need to pass more than one kind of option to the OCS inventory agent.

Section name	Description	Optional
main	Common inventory server configuration directives	*no*
database	Database connection parameters	*no*
daemon	Inventory server daemon related behaviors	*no*
option_XXX	Inventory agent option XXX	yes

All the other sections (loggers, handlers, ...) are related to Python language logging framework. See http://docs.python.org/lib/logging-config-fileformat.html.

« main » section This section is used to configure the inventory server service.

Available options for the "main" section:

Option name	Description	Optional	Type	Default value
host	The hostname or ip address where the inventory.	yes	string	localhost
port	The port on which the inventory listen.	yes	int	9999
ocsmapping	The mapping file betwen ocs inventory agent xml output and the database schema	yes	path	/etc/mmc/pulse2/inventory-server/OcsNGMap.xml
xmlfixplugindir	Directory containing Python scripts to fix the xml inventory before injecting	yes	path	/etc/mmc/pulse2/inventory-server/xml-fix
enablessl	SSL mode support	yes	boolean	False
verifypeer	use SSL certificates	yes	boolean	False
cacert	path to the certificate file describing the certificate authority of the SSL server	yes	path	/etc/mmc/pulse2/inventory-server/keys/cacert.pem
localcert	path to the SSL server private certificate	yes	path	/etc/mmc/pulse2/inventory-server/keys/privkey.pem
hostname	allow hostname in incoming inventory to be overridden by an other information from the inventory, for exemple Registry/Value|Path:DisplayName .	yes	string	Hardware/Name
default_entity	Default entity where computers are stored	yes	string	"." (root entity)
entities_rules_file	Rules file defining computer to entity mappings. See specific section to learn how it works.	yes	path	"" (no mapping)

The hostname option is a representation of the path in the inventory XML.

« database » section This section is documented into the MSC inventory plugin configuration documentation (see section *« inventory » section*).

« daemon » section This section sets the inventory service run-time options and privileges.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
pidfile	The inventory service store its PID in the given file.	yes	path	/var/run/pulse2-inventoryserver.pid
user	The inventory service runs as this specified user.	yes	string	root
group	The inventory service runs as this specified group.	yes	string (can be base64 encoded)	root
umask	The inventory service umask defines the right of the new files it creates (log files for example).	yes	octal	0077

« option_XXX » section This section define options that has to be given to the ocs inventory agent.

At the moment the only option which return will be inserted in the database is REGISTRY.

Each PARAM_YYY is for an XML tag PARAM in the inventory request. It is made of two values separated by ##. The first value is PARAM XML attributes, the second one is the content of the PARAM XML tag. The attributes are a list of couple attribute name, attribute value, the name and the value are separated by ::, each couple is separated by ||.

Available options for the `option_XXX` section:

Option name	Description	Optional	Type	Default value
NAME	The option name.	*no*	string	
PARAM_YYY	The option params.	yes	string	

For example :

```
[option_01]
NAME = REGISTRY
PARAM_01 = NAME::srvcomment||REGKEY::SYSTEM\\CurrentControlSet\\Services\\lanmanserver\\parameters||R
PARAM_02 = NAME::DisplayName||REGKEY::SYSTEM\\CurrentControlSet\\Services\\lanmanserver||REGTREE::2#
```

Rules file for computer to entity mapping This file defines a set of rules to assign a computer to an entity according to its inventory content.

Each line of the rules file is processing starting from the top of the file, until one rule is valid. When a rule matches, the processing stop, and the computer is linked to the entity. If no rule match, the user is linked to no entity.

If no rule matches, the computer is assigned to the default entity. If the entity does not exist, it is created automatically in the database, as a child of the root entity (the root entity always exists).

This file is made of four or more columns. Each column is separated by space or tab characters.

- The first column is the entity that will be assigned to the computer if the rule is valid. The root entity is specified by the dot character.
- The second column is the inventory component value that will be tested by the rule. This component is made of the name of an inventory table, the "/" character, and a column of this table. For example: Network/IP, Bios/ChipVendor, ... The `OcsNGMap.xml` file can also be used to get the available inventory component value.
- The third column is the operator of the rules. For the moment, only the "match" operator is available. The "match" operator allows to test the inventory component value with a regexp.
- The fourth column is a value that will be used by the operator. For the "match" operator, the value must be a regular expression.

For example:

```
.               Network/IP      match   ^192\\.168\\.0\\..*$
"entity A"      Network/IP      match   ^172\\.16\\..*$
entityB         Network/IP      match   ^10\\..*$          and        Hardware/OperatingSystem     ma
```

The first line links all computers with an IP address starting with 192.168.0. (network 192.168.0.0/24) to the inventory root entity.

The second line links all computers with an IP address starting with 172.16. (network 172.16.0.0/24) to the entity called "entity A". Entity name can be written between double-quotes if they contains space characters in their name.

The third line links all computers with an IP address starting with "10." (network 10.0.0.0/8) and with the "Linux" OS to the entity called entityB.

Pulse 2 Launcher configuration file

This document explains the content of the configuration file of the launcher service from Pulse 2.

Introduction

The « Launcher » service is the Pulse 2 daemon in charge of doing jobs on clients on scheduler orders.

The service configuration file is `/etc/mmc/pulse2/launchers.ini` (please note the ending "s").

Like all Pulse 2 related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

Some sections describing the different available launchers may appear, their name must begin with `launcher_`. The idea behind this is that the main section controls the common behavior of launchers, the others control the specific behaviors.

Section name	Description	Optional
launchers	Common launchers configuration directives	yes
wrapper	wrapper related options	yes
ssh	ssh modus-operandi related section	yes
daemon	Launchers services related behaviors	yes
wol	WOL related behaviors	yes
wget	Wget related options	yes
tcp_sproxy	Wget related options	yes
smart_cleaner	Smart cleaning options	yes
scheduler_XXX	Referent scheduler location	no
launcher_XXX	Configuration for launcher_XXX	no

All the other sections (loggers, handlers, ...) are related to Python language logging framework. See http://docs.python.org/lib/logging-config-fileformat.html.

« launchers » section This section is used to give directives common to every launcher service.

Available options for the "launchers" section:

Option name	Description	Optional	Type	Default value
halt_command	The halt command to use on a client, after a successful deployment.	yes	string	/bin/shutdown.exe -f -s 1 || shutdown -h now
inventory_command	The inventory command to use on a client, after a successful deployment.	yes	string	*export P2SRV=‘echo $SSH_CONNECTION \| cut -f1 -d*; export P2PORT=9999; export http_proxy=””; export ftp_proxy=””; ([-x /cygdrive/c/ProgramFiles/FusionInventory-Agent/perl/bin/fusioninventory-agent] && /cygdrive/c/ProgramFiles/FusionInventory-Agent/perl/bin/perl “C:Program FilesFusionInventory-Agentperl\bin\fusioninventory-agent” /server=http://P2SRV:$P2PORT) || ([-x /cygdrive/c/ProgramFiles(x86)/FusionInventory-Agent/perl/bin/fusioninventory-agent] && /cygdrive/c/ProgramFiles(x86)/FusionInventory-Agent/perl/bin/perl “C:Program Files (x86)FusionInventory-Agentperl\bin\fusioninventory-agent” /server=http://P2SRV:$P2PORT) \|\| ([-x /cygdrive/c/ProgramFiles/OCSInventoryAgent/OCSInventory.exe] && /cygdrive/c/ProgramFiles/OCSInventoryAgent/OCSInventory.exe /np /server:$P2SRV /pnum:$P2PORT) \|\| ([-x /cygdrive/c/ProgramFiles(x86)/OCSInventoryAgent/OCSInventory.ex] && /cygdrive/c/ProgramFiles(x86)/OCSInventoryAgent/OCSInventory.ex /np /server:$P2SRV /pnum:$P2PORT) \|\| ([-x /usr/bin/ocsinventory-agent] && /usr/bin/ocsinventory-agent –server=http://$P2SRV:$P2PORT) \|\| ([-x /usr/sbin/ocsinventory-agent] && /usr/sbin/ocsinventory-agent –server=http://$P2SRV:$P2PORT) || ([-x /usr/local/sbin/ocs_mac_agent.php] && /usr/local/sbin/ocs_mac_agent.php)‘
launcher_path	The Launcher main script location, used by launchers-manager to start and daemonize the services.	yes	path	/usr/sbin/pulse2-launcher
max_command_age	The parameter which limits a command’s time lenght. A command must take less than this value (in seconds), or being killed; High values mean that the command will have more time to complete, thus may also stay blocked longer. Only works for ASYNC commands.	yes	int, seconds	86400 (one day)
max_ping_time	Timeout when attempting to ping a client: A ping is aborded if it takes more that this value (in seconds). High values will minimize false-positives (aborded probe even if the client if obviously reachable). Lower values will enhance interface reponse time (but lead to more false-positives).	yes	int, seconds	4 (seconds)
max_probe_time	Timeout when attempting to probe a client: A probe is aborded if it takes more that this value (in seconds). High values will	yes	int, seconds	20 (seconds)

« daemon » section This section sets the pulse2-launchers-manager and pulse2-launchers service run-time options and privileges.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
group	The pulse2-launchers-manager and pulse2-launchers services run as this specified group.	yes	group	root
pidfile	The launcher services PID, used by pulse2-launchers-manager to track the launchers services.	yes	path	/var/run/pulse2
umask	The pulse2-launchers-manager and pulse2-launchers services umask defines the right of the new files they create (log files for example).	yes	octal	0077
user	The pulse2-launchers-manager and pulse2-launchers service run as this specified user.	yes	user	root

« wrapper » section This section define the wrapper behavior.

Available options for the "wrapper" section:

Option name	Description	Optional	Type	Default value
max_exec_time	Default max exec time in seconds, older process are killed using SIGKILL. Different from max_command_age as beeing handled by the wrapper itself, so it also works for SYNC commandS.	yes	int, in seconds	21600 (6 hours)
max_log_size	Cap generated logs to this value	yes	int, in bytes	512000 (500 kB)
path	Pulse 2 launcher wrapper (ie "job launcher") location.	yes	path	/usr/sbin/pulse2-output-wrapper

« ssh » section This section define global ssh (and scp) options.

Available options for the "ssh" section:

Option name	Description	Optional	Type	Default value
default_key	The default SSHv2 key to use, the config code will look for an "ssh_<default_key>" entry in the config file. ssh_* are ssh keys, * her names, f.ex. by using sshkey_default = /root/.ssh/id_rsa, /root/.ssh/id_rsa will be known as the 'default' key.	yes	string	default
forward_key	Should we perform key-forwarding (never, always, or let = let the scheduler take its decision)	yes	string	let
scp_path	Path to the SCP binary	yes	string	/usr/bin/scp
ssh_options	Options passed to OpenSSH binary (-o option).	yes	list of space separated strings	LogLevel=ERROR UserKnownHostsFile=/dev/null StrictHostKeyChecking=no Batchmode=yes PasswordAuthentication=no ServerAliveInterval=10 CheckHostIP=no ConnectTimeout=10
ssh_agent_path	Path to the SSH agent	yes	string	/usr/bin/ssh-agent
ssh_path	Path to the SSH binary	yes	string	/usr/bin/ssh
sshkey_default	The "default" ssh key path.	yes	path	/root/.ssh/id_rsa
sshkey_XXXXXX	The "XXXXXX" ssh key path (when more than one key may be used).	yes	string	

« wget » section This section sets the pulse2-launchers wget options (for the pull part of the push/pull mode)

Available options for the "wget" section:

Option name	Description	Optional	Type	Default value
check_certs	Put the check certificate flag.	yes	boolean	False
resume	Attempt to resume a partialy completed transfert	yes	boolean	True
wget_options	Options passed to wget binary.	yes	string	""
wget_path	wget binary path (on client)	yes	string	/usr/bin/wget

« rsync » section This section sets the pulse2-launchers rsync options (for the push mode)

Available options for the "rsync" section:

Option name	Description	Optional	Type	Default value
resume	Attempt to resume a partial completed transfert	yes	boolean	True
rsync_path	rsync binary path (on server)	yes	string	/usr/bin/rsync
set_executable	Do we force +/-X on uploaded files (yes/no/keep). See below.	yes	string	yes
set_access	Do we enforce permissions of uploaded files (private/restricted/public). See below.	yes	string	private

Uploaded file permissions:

set_access \ set_executable	yes	no	keep
private	u=rwx,g=,o=	u=rw,g=,o=	u=rwX,g=,o=
restricted	u=rwx,g=rx,o=	u=rw,g=r,o=	u=rwX,g=rX,o=
public	u=rwx,g=rwx,o=rx	u=rw,g=rw,o=r	u=rwX,g=rwX,o=rX

« wol » section This section sets the wol feature handling.

Available options for the "wol" section:

Option name	Description	Optional	Type	Default value
wol_bcast	WOL IP BCast adress.	yes	string	255.255.255.255
wol_path	Pulse 2 scheduler awaker (via WOL "magic packet").	yes	path	/usr/sbin/pulse2-wol
wol_port	WOL TCP port.	yes	string	40000

« tcp_sproxy » section This section sets the tcp_sproxy feature handling, mainly used by the VNC feature.

Available options for the "tcp_sproxy" section:

Option name	Description	Optional	Type	Default value
tcp_sproxy_path	Pulse 2 TCP Secure Proxy (woot !) path	yes	path	/usr/sbin/pulse2-tcp-sproxy
tcp_sproxy_host	Fill-in the following option if you plan to use VNC, it will be the "external" IP from the VNC client point-of-view	yes	string	""
tcp_sproxy_port_range	The proxy uses a port range to establish proxy to the client: 2 ports used per connection	yes	int range	8100-8200
tcp_sproxy_establish_delay	This is the max delay in seconds for the connection to the client timeout	yes	seconds	20
tcp_sproxy_connect_delay	The connect delay allow the initial connection to be established within N seconds (ie. a client as N seconds to connect to the proxy after a port has bee found, then the connection is dropped and further connections will be impossible	yes	seconds	60
tcp_sproxy_session_length	The number of seconds a connection will stay open after the initial handshake, conenction will be closed after this delay even if still in use	yes	seconds	3600 (one hour)

« smart_cleaner » section This section sets the wol feature handling.

Available options for the "wol" section:

Option name	Description	Optional	Type	Default value
smart_cleaner_path	Pulse 2 smart cleaner path (on client), not used if empty	yes	path	/usr/bin/pulse2-smart-cleaner.sh
smart_cleaner_options	Pulse 2 smart cleaner option (see win32 agent doc)	yes	array, space-separated	''

« scheduler_XXX » section This section define how the launchers may reach their referent scheduler.

Available options for the "scheduler" section:

Option name	Description	Optional	Type	Default value
awake_incertitude_factor	Add or remove to awake_time can be the same that the scheduler awake_time, add a little randomness here. Default value is .2, ie +/- 20 %. For example we will awake every 10 minutes, more or less 2 minutes. Values lower than 0 or greater than .5 are rejected Use this if your scheduler has the same awake time and busy each time we have to send our results	yes	float	.2
awake_time	The launcher will periodicaly awake (for exemple to send results to is scheduler), with this key a specific periodicity can be given. Field unit is the "second".	yes	int	600
defer_results	In async mode, whenever immedialetly send results to referent scheduler upon job completion or wait for being waked up (see above)	yes	string	no
enablessl	Flag that tells if SSL should be used to connect to the scheduler	yes	boolean	True
host	The referent scheduler IP address	yes	string	127.0.0.1
password	The password to use when authenticating vs our referent scheduler	yes	string or base64	password
port	The referent scheduler TCP port	yes	string	8000
username	The login name to use when authenticating vs our referent scheduler	yes	string	username

« launcher_XXX » section This section define specific options for all launchers on the server.

Available options for the "launcher_XXX" section:

Option name	Description	Optional	Type	Default value
bind	The launcher binding IP address.	yes	string	127.0.0.1
cacert	path to the certificate file describing the certificate authority of the SSL server	*no if enablessl is set*	path	/etc/mmc/pulse2/scheduler/keys/cace
certfile	deprecated (see cacert)			
enablessl	SSL mode support	*no*	boolean	1
localcert	path to the SSL serverprivate certificate	*no if enablessl is set*	path	/etc/mmc/pulse2/scheduler/keys/privl
password	The password to use when authenticating vs this launcher	yes	string or base64	password
port	The launcher binding TCP port.	*no*	int	
privkey	deprecated (see localcert)			
slots	The number of available slots (ie. maximum number of concurrent jobs)	yes	int	300
scheduler	The referent scheduler	yes	string	the first defined scheduler
username	The login name to use when authenticating vs this launcher	yes	string	username
verifypeer	Check that our parent scheduler present a signed certificate	*no if enablessl is set*	boolean	False
logconffile	path to the file containing the logging configuration of this launcher (the format of this file is described in the Python documentation. If it is not set, the default logging configuration is read from the launchers.ini file.	yes	string	

MMC MSC plugin configuration file

This document explains the content of the MMC MSC plugin configuration file.

Introduction

The « MSC » plugin is the MMC plugin in charge of recording commands in the MSC database, and gathering results from the database.

The plugin configuration file is `/etc/mmc/plugins/msc.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
```

```
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now five sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	yes
msc	MSC related options	yes
web	Web interface default options	yes
package_api	Describe how to reach the API package service	yes
schedulers	Describe how to reach the different MSC Schedulers	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	yes	1

« msc » section This section defines some global options.

Available options for the "msc" section:

Option name	Description	Optional	Default value
qaction-path	Folder from where Quick Action scripts are tacken	yes	/var/lib/pulse2/qactions
repopath	Folder from where packages will be copied (push mode)	yes	/var/lib/pulse2/packages
dbdriver	DB driver to use	yes	mysql
dbhost	Host which hosts the DB	yes	127.0.0.1
dbport	Port on which to connect to reach the DB	yes	3306 (aka "default MySQL port")
dbname	DB name	yes	msc
dbuser	Username to give while conencting to the DB	yes	msc
dbpasswd	Password to give while connecting to the DB	yes	msc
dbdebug	Whenever log DB related exchanges	yes	ERROR
dbpoolre-cycle	DB connection time-to-live	yes	60 (seconds)
default scheduler	default scheduler to use	yes	
ig-nore_non_rfc2780	Enable filter for non unicast IP addresses when inserting computers IP address in MSC database	yes	1
ig-nore_non_rfc1918	Enable filter for non private IP addresses when inserting computers IP address in MSC database	yes	0
ex-clude_ipaddr	Enable filter made of comma separated values with filtered ip addresses or network ranges, used when inserting computers IP address in MSC database. For example: exclude_ipaddr = 192.168.0.1,10.0.0.0/10.255.255.255	yes	
in-clude_ipaddr	Disable filter made of comma separated values with accepted ip addresses or network ranges, used when inserting computers IP address in MSC database. The IP addresses matching this filter are always accepted and never take out by the other filters. For example: include_ipaddr = 192.168.0.1,10.0.0.0/10.255.255.255	yes	
ig-nore_non_fqdn	Enable filter for host name that are not FQDN. If filtered, the host name won't be used by the scheduler to find the target IP address	yes	0
ig-nore_invalid_hostnames	Enable filter for host name that are invalid (that contains forbidden characters). If filtered, the host name won't be used by the scheduler to find the target IP address.	yes	0
ex-clude_hostname	Enable filter for host name that are invalid if they match a regexp from this list of regexp. If filtered, the host name won't be used by the scheduler to find the target IP address. For example: exclude_hostname = computer[0-9]* server[0-9]*	yes	
in-clude_hostname	The host names matching at least one regexp from this list of regexp will never be filtered. For example: For example: include_hostname = computer[0-9]* server[0-9]*	yes	
wol_macaddr_blacklist	Space separated regexps to match MAC address to filter when inserting a target for a command into the database. For example: wol_macaddr_blacklist = 12:.* 00:.*	yes	

« scheduler_XXX » section This section define available schedulers (one per scheduler, "XXX" must be an integer).

Available options for the "scheduler_XXX" section:

Option name	Description	Optional	Default value
host	The scheduler IP address.	yes	127.0.0.1
port	The scheduler TCP port.	yes	8000
enablessl	Flag that tells if SSL should be used to connect to the scheduler	yes	1
username	The name to use when we send XMLRPC commands to this scheduler.	yes	username
password	The password to use when we send XMLRPC commands to this scheduler.	yes	password

By default, a scheduler is always defined:

```
[scheduler_01]
host=127.0.0.1
port=8000
username = username
password = password
enablessl = 1
```

« web » section This section defined some default web fields.

Available options for the "main" section:

Option name	Description	Optional	Default value
web_def_awake	Check "Do WOL on client" ?	yes	1
web_def_inventory	Check "Do inventory on client" ?	yes	1
web_def_mode	Fill default package send mode	yes	push
web_def_maxbw	Fill default max usable bw	yes	0
web_def_delay	Fill delay between two attempts	yes	60
web_def_attempts	Fill max number of attempts	yes	3
web_def_deployment_intervals	Fill deployment time window	yes	
web_dlpath	Directory of target computers from which a file is download when a user perform the download file action in the computers list. If empty, the download file action is not available on the web interface.	yes	
web_def_dlmaxbw	Max bandwidth to use when download a file from a computer. Set to 0 by default. If set to 0, there is no bandwidth limit applied.	yes	0
web_allow_local_proxy	Possibility to use proxy mode for software deployment on groups	yes	True
web_def_local_proxy_mode	Default proxy mode, defaut "multiple", other possible value "single".	yes	multiple
web_def_max_clients_per_proxy	Max number of clients per proxy in proxy mode.	yes	10
web_def_proxy_number	Number of auto-selected proxy in semi-auto mode.	yes	2
web_def_proxy_selection_mode	Default mode (semi_auto / manual).	yes	semi_auto
vnc_show_icon	May the VNC applet used ? (this setting simply (en/dis)able the display of the VNC action button)	yes	True
vnc_view_only	Allow user to interact with remote desktop	yes	False
vnc_network_connectivity	Use a VNC client pre-defined rule	yes	lan
vnc_allow_user_control	Display applet control to user	yes	True
vnc_port	The port to use to connect to a VNC	yes	5900

Currently available profiles for VNC (`vnc_network_connectivity`):

- fiber: for high speed local networks (low latency, 10 Mb/s per connection)
- lan: for 100 Mb local networks (low latency, 3 Mb/s per connection)
- cable: for high-end broadband links (high latency, 400 kb/s per connection)

- dsl: for low-end broadband links (high latency, 120 kb/s per connection)
- isdn: (high latency, 75 kb/s)

« Client probing behavior » The LED which represents the client status can take four colors:

- black: no probe done
- red: all probe failed
- orange: minimal probe succedeed (ping), maximal probe failed (ssh)
- green: all probe succedeed

Available probes are: none (field is empty), ping, ssh, ping_ssh (ie. both).

For networks where icmp is not allowed, ping may be disabled: probe_order=ssh

To speed-up display, ssh may be disabled: probe_order=ping

To fully disable probe: probe_order=

Default conf: none (empty)

« package_api » section This section is used to tell to the plugin where to find its Package service.

Available options for the "main" section:

Option name	Description	Optional	Default value
mserver	The service IP address	yes	127.0.0.1
mport	The service TCP port	yes	9990
mmountpoint	The service path	yes	/rpc

Pulse 2 Package server configuration file

This document explains the content of the configuration file of the package server service from Pulse 2.

Introduction

The « package server » service is the Pulse 2 daemon that implement all the package apis, it permit the creation, edition, suppression, share, mirroring... of packages.

The service configuration file is `/etc/mmc/pulse2/package-server/package-server.ini`.

Like all Pulse 2 related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

The section describing the `mirror`, `package_api_get` or `package_api_put` can be duplicated if you need to have more than one api of this kind.

Section name	Description	Optional
main	Common package server configuration directives	no
daemon	Package server daemon related behaviors	yes
ssl	Package server ssl related dehaviors	yes
mirror_api		yes
user_package_api		yes
scheduler_api		yes
imaging_api		yes
mirror:XX		yes
package_api_get:XX		yes
package_api_put:XX		yes

All the other sections (loggers, handlers, ...) are related to Python language logging framework. See http://docs.python.org/lib/logging-config-fileformat.html.

« main » section This section is used to configure the inventory server services.

Available options for the "main" section:

Option name	Description	Optional	Default value
host	The hostname or ip address where the inventory.	yes	localhost
port	The port on which the inventory listen.	yes	9999
use_iocp_reactor	*Windows XP, Windows 2003 and Windows 2008 only.* This option sets the Twisted event loop to use the IOCP reactor for better performance. Please read *Pulse 2 package server performance on win32 platforms*	yes	0
package_detect_activate	Is package autodetection activated	yes	0
package_detect_loop	Time between two loops of detection	yes	60
package_detect_smart_method	methods in none, last_time_modification, check_size; for more than 1 method, separate with ","	yes	none
package_detect_smart_time		yes	60
package_mirror_loop		yes	5
package_mirror_target	Package api can synhronise package data to others servers; package synchronisation targets	yes	
package_mirror_status_file	package synchronisation state file. used only if package_mirror_target is defined. File where pending sync are written so that they can be finished on package server restart.	yes	/var/data/mmc/status
package_mirror_command	package synchronisation command to use	yes	/usr/bin/rsync
package_mirror_command_options	package synchronisation command options	yes	-ar –delete
package_mirror_level0_command_options	package synchronisation command on only one level options	yes	-d –delete
package_mirror_command_options_ssh_options	options passed to SSH via "-o" if specified –rsh is automatically added to package_mirror_command_options	yes	""
package_global_mirror_activate	loop for the sync of the whole package directory; can only be activated when package_mirror_target is given	yes	1
package_global_mirror_loop		yes	3600
package_global_mirror_command_options		yes	-ar –delete
real_package_deletion	real package deletion	yes	1
mm_assign_algo	machine/mirror assign algo	yes	default
up_assign_algo	user/packageput assign algo	yes	default

`package_mirror_command_options_ssh_options` can be for exemple :

```
IdentityFile=/root/.ssh/id_rsa StrictHostKeyChecking=no Batchmode=yes PasswordAuthentication=no Serve
```

« daemon » section This section sets the package server service run-time options and privileges.

Available options for the "daemon" section:

Option name	Description	Optional	Default value
pidfile	The package server service store its PID in the given file.	yes	/var/run/pulse2-package-server.pid
user	The inventory service runs as this specified user.	yes	root
group	The inventory service runs as this specified group.	yes	root
umask	The inventory service umask defines the right of the new files it creates (log files for example).	yes	0077

« ssl » section Available options for the "ssl" section:

Option name	Description	Optional	Default value
username		yes	""
password		yes	""
enablessl	SSL mode support	yes	1
verifypeer	use SSL certificates	yes	0
cacert	path to the certificate file describing the certificate authority of the SSL server	yes	/etc/mmc/pulse2/package-server/keys/cacert.pem
localcert	path to the SSL server private certificate	yes	/etc/mmc/pulse2/package-server/keys/privkey.pem

« mirror_api » section This section define options for the mirror_api api implementation (it assign mirrors and package_api to machines).

Available options for the "mirror_api" section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	/rpc

« user_package_api » section This section define options for the user_package_api api implementation (it assign package_api to users, it's used for the package edition permissions).

Available options for the "user_package_api" section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	/upaa

« scheduler_api » section This section define options for the scheduler_api api implementation (it assign a scheduler to each machine).

Available options for the "scheduler_api" section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	/scheduler_api
schedulers	The possible schedulers (can be a url or an id).	no	

« imaging_api » section This section define options for the imaging API.

Available options for the "imaging_api" section:

Option name	Description	Optional	Default value
mount_point	The API mount point	yes	/imaging_api
uuid	The package server UUID. You can use the uuidgen command to compute one.	no	
base_folder	Base folder where Pulse 2 imaging sub directories are contained.	yes	/var/lib/pulse2/imaging
bootloader_folder	Where bootloader (and bootsplash) is stored, relative to "base_folder"	yes	bootloader
cdrom_bootloader	The CD-ROM boot loader file. It is used to create bootable restoration CD/DVD.	yes	cd_boot
bootsplash_file	The imaging menu (GRUB menu) backgroung image, in XPM format.	yes	bootsplash.xpm
bootmenus_folder	Where boot menus are generated / being served, relative to "base_folder"	yes	bootmenus
diskless_folder	Where kernel, initrd and other official diskless tools are stored, relative to "base_folder"	yes	diskless
diskless_kernel	Name of the diskless kernel to run, relative to "diskless_folder"	yes	kernel
diskless_initrd	Name of the diskless initrd to boot (core), relative to "diskless_folder"	yes	initrd
diskless_initrdcd	Name of the diskless initrd to boot (add on to boot on CD), relative to "diskless_folder"	yes	initrdcd
diskless_memtest	Diskless memtest too, relative to "diskless_folder"	yes	initrdcd
diskless_dban	Diskless dban too, relative to "diskless_folder"	yes	initrdcd
inventories_folder	Where inventories are stored / retrieved, relative to "base_folder"	yes	inventories
computers_folder	Where additionnal material (hdmap, exclude) are stored / retrieved, relative to "base_folder"	yes	computers
masters_folder	Where images are stored, relative to "base_folder"	yes	masters
postinst_folder	Where postinst tools are stored, relative to "base_folder"	yes	postinst
archives_folder	Will contain archived computer imaging data, relative to "base_folder"	yes	archives
isos_folder	Will contain generated ISO images	yes	/var/lib/pulse2/imaging/isos
isogen_tool	tool used to generate ISO file	yes	/usr/bin/mkisofs
rpc_replay_file	File contained in "base_folder" where failed XML-RPC calls from the package server to the central MMC agent are stored.	yes	rpc-replay.pck
rpc_loop_timer	RPC replay loop timer in seconds. The XML-RPC are sent again to the central MMC agent at each loop.	yes	60
rpc_count	RPC to replay at each loop.	yes	10
rpc_interval	Interval in seconds between two RPCs	yes	2
uuid_cache_file	Your UUID cache *inside* "base_folder"	yes	uuid-cache.txt

« mirror:XX » section This section define options for the mirror api implementation.

Available options for the `mirror:XX` section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	
src	The root path of the package tree.	no	

« package_api_get:XX » section This section define options for the `package_api_get` API implementation.

Available options for the `package_api_get:XX` section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	
src	The root path of the package tree.	no	

« package_api_put:XX » section This section define options for the `package_api_put` API implementation.

Available options for the `package_api_put:XX` section:

Option name	Description	Optional	Default value
mount_point	The api mount point	no	/rpc
src	The root path of the package tree.	no	
tmp_input_dir	The directory where the data for package creation is put	yes	

Pulse 2 package server performance on win32 platforms

Using the default configuration, the service won't accept more than 64 concurrent TCP connections. The default event loop used by the Python Twisted library use the select() system call, which is limited to waiting on 64 sockets at a time on Windows.

Fortunately Twisted allows to choose another reactor instead of the default select() one. If sets to 1 in the package server configuration file, the `use_iocp_reactor` option lets Twisted runs with the IOCP reactor. IOCP (IO completions Ports) is a fast and scalable event loop system available on win32 platform. More informations are available in the Twisted documentation.

But there are some limitations:

- SSL is not supported (for the moment) by the IOCP reactor, so the package server can't be run with IOCP and SSL enabled at the same time,
- The IOCP reactor implementation from Twisted only works on win32 platform where the ConnectEx() API is available. So it won't works on Windows NT and Windows 2000 platforms.

Using the IOCP reactor, the package server can handle at least 300 parallel TCP connections, but more benchmarks need to be done to guess its limits.

MMC pkgs plugin configuration file

This document explains the content of the MMC pkgs plugin configuration file/

Introduction

The « pkgs » plugin is the MMC plugin in charge of the edition, removal and creation of packages in the Pulse2 package system.

The plugin configuration file is `/etc/mmc/plugins/pkgs.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2
```

```
[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now two sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	yes
user_package_api	Describe how to reach the User package API service	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	yes	0

« user_package_api » section This section is used to tell to the plugin where to find its User Package API service.

Available options for the "user_package_api" section:

Option name	Description	Optional	Default value
server	The service IP address	yes	127.0.0.1
port	The service TCP port	yes	9990
mountpoint	The service path	yes	/upaa
username	The name to use when we send XMLRPC call	yes	""
password	The password to use when we send XMLRPC call	yes	""
enablessl	SSL mode support	yes	1
verifypeer	use SSL certificates	yes	0
cacert	path to the certificate file describing the certificate authority of the SSL server	yes	""
localcert	path to the SSL server private certificate	yes	""

MMC pulse2 plugin configuration file

This document explains the content of the MMC pulse2 plugin configuration file.

Introduction

The « pulse2 » plugin is the MMC plugin in charge of the very generic part of pulse2 plugins.

The plugin configuration file is `/etc/mmc/plugins/pulse2.ini`.

Like all MMC related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

For now two sections are available in this configuration file:

Section name	Description	Optional
main	Mostly MMC related behaviors	yes
database	Describe how to reach the pulse2 mysql database	yes

« main » section This section is used to give directives to the MMC agent.

Available options for the "main" section:

Option name	Description	Optional	Default value
disable	Whenever use this plugin (or not)	yes	0
location	Set the name of the location manager (by default use the only component that can do that, ie the computer backend)	yes	

« database » section This section defines some global options.

Available options for the "database" section:

Option name	Description	Optional	Default value
dbdriver	DB driver to use	no	mysql
dbhost	Host which hosts the DB	no	127.0.0.1
dbport	Port on which to connect to reach the DB	yes	3306 (aka "default MySQL port")
dbname	DB name	no	pulse2
dbuser	Username to give while conencting to the DB	no	mmc
dbpasswd	Password to give while connecting to the DB	no	mmc
dbpoolrecycle	This setting causes the pool to recycle connections after the given number of seconds has passed	yes	60
dbpoolsize	The number of connections to keep open inside the connection pool	yes	5
dbsslenable	SSL connection to the database	yes	0
dbsslca	CA certificate for SSL connection	yes	
dbsslcert	Public key certificate for SSL connection	yes	
dbsslkey	Private key certificate for SSL connection	yes	

Pulse 2 Scheduler configuration file

This document explains the content of the configuration file of the scheduler service from Pulse 2.

Introduction

The « Scheduler » service is the Pulse 2 daemon in charge of reading the MSC database, dispatching commands over available launchers and writing results in the MSC database.

The main service configuration file is `/etc/mmc/pulse2/scheduler/scheduler.ini`.

Optionnaly, the database configuration may also be defined into `/etc/mmc/plugins/msc.ini`

Like all Pulse 2 related configuration file, its file format is INI style. The file is made of sections, each one starting with a « [sectionname] » header. In each section options can be defined like this: « option = value ».

For example:

```
[section1]
option1 = 1
option2 = 2

[section2]
option1 = foo
option2 = plop
```

Configuration file sections

The configuration file is splitted into several sections: Some sections describing the different available launchers may appear, their name must begin with `launcher_`.

- scheduler, daemon, database and logging,
- launchers declaration (describing the different available launchers may appear, their name must begin with `launcher_`).

`scheduler.ini` available sections:

Section name	Description	Optional
scheduler	Mostly scheduler related behaviors	*no*
daemon	Scheduler service related behaviors	yes
database	Scheduler database access	yes (see below)
launcher_XXX	A way to talk to launcher_XXX	*no*

All the other sections (loggers, handlers, ...) are related to Python language logging framework. See http://docs.python.org/lib/logging-config-fileformat.html.

« scheduler » section This section is used to give directives to the scheduler service.

Available options for the "main" section:

Option name	Description
id	This scheduler name, used to take the right jobs in the database.
active_clean_states	Declare which kind of unconsistant states should be fixed. States can be either 'run', 'stop', or both, co
analyse_hour	Once per day, at "analyse_hour" hour (HH:MM:SS), the scheduler will analyse the database, looking to
announce_check	To announce what we are currently try to do on client, for each stage. For example TRANFERT while t
awake_time	The scheduler will periodicaly awake (for exemple to poll the database), with this key a specific periodi
cacert	path to the certificate file describing the certificate authority of the SSL server
clean_state_time	The scheduler will periodicaly awake to hunt for unconsistant command states, with this key a specific
client_check	comma-separated list of <key>=<value> tokens to ask to the client; value (as part ot the 'target' table' r
checkstatus_period	The period of the loop in charge of checking the scheduler health

Option name	Description
dbencoding	The encoding to use when injecting logs into the MSC database.
enablessl	SSL mode support
initial_wait	The amount of seconds to wait for the system to be stabilized when starting.
initial_wait	Add a little randomness to some loops. Default value is .2, ie +/- 20 %
localcert	path to the SSL server private certificate
host	This scheduler listing binding IP address
lock_processed_commands	Locking system, use with caution ! The only reason to activate this feature is for systems under heavy l
loghealth_period	The period of the loop in charge of logging the scheduler health
max_command_time	Command max authorized time, used by the launcher
max_upload_time	Upload max authorized time, used by the launcher
max_slots	The max number of slot to use for all launchers
max_wol_time	WOL wait time
mg_assign_algo	The plugin the scheduler will use to assign a computer to a group. See doc.
mode	The scheduler way-of-giving-task-to-its-launchers (see doc).
password	The password to use when sending XMLRPC commands to this scheduler.
port	This scheduler listing TCP port.
preempt_amount	Starting with version 1.2.5, the scheduler will perform this amount of command at a time.
preempt_period	Starting with version 1.2.5, the scheduler will periodicaly perform commands, using this period.
resolv_order	The different means used to find a client on the network (see doc).
scheduler_path	The Scheduler main script location, used by scheduler-manager to start and daemonize the service.
server_check	see client_check for option formating, the main differente is that checks are done server-side, not client-
username	The name to use when sending XMLRPC commands to this scheduler.
verifypeer	SSL cert verirfication (if set to True, you will have to build and use a PKI)

« daemon » section This section sets the scheduler service run-time options and privileges.

Available options for the "daemon" section:

Option name	Description	Optional	Type	Default value
group	The scheduler service runs as this specified group.	yes	group	root
pidfile	The scheduler service PID, used by scheduler-manager to track the scheduler service.	yes	path	/var/run/pulse2
umask	The scheduler service umask defines the right of the new files it creates (log files for example).	yes	octal	0077
user	The scheduler service runs as this specified user.	yes	user	root
setrlimit	Resource usage limits to apply to the scheduler process, specified by a string of triplets (resource, soft limit, hard limit). See the Python documentation for more information	yes	string	

Example:

```
[daemon]
pid_path = /var/run/pulse2
user = mmc
group = mmc
umask = 0007
setrlimit = RLIMIT_NOFILE 2048 2048 RLIMIT_CORE 0 0
```

« database » section This section can either be defined in `scheduler.ini`, or in `msc.ini` (in that order).

This section is documented into the *MMC MSC plugin configuration file*.

« launcher_XXX » section This section define available launchers (one per launcher, "XXX" must be an integer). By default, no launcher is defined.

Available options for the "launcher_XXX" section:

Option name	Description	Optional	Type	Default value
enablessl	Flag telling if SSL mode should be used to connect to the launcher.	*no*	boolean	
host	The launcher IP address.	*no*	string	
password	The password to use when we send XMLRPC commands to this launcher.	*no*	string or base64	
port	The launcher TCP port.	*no*	string	
username	The name to use when we send XMLRPC commands to this launcher.	*no*	string	

CHAPTER 2

Other documentation

2.1 Development

This section explains how to develop new modules for the Mandriva Management Console (MMC).

2.1.1 Contributing to MMC with git

MMC source code is hosted on github: https://github.com/mandriva-management-console/mmc

Repo setup

1. Setup an account on github and fork https://github.com/mandriva-management-console/mmc.

   Then checkout your fork:

   ```
   git clone git@github.com:<USER>/mmc.git
   ```

2. Add a remote on mandriva-management-console/mmc

   ```
   git remote add mmc git@github.com:mandriva-management-console/mmc.git
   git remote update
   ```

3. Create a local branch that is tracking the main repository

   ```
   git branch master-mmc --track mmc/master
   ```

4. Change the default behavior of git push (if no ref is specified, use the tracked branch by default)

   ```
   git config --global push.default tracking
   ```

Using pull requests

We use github pull requests to review fixes and new features.

For each bugfix or new feature you will propose a pull request which can be merge directly in the MMC repository trough the github interface.

This means that you need to create and publish a branch for every fix or feature then ask a pull of these branches. Thanks to git, this is very easy.

The pull request commits must be clean and atomic.

Fixing a bug or developping a new feature in master

1. Update the master-mmc branch to make sure we are working with an up-to-date installation.

```
git checkout master-mmc
git pull
```

2. Create a local branch based on master-mmc

```
git checkout -b fix-blah-blah (or feature-blah-blah)
```

3. Fix the bug with one or two commits (each commit must be atomic)
4. Publish the branch to your github account

```
git-publish-branch (http://git-wt-commit.rubyforge.org/git-publish-branch)
```

5. Test your fix or feature ! (others can also test it by merging your branch since its public now)
6. If everything is fine, in the github interface, select the fix-blah-blah (or feature-blah-blah) branch and click on pull request.

Github will try to create a pull request on the master branch by default

In the pull request you can explain what your commit is fixing and how to reproduce the bug if needed.r

Commit format

You should type in the commit title the name of the module that is concerned by the commit. A commit usually fix or add a feature in a specific module (inventory, samba, etc...). If the commit target is not a single module you can use the project name (pulse2, core, mds).

```
<module|project>: <title> (70 chars max)
<blank line>
<detailed description>
```

If the commit is releated to an reported issue on http://projects.mandriva.org reference the issue number in the commit title or the commit body like this:

```
Fixes #1000
```

Will close the issue #1000 and make a relation between the commit and the issue in Redmine.

```
Refs #1000
```

Will just make a relation between the commit and the issue. The issue state won't be changed.

Commit directly in master

If you have the rights to commit to the main repository and you are sure of your fix, you might want to commit directly in the main repo directly instead of creating a pull request.

Use the following procedure to keep a clean history on the main branch (ie: no merge commits)

1. Always create a branch for working on your fix

```
git checkout master-mmc
git pull
git checkout -b fix-foo
```

2. Fix what you want to fix, test etc. When ready go to 3.
3. Update master-mmc to get latests commits

   ```
   git checkout master-mmc
   git pull
   ```

4. Rebase your branch on top of master-mmc

   ```
   git checkout fix-foo
   git rebase master-mmc
   ```

   This will apply all the commits you have made in fix-foo on top of the latest master-mmc history
5. Then merge your branch on master-mmc and push the changes to the main repository

   ```
   git checkout master-mmc
   git merge fix-foo
   git push
   ```

   Since all commits in fix-foo are on top of the master-mmc commits thanks to rebase, the merge will be done in fast forward mode and there will be no merge commit.

2.1.2 Writing MMC scripts

Following are a few examples of scripts that you can write using the MMC API.

This script adds a few users to the LDAP directory:

```
#!/usr/bin/env python

from mmc.plugins.base import ldapUserGroupControl

users = [('login1', 'passwd1', 'firstname1', 'lastname1'),
         ('login2', 'passwd2', 'firstname2', 'lastname2'),
         ('login3', 'passwd3', 'firstname3', 'lastname3'),
         # ...
         ]

l = ldapUserGroupControl()

for login, password, firstname, lastname in users:
    # Store user into LDAP
    l.addUser(login, password, firstname, lastname)
    # Change user "mail" attribute value
    l.changeUserAttributes(login, 'mail', login + '@example.com')
```

This script creates SAMBA users into the LDAP directory, with all needed mail attributes for Postfix delivery:

```
#!/usr/bin/env python

from mmc.plugins.samba import sambaLdapControl
from mmc.plugins.mail import MailControl

users = [('username', 'pass', 'name', 'lastname'),
        ('username2', 'pass2', 'name2', 'lastname2'),
        ]

l = MailControl()
s = sambaLdapControl()
```

```
# Add group 'allusers' to the LDAP
l.addGroup('allusers')
for login, password, firstname, lastname in users:
    # Create user into the LDAP
    l.addUser(login, password, firstname, lastname)
    # Add user to a group 'allusers'
    l.addUserToGroup('allusers', login)
    # Set user mail
    l.changeUserAttributes(login,'mail',login+'@domain')
    # Add user needed mail objectClass
    l.addMailObjectClass(login,login)
    # Set user mail quota
    l.changeUserAttributes(login,'mailuserquota','512000')
    # Set user mail alias
    l.changeUserAttributes(login,'mailalias','allusers')
    # Add all SAMBA related attributes to the user
    # The SAMBA account will log in with the given SAMBA password
    s.addSmbAttr(login, password)
    # Set user POSIX account password (set the userPassword LDAP field)
    l.changeUserPasswd(login,password)
    # Enable mail delivery for this user
    l.changeMailEnable(login, True)
```

2.1.3 How to write a python module for the MMC agent

Related documentations

- Full MMC Python API documentation.
- *Style guide for python code*
- Some basic Python / LDAP bindings documentation.

Creating a Python module

Each MMC agent module must be located in the `$PYTHONPATH/site-packages/mmc/plugins` directory.

When the MMC agent starts, it looks for all Python modules in this path, and tries to activate them.

Each MMC Python module must declare a function call "activate". This function should make all needed tests that ensures the module will works. This function returns True if all the tests are OK, else False. In the later case, the MMC agent will give up on this module, and won't export it on the network.

The following method must also be implemented

- getVersion: must return the MMC version of the Python module, which is the same then the MDS version number
- getApiVersion: must return the Python module API number
- getApiRevision: must return the SVN revision number

Here is a MMC Python module skeleton. For example `/usr/lib/python2.5/site-packages/mmc/plugins/modulename`

```
VERSION = "2.0.0"
APIVERSION = "4:1:3"
REVISION = int("$Rev$".split(':')[1].strip(' $'))
def getVersion(): return VERSION
```

```
def getApiVersion(): return APIVERSION
def getRevision(): return REVISION
def activate(): return True
```

A MMC Python module is in the Python language terminology a "package". So making a `__init__.py` file is required to make Python treats a directory as containing a package. Please read this section from the Python language tutorial to know more about Python packages system.

Python module configuration file

The module configuration file must be located into the `/etc/mmc/plugins/module_name.ini` file.

The configuration file should be read using a PluginConfig class instance. This class inherits from the `ConfigParser` class.

This configuration file must at least contains a "main" section with the "disable" option, telling if the module is disabled or not:

```
[main]
disable = 0
```

If the configuration file doesn't exist, or doesn't have the "disable" option, the module is by default considered as disabled.

```
from mmc.support.config import PluginConfig, ConfigException

class ModulenameConfig(PluginConfig):

    def setDefault(self):
        """
        Set good default for the module if a parameter is missing the
        configuration file.
        This function is called in the class constructor, so what you
        set here will be overwritten by the readConf method.
        """
        PluginConfig.setDefault(self)
        self.confOption = "option1"
        # ...

    def readConf(self):
        """
        Read the configuration file using the ConfigParser API.
        The PluginConfig.readConf reads the "disable" option of the
        "main" section.
        """
        PluginConfig.readConf(self)
        self.confOption = self.get("sectionname", "optionname")
        # ...

    def check(self):
        """
        Check the values set in the configuration file.
        Must be implemented by the subclass. ConfigException is raised
        with a corresponding error string if a check fails.
        """
        if not self.confOption: raise ConfigException("Conf error")

    def activate():
```

```
        # Get module config from "/etc/mmc/plugins/module_name.ini"
        config = ModulenameConfig("module_name")
        ...
        return True
```

Exporting Python module API

All methods defined in the Python module are exported by the MMC agent, and can be directy called using XML-RPC.

For example:

```
def activate():
    return True

# Module attribute can't be exported with XML-RPC
value = 1234

# This method will be exported
def func1(arg1A, arg1B):
    # ...
    return SomeClass().func1(arg1A, arg1B)

# This method will be exported too
def func2(arg2A, arg2B):
    # ...
    return SomeClass().func2(arg2A, arg2B)

# Class can't be exported with XML-RPC !
class SomeClass:
    def func1(self, argA, argB):
        # ...
        return "xxx"

    def func2(self, argA, argB):
        # ...
        return "zzz"
```

How to launch shell commands inside a Python module

As the MMC agent is written on top of Python Twisted, you can't use the dedicated standard Python modules (like commands or popen) to run shell commands. You must use the Twisted API, and write ProcessProtocol classes.

But we provide simple ProcessProtocol based functions to run a process, and get its outputs.

Blocking mode

In blocking mode, if we start a shell command, the twisted server will loop until a process terminates. Blocking mode should not be used for functions that can be called by XML-RPC, because they will completely block the server. The server won't process other requests until the blocking code is terminated.

But when using the MMC API in command line, it's simpler to use the blocking mode.

Here is an example:

```
# Import the shLaunch method
from mmc.support.mmctools import shLaunch
# Run "ls -l"
# shLaunch returns once the shell command terminates
proc = shLaunch("ls -l")
# Return shell command exit code
print proc.exitCode
# Return shell command stdout
print proc.out
# Return shell command stderr
print proc.err
```

Non blocking mode

Non blocking-mode should be used when a method called by XML-RPC may block. Basically, the method should not return the result, but a Deferred object attached to a callback corresponding to the result. The twisted reactor will process the deferred, send the result to the callback, and the callback will finally return the wanted result.

Here is an example:

```
# Import the shLaunchDeferred method
from mmc.support.mmctools import shLaunchDeferred

def runLs():
    def cb(shprocess):
        # The callback just return the process outputs
        return shprocess.exitCode, shprocess.out, shprocess.err
    d = shLaunchDeferred("ls -l")
    # shLaunchDeferred returns a Deferred() object
    # We add the cb function as a callback
    d.addCallback(cb)
    # We return the Deferred() object
    return d
```

For more explanation about Python Twisted and Deferred objects, please read this page.

To use the runLs function in a python script, without the XML-RPC server:

```
from twisted.internet import reactor, defer
from xxx import runLs

def printResult(ret):
    print ret
    reactor.stop()

d = runLs()
# runLs returns a deferred object, we add a callback that is just printing the result
d.addCallback(printResult)
reactor.run()
```

2.1.4 How to write a PHP module for the MMC web interface

Related documentations

- Full MMC PHP web interface documentation.

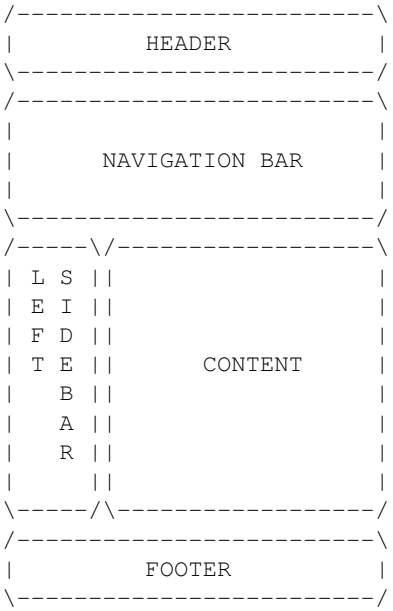
- *Style guide for PHP code*

MMC Page format

A MMC page is made of 5 elements:

- page header: expert mode button, disconnect button
- page footer: displays MMC components version
- top navigation bar: shows all available MMC sub-modules. A MMC module can offer more than on sub-modules. For example, the "base" module display the "Users" and "Groups" pane.
- left sidebar: shows all available actions inside a sub-modules.
- content: HTML content that allows a user to make an action (forms, button, etc.)

Here is a simple schema:



When writing a MMC web module, you can:

- defines new sub-modules (new panes) into the navigation bar
- defines new actions into the left sidebar
- set a content for each action

How MMC pages are displayed

The `/usr/share/mmc/main.php` file is the key.

Called without argument (e.g. http://127.0.0.1/mmc/main.php), the MMC portal page is displayed. When a user login into the interface, this is the first page that is displayed.

To display other pages, the following parameters must be given to this PHP scipt:

- module: the name of the module (top navigation bar pane) where the page is located
- submod: the name of the sub-module (left navigation bar pane) where the page is located

- action: the base name of the PHP script that displays the page

For example: http://127.0.0.1/mmc/main.php&module=base&submod=users&action=add will call the "add.php" script of the "users" sub-module of the "base" module.

PHP module structure

A PHP module of the MMC web interface is fully contained into the `/usr/share/mmc/modules/[module_name]` directory of a MMC installation.

This directory should looks like this:

```
.
|-- graph
|   |-- img
|   |   |-- ...
|   `-- submodule1
|       `-- index.css
|-- includes
|   |-- module-xmlrpc.php
|   |-- publicFunc.php
|-- infoPackage.inc.php
|-- submodule1
|   |-- page1.php
|   |-- page2.php
|   |-- page3.php
|   |-- ...
|   |-- localSidebar.php
|-- submodule2
|   |-- localSidebar.php
|   |-- ...
|-- locale
|-- fr_FR.utf8
|   `-- LC_MESSAGES
|       `-- module.po
|-- nb_NO.utf8
|   `-- LC_MESSAGES
|       `-- module.po
|-- ...
```

- infoPackage.inc.php: module declaration. See the section *Module declaration: infoPackage.inc.php*
- includes: where should be put module include files: module widgets, module XMLRPC calls, etc.
- includes/publicFunc.php: this file included by various MMC pages. For example, if the module allows to manage user LDAP fields, his file can be used when rendering the user edit page.
- graph: where should be stored all graphical elements: images (in graph/img), extra CSS, etc.
- submoduleN: owns all the pages of a submodule
- submoduleN/localSidebar: left sidebar of a submodule when displaying sub-module pages
- locale: owns the i18n internationalization files of the module, used by gettext.

Mapping between main.php arguments and modules

The main.php arguments are directly related to modules directory organization.

For example, when calling http://127.0.0.1/mmc/main.php&module=base&submod=users&action=add, the file `/usr/share/mmc/modules/base/users/add.php` is executed.

Module declaration: infoPackage.inc.php

This mandatory file defines:

- the module name and description
- the sub-modules name, description, and their corresponding icons into the top navigation bar
- all the available module web pages, their names and their options
- form input fields that are protected by the ACL system

These informations are also used by the MMC home page to display the module summary.

Commented example:

```
<?php

# Register a new module called "module1"
$mod = new Module("module1");
# MMC module version, should follow MDS version release
$mod->setVersion("2.0.0");
# SVN revision bumber
$mod->setRevision("$Rev$");
# module description. The _T("") syntax will be explained later
$mod->setDescription(_T("Module 1 service"),"module1");
/*
Module API version this version can use.
The MMC agent Python module and the web interface PHP module
API version must match.
*/
$mod->setAPIVersion("4:1:3");
/* Register a new sub-module */
$submod = new SubModule("submodule1");
/* Set submodule description */
$submod->setDescription(_T("Sub module 1", "module1"));
/*
Icons to use in the top navigation bar for this sub-module.
The following images will be displayed:
- /usr/share/mmc/modules/module1/graph/img/submodule1.png: sub-module not selected
- .../submodule1_hl.png: mouse hover on the sub-module icon (the icon is highlighted)
- .../submodule1_select: the sub-module is selected
*/
$submod->setImg("modules/module1/graph/img/submodule1");
/*
The page to load when selecting the sub-module
e.g.: main.php?submod=module1&submod=submodule1&action=index
*/
$submod->setDefaultPage("module1/submodule1/index");
/* Sub-module priority in the top navigation bar */
$submod->setPriority(300);
/* Register pages in this sub-module */
/*
This new page will be displayed when using this URL:
e.g.: main.php?submod=module1&submod=submodule1&action=index
The corresponding PHP file will be: /usr/share/mmc/modules/module1/submodule1/index.php
A page must be registered to be displayed.
```

```
*/
$page = new Page("index", _T("Sub-module index page", "module1"));
/* Add this page to the sub-module */
$submod->addPage($page);
/* Another page */
$page = new Page("edit",_T("Sub-module edit page", "module1"));
/*
Options can be set on pages.
If "visible" is set to False, the page won't be displayed in the sub-module summary on the MMC home p
*/
$page->setOptions(array("visible"=>False));
/* A page can contain tabs. These tabs must be declared to get ACL support on them */
$page->addTab(new Tab("tabid1", "Tab description 1"));
$page->addTab(new Tab("tabid2", "Tab description 2"));
$submod->addPage($page);
/* Add the sub-module to the module */
$mod->addSubmod($submod);
/* Defines other submodules and pages */
$submod = new SubModule("submodule2");
...
...
/* And put the module into MMC application */
$MMCApp = &MMCApp::getInstance();
$MMCApp->addModule(&$mod);

?>
```

The following options can be set on a page:

- visible: if set to False, the page won't be displayed in the sub-module summary on the MMC home page. Always True by default.
- noHeader: If set to True, the header and the footer won't be inserted automatically when rendering the page. This option is useful for popup page and AJAX related pages. False by default.
- noACL: If set to True, no ACL entry is linked to this page. False by default.
- AJAX: same as setting noACL to True and noHeader to true. Always use this for URL that will be called by scriptaculous Ajax.Updater objects. False by default.

How to render a page

Once a page is registered into the infoPackage.php file, it can be rendered. The main.php script take care of this:

- It checks that the current user has the rights to see the page. If not, the user is redirected to the MMC home page
- If page noHeader option is set to False, the MMC header is rendered
- The registered PHP script corresponding to the page is executed
- If page noHeader option is set to False, the MMC footer is rendered

Notice that only the header and the footer can be rendered automatically. The top navigation bar, the left sidebar and the page content must be provided by the registered PHP script.

Notice that for special page like the popup, there is no need of header, footer and bars, only a content should be provide.

The PageGenerator class

This class allows to easily creates a page with the top navigation bar and the left sidebar. Here is a commented example of a simple MMC page:

```
<?php

/* localSidebar.php contains the left sidebar elements of all the pages sub-module. See next section
require("localSidebar.php");
/*
Display the top navigation bar, and prepare the page rendering.
The current sub-module pane is automatically selected.
*/
require("graph/navbar.inc.php");
/*
Create a page with a title
The title will be displayed as a H2
*/
$p = new PageGenerator(_T("Simple page example"));
/*
$sidemenu has been defined in the localSidebar.php file
We set it as the page left side bar
*/
$p->setSideMenu($sidemenu);
/*
We ask to the PageGenerator instance to render.
The page title and the left sidebar are displayed.
The current page corresponding pane is automatically selected in the left side bar.
*/
$p->display();
/* Fill the page with content */
...

?>
```

The SideMenu and SideMenuItem classes

The SideMenu class allows to build the left sidebar menu of a page. Here is an example, that could have been the content of the "localSidebar.php" of the previous section.

```
<?php

$sidemenu = new SideMenu();
/*
CSS class name to use when rendering the sidebar.
You should use the sub-module name
*/
$sidemenu->setClass("submodule1");
/*
Register new SideMenuItem objects in the menu.
Each item is a menu pane.
*/
$sidemenu->addSideMenuItem(new SideMenuItem(_T("Simple page"),
    "module1", "submodule1", "index", "modules/module1/graph/img/module1_active.png",
    "modules/module1/graph/img/module1_inactive.png")
);
$sidemenu->addSideMenuItem(new SideMenuItem(_T("Another page"),
```

```
    "module1", "submodule1", "add", "modules/module1/graph/img/module1_active.png",
    "modules/module1/graph/img/module1_inactive.png")
);

?>
```

The SideMenuItem constructor arguments are

- the item label
- the next three arguments are needed to create the URL link so that clicking on the menu item loads the right page. They corresponds to a module name ("module1"), a sub-module name ("submodule1"), and a registered page ("index").
- the last two optional arguments allow to define an icon to use when the sidemenu item is selected, and when not selected. If not specified, no icon will be used.

Adding page from a module to another module

With the `infoPackage.inc.php` file, you can also add the page of a module to another module. This is useful if you want to provide new features to an already existing module.

In our example, we add a new page to the "computers" sub-module of the "base" module. Here is the corresponding infoPackage.inc.php:

```
<?php

/* Get the base module instance reference */
$base = &$MMCApp->getModule('base');
/* Get the computers sub-module instance reference */
$computers = &$base->getSubmod('computers');
/* Add the page to the module */
$page = new Page("extrapage", _T("Extra page", "module1"));
$page->setFile("modules/module1/extra/extrapage.php");
$computers->addPage($page);
/* You should unset the references when you finished using them */
unset($base);
unset($computers);

?>
```

With this code, the PHP script `modules/module1/extra/extrapage.php` will be called when using the `main.php?module=base&submod=computers&action=extrapage`.

The remaining problem is the sidebar management. In the called PHP script, you must include the `localSidebar.php` script from the other sub-module module, and add your SideMenuItem object to it.

For example:

```
<?php
require("modules/base/computers/localSidebar.php");
require("graph/navbar.inc.php");
$p = new PageGenerator(_T("Extra page with new functions"));
/* Add new sidemenu item \*/
$sidemenu->addSideMenuItem(new SideMenuItem(_T("Extra page"),"base",
    "computers", "extrapage", "modules/base/graph/img/computers_active.png",
    "modules/base/graph/img/computers_inactive.png")
);
$p->setSideMenu($sidemenu);
```

```
$p->display();
...
?>
```

Including CSS file

When a page is rendered, the framework includes the file `modules/currentmodule/graph/currentmodule/currentsubm` if it exists.

"currentmodule" and "currentsubmodule" are guessed from the current URL.

The MMC widget framework

The MMC widget framework is a set of classes that allows to wrap HTML code into PHP classes. The goal of this very simple framework is:

- separate HTML code and PHP code
- factorize HTML and PHP code
- use the same set of widgets accross all the module interface, for a better user experience

There are two kinds of widgets: widgets that contains other widgets, widgets that doesn't contain other widgets.

MMC widgets that are containers inherits from the HtmlContainer class, and the other widgets inherits from the HtmlElement.

Every MMC pages have been built using instances of these classes. Here is a little example:

```
<?php
/* Build a new validating form */
$f = new ValidatingForm();
/* Push a table into the form, and go to the table level */
$f->push(new Table());
/* Add two TR to the table */
/* Ask for a given name */
$f->add(
new TrFormElement(_T("Given name"), new InputTpl("givenName"),
array("value" => "", "required" => True)
);
/* Ask for a family name */
$f->add(
new TrFormElement(_T("Family name"), new InputTpl("name"),
array("value" => "", "required" => True)
);
/* Go back to the validating form level */
$f->pop();
/* Add a button to the form */
$f->addButton("bvalid", _T("Validate"));
/* Close the form */
$f->pop();
/* Render all the form and the objects it contains \*/
$f->display();
?>
```

This example renders a HTML form, with two input fields asking for a given name and a family name.

In this example, ValidatingForm and Table are two HtmlContainer sub-classes. TrFormElement and InputTpl are two HtmlElement sub-classes.

HtmlContainer objects

A HtmlContainer object owns an ordered list of elements. An element is either an instance from a HtmlContainer sub-class, either an instance from a HtmlElement sub-class.

This list of elements is either opened (new elements can be added to the list), either closed (no more elements can be added).

When adding a HtmlElement or a HtmlContainer object to a HtmlContainer, the object is added to the last added HtmlContainer which does not have a closed element list.

The HtmlContainer class main methods are:

- push($newHtmlContainer): recursively push into the widget element list a new container
- pop(): pop the last pushed HtmlContainer with an opened element list, and close the list.
- add(NewHtmlElement): recursively add into the widget element list a new element
- display(): recursively render HTML code. The display method is called on each element of the list.

Here is an example. The indentation helps to show which container is used:

```
<?php
$o = new HtmlContainer;
$o->add(HtmlElement());
$o->push(HtmlContainer());
/* The HtmlElement are added to the latest added and open HtmlContainer */
$o->add(HtmlElement());
$o->push(HtmlContainer());
/* The HtmlElement are added to the latest added and open HtmlContainer */
$o->add(HtmlElement());
$o->add(HtmlElement());
/* closing the element list of the latest HtmlContainer */
$o->pop();
/* falling back to the previous HtmlContainer */
$o->add(HtmlElement());
/* closing the element list of the latest HtmlContainer */
$o->pop();
$o->add(HtmlElement());
/* Popping the root container */
$o->pop();
/* Display the HTML code */
$o->display();
?>
```

To render HTML code, a HtmContainer subclass needs only to implement these two functions:

- begin: before recursivelly calling display() on each element of its list, the container must put its starting HTML tag. This method returns the HTML tag as a string.
- end: After recursivelly calling display() on each element of its list, the container must put its ending HTML tag. This method returns the HTML tag as a string.

Here is an example of a HtmlContainer subclass that wraps a HTML table:

```
<?php

class Table extends HtmlContainer {
    function Table() {
        $this->HtmlContainer();
    }
```

```
    function begin() {
        return "<table>";
    }

    function end() {
        return "</table>";
    }
}

?>
```

HtmlElement objects

These objects are very simple PHP class wrapper around HTML code, and can be stored into a HtmlContainer object.

To render HTML code, a HtmElement subclass needs only to implement the display() function. This function just prints the HTML code implementing the widget. For example:

```
<?php

class Title Extends HtmlElement {

    function Title($text) {
        $this->$text = $text
    }

    function display() {
        print "<h1>" . $this->text . "</h1>";
    }
}

?>
```

Useful MMC widgets

The following widgets are defined in the `includes/PageGenerator.php` file.

The ListInfos class

The ListInfos class allows to create a paged multi-column table with a navigation bar, and to link each row to a set of actions. For example, the MMC user list is implemented using a ListInfos widget.

Here is an example. We create a table with two columns: the first is a fruit, the second is a quantity.

```
<?php

require ("includes/PageGenerator.php");
$fruits = array("apple", "banana", "lemon", "papaya", "fig", "olive",
    "clementine", "orange", "mandarin", "grapes", "kumquat");
$stock = array("5", "8", "40", "12", "40", "51", "12", "7", "9", "15", "21");
/*
Create the widget. The first column will be labeled "Fruit name",
and each cell will contain an item of the $fruits array.
*/
$l = new ListInfos($fruits, _T("Fruit name"));
```

```
/* Add the second column */
$l->addExtraInfo($stock, _T("Quantity"));
/*
Set the item counter label.
The counter is displayed just above the table:
Fruits 1 to 10 - Total 11 (page 1/2)
*/
$l->setName(_T("Fruits"));
/* Display the widget */
$l->display();

?>
```

The item counter label is displayed just above the table. In our example, it shows: Fruits 1 to 10 - Total 11 (page 1/2). It means:

- Fruits 1 to 10: from all table rows, the row #1 to row #10 are displayed. By default, the ListInfos widget is configured to display only 10 rows. This setting is set into the "maxperpage" option of the `/etc/mmc/mmc.ini` file.
- Total 11: the total table rows number
- (page 1/2): the first page, that corresponds to the first 10 rows of the table, is displayed. If you click on the "Next" button, the second page will be displayed, with the single row #11.

Now we are going to add some action items to each rows:

```
<?php

require ("includes/PageGenerator.php");
$l = new ListInfos($fruits, _T("Fruit name"));
$l->addExtraInfo($stock, _T("Quantity"));
$l->setName(_T("Fruits"));
/* Add actions */
$l->addActionItem(new ActionItem(_T("View fruit"), "view", "display", "fruit"));
$l->addActionItem(new ActionPopupItem(_T("Delete fruit"), "view", "delete", "fruit"));
$l->display();

?>
```

Thanks to addActionItem, we add to each row two actions: view the fruit, and delete the fruit.

ActionItem constructor arguments are:

- action label ("View fruit"), displayed when the mouse hover on the action icon
- the web page ("view") of the current sub-module to use to perform the action These
- the CSS class ("display") to use to set the action icons
- the URL parameter name ("fruit") used to give to the web page that will perform the action the object. The content of the first row is always used as the parameter value.

In our example, the URL link for the first row will be: `main.php?module=module1&submod=submodule1&action=view&f`
For the second row, "`...&fruit=banana`", etc.

Sometimes an action link needs to send the user to another module or submodule, instead of the current one. To do this, you add these parameters to the ActionItem constructor:

- $module: the module part of the URL link
- $submod: the sub-module part of the URL link

- $tab: the tab part of the URL link (if the link goes to a specific tab of a widget

ActionPopupItem displays a little popup page when clicked. This is useful for actions that just need an extra validation to be performed.

When there are actions, the first column cells are automatically linked to the first action. But this can be disabled with:

```
<?php

$l->disableFirstColumnActionLink();

?>
```

The default size of the JavaScript popup window is 300 pixel. This can be changed like this:

```
<?php

$p = new ActionPopupItem(_T("Delete fruit"), "view", "delete", "fruit");
$p->setWidth(500); /* Size is now 500 px */
$l->addActionItem($p);

?>
```

Conditional actions

With the addActionItem method, you add an action to every row of a ListInfos widget. In some cases, an action can't be performed for a specific row, so you don't want the action link to be available.

The addActionItemArray method allows to pass to the ListInfos widget an array of actions to display:

```
<?php

require ("includes/PageGenerator.php");
$fruits = array("apple", "banana", "lemon", "papaya", "fig", "olive",
    "clementine", "orange", "mandarin", "grapes", "kumquat");
$stock = array("5", "8", "40", "12", "40", "51", "12", "7", "9", "15", "21");
$viewAction = new ActionItem(_T("View fruit"), "view", "afficher", "fruit");
$deleteAction = new ActionPopupItem(_T("Delete fruit"), "view", "supprimer", "fruit");
/* an EmptyActionItem will be displayed as a blank space */
$emptyAction = new EmptyActionItem();
$actionsView = array();
$actionsDel = array();
foreach($stock as $value) {
    if ($value < 10) {
        /* Only put the deleteAction link if value is lower than 10 */
        $actionsDel[] = $deleteAction;
        $actionsView[] = $emptyAction;
    } else {
        /* else only put the viewAction link */
        $actionsView[] = $viewAction;
        $actionsDel[] = $emptyAction;
    }
}
$l = new ListInfos($fruits, _T("Fruit name"));
$l->addExtraInfo($stock, _T("Quantity"));
$l->setName(_T("Fruits"));
$l->addActionItemArray($actionsView);
$l->addActionItemArray($actionsDel);
$l->display();
```

```
?>
```

Ajaxified ListInfos

A ListInfos widget content can be dynamically filtered.

First, we write the page that render the ListInfos widget. This page gets the filter to apply to the ListInfos widget as a GET parameter. Here is the code of `/usr/share/mmc/modules/module1/submodule1/ajaxFruits.php`:

```
<?php

$filter = $_GET["filter"];
$fruits = array("apple", "banana", "lemon", "papaya", "fig", "olive",
    "clementine", "orange", "mandarin", "grapes", "kumquat");
/* Make a fruit list using the filter */
$filtered = array();
foreach($fruits as $fruit) {
    if ($filter == "" or !(strpos($fruit, $filter) === False))
        $filtered[] = $fruit;
}
$l = new ListInfos($filtered, _T("Fruit name"));
/*
Instead of using the standard widget navigation bar, use the AJAX version.
This version allows to keeps the filter when clicking on previous / next.
*/
$l->setNavBar(new AjaxNavBar(count($filtered), $filter));
$l->setName(_T("Fruits"));
$l->display();

?>
```

This PHP code just displays a ListInfos widget where the elements are filtered.

Now we create a page where the ListInfos widget is automatically updated using a filter. Here is the code of `/usr/share/mmc/modules/module1/submodule1/index.php`:

```
<?php

require("localSidebar.php");
require("graph/navbar.inc.php");
/*
Create the filtering form with a input field, and bind this input field
to an AJAX updater that will use the specified URL to dynamically fill
in a DIV (see below) container.
*/
$ajax = new AjaxFilter(urlStrRedirect("module1/submodule1/ajaxFruits"));
/* You can ask the AJAX updater to be called every 10s */
$ajax->setRefresh(10000);
$ajax->display();
/* Set page title and left side bar */
$p = new PageGenerator(sprintf(_T("Fruits"), "module1"));
$p->setSideMenu($sidemenu);
$p->display();
/* Display the DIV container that will be updated */
$ajax->displayDivToUpdate();
```

```
?>
```

In `infoPackage.inc.php`, these two PHP scripts should be registered like this:

```
<?php

$mod = new Module("module1");
...
$submod = new SubModule("submodule1");
...
/* Register the first page */
$page = new Page("index", _T("Fruit list", "module1"));
$submod->addPage($page);
/* Register the page called using the AJAX DIV updater */
$page = new Page("ajaxFruit");
$page->setFile("modules/module1/submodule1/ajaxFruits.php",
    array("AJAX" => True, "visible" => False)
);
$submod->addPage($page);
...

?>
```

The ValidatingForm widget

This widget (a subclass of HtmlContainer) is a HTML form with input fields validation. The form can't be validated (POSTed) if some required fields are not filled in, or if their values don't match a given regex.

A lot of MMC pages display a HTML form, containing a HTML table with multiple rows of a single labeled input field. Here is an example

```
<?php

/* Build a new validating form */
$f = new ValidatingForm();
/* Push a table into the form, and go to the table level */
$f->push(new Table());
/* Add two TR to the table */
/* Ask for a given name */
$f->add(
    new TrFormElement(_T("Given name"), new InputTpl("givenName"),
        array("value" => "", "required" => True)
    )
);
/* Ask for a family name */
$f->add(
    new TrFormElement(_T("Family name"), new InputTpl("name"),
        array("value" => "", "required" => True)
    )
);
/* Go back to the validating form level */
$f->pop();
/* Add a button to the form */
$f->addButton("bvalid", _T("Validate"));
/* Close the form */
$f->pop();
/* Render all the form and the objects it contains */
```

```
$f->display();

?>
```

The TrFormElement class creates objects that will render a HTML row (a TR) with two columns (two TDs). The first column contains a describing label, and the second column an input field. In the example:

```
<?php
/* Ask for a given name */
$f->add(
    new TrFormElement(_T("Given name"), new InputTpl("givenName"),
        array("value" => "", "required" => True)
    )
);

?>
```

TrFormElement takes three argument:

- "Given name" is the label of the input field.
- InputTpl("givenName") is a standard HTML input field, with "givenName" as the HTML "name" attribute.
- array("value" => "", "required" => True) is an array of option for the InputTpl object. "value" => "" means the HTML "value" attribute of the input field is empty. "required" => True means that the form can't be posted if the input field is empty.

See next section about all the InputTpl widget options.

The InputTpl based widgets

The InputTpl class allows to render a standard HTML input field. The constructor takes two arguments:

- $name: the value of the "name" attribute of the INPUT HTML field
- $regexp: a regexp that must be matched by the input field, else the HTML form won't be posted. The regexp is used only if the input field is inserted into a ValidatingForm object. If not given, the default regexp is "/.+/".

When rendering the widget, additional options can be given to the "display" method thanks to an array:

- "value": an empty string by default. That's the input field value.
- "required": False by default. If set to true and the InputTpl object is inside a ValidatingForm object, the form can't be posted if the field is empty

A lots of class that inherits from InputTpl have been written. For example: MACInputTpl is an HTML input field that only accepts MAC address, NumericInputTpl only accepts numeric value. Theses kind of classes are very easy to write:

```
<?php

class NumericInputTpl extends InputTpl {
    function NumericInputTpl($name) {
        $this->name = $name;
        $this->regexp = '/^[0-9]*$/';
    }
}
class MACInputTpl extends InputTpl {
    function MACInputTpl($name) {
        $this->name = $name;
        $this->regexp = '/^(\[0-9a-f]{2}:){5}[0-9a-f]{2}$/i';
```

```
    }
}

?>
```

The PopupForm widget

This widget allows to build a MMC popup form triggered by a ActionPopupItem very quickly. For example:

```
<?php

if (isset(_POST["bdel"])) {
    /* action to remove the fruit */
    ...
} else {
    $fruit = urldecode($_GET["fruit"]);
    /* Create the form and set its title */
    $f = new PopupForm(_T("Delete a fruit"));
    /* Add a little description text */
    $f->addText(_T("This action will delete all the fruit"));
    /*
    Put a hidden input field into the form.
    The HiddenTpl is explained later in this document
    */
    $hidden = new HiddenTpl("fruit");
    /* Add this field to the form */
    $f->add($hidden, array("value" => $fruit, "hide" => True));
    /* Add validation and cancel buttons */
    $f->addValidateButton("bdel");
    $f->addCancelButton("bback");
    $f->display();
}

?>
```

The NotifyWidgetSuccess and NotifyWidgetFailure class

These two widgets displays a javascrip popup with a message, with a OK button.

```
<?php

/* Error message popup */
new NotifyWidgetFailure(_T("Error ! /o\"));
/* Success */
new NotifyWidgetSuccess(_T("Reboot was successful ! \o/"));

?>
```

Creating page with tabs

This widget allows to include a tab selector that displays a page when clicking on a tab.

For example:

```php
<?php

require("localSidebar.php");
require("graph/navbar.inc.php");
/* We use the TabbedPageGenerator class */
$p = new TabbedPageGenerator();
/* Set the sidemenu, as the PageGenerator class */
$p->setSideMenu($sidemenu);
/*
Not required: you can add some content above the tab selector
The content is a title, and a PHP file to include.
*/
$p->addTop("Page title", "modules/module1/submodule1/top.php");
/*
Now we add new tab to the tab selector.
Each tab is associated to an id, a tab title, a page title, and a PHP file to include.
*/
$p->addTab("tab1", "Tab 1 title", "Page 1 title", "modules/module1/submodule1/tab1.php");
$p->addTab("tab2", "Tab 2 title", "Page 2 title", "modules/module1/submodule1/tab2.php");
$p->addTab("tab3", "Tab 3 title", "Page 3 title", "modules/module1/submodule1/tab3.php");
/*
You can add a fifth argument, which is an array of URL parameters
that will be used when building the URL link of the tab.
*/
$p->addTab("tab4", "Tab 4 title", "Page 4 title", "modules/module1/submodule1/tab4.php",
    array("uid" => "foo")
);
$p->display();

?>
```

If no tab is selected, the first tab is automatically activated.

To build a tab URL link, the current module, submodule and action are used, with the given tab id and the given array of URL parameters. For example:

```php
<?php

$p->addTab("tab4", "Tab 4 title", "Page 4 title", "modules/module1/submodule1/tab4.php",
    array("uid" => "foo")
);

?>
```

will build this link: `module=currentmod&submod=currentsubmod&action=currentaction&tab=tab4&uid=foo`

2.1.5 Internationalization and localization

The MMC uses the GNU gettext system to produce multi-lingual messages. If you are not famliar with GNU gettext, please read the GNU gettext manual.

Two special PHP methods are needed to translate the interface:

- _($msg): the underscore is a PHP alias for the gettext($msg) method. The gettext method looks up a message in the current text domain. The default text domain is the one from the MMC "base" module. In other words, the _("$msg") method can be only used to translate strings from the MMC "base" module.
- _T($msg, $module): this function looks up a message for a given module. So if you create MMC web module called "module1", to translate a message you write:

```
echo _T("This is a message to translate", "module1");
```

As the module name is already in the URL to be displayed (see *How MMC pages are displayed*, if you don't specify a module name it can be automatically guessed.

2.1.6 Style guide for python code

Coding conventions for the python code of all MMC components

Introduction

A lot of MMC components are written in Python, among them the MMC agent and its python plugins.

This document sets the coding conventions for the Python code of all MMC components.

This document is totally based on Guido Van Rossum "Style Guide for ython Code" document (see http://www.python.org/dev/peps/pep-0008/): you must read it too. This document only emphases on important coding conventions.

Code layout

Indentation: use 4 spaces per indentation level, no tabs allowed. It's ok with Emacs Python mode.

Encoding: the source code must always use the UTF-8 encoding.

Whitespace in Expressions and Statements

```
Yes: spam(ham[1], {eggs: 2})
No:  spam( ham[ 1 ], { eggs: 2 } )
Yes: if x == 4: print x, y; x, y = y, x
No:  if (x == 4): print x, y; x, y = y, x
No:  if x == 4 : print x , y ; x , y = y , x
Yes: spam(1)
No:  spam (1)
Yes: dict['key'] = list[index]
No:  dict \['key'] = list \[index]
Yes:
x = 1
y = 2
long_variable = 3
No:
x             = 1
y             = 2
long_variable = 3
```

Naming conventions

Module name: short, lowercase names, without underscores

Class Names: CapitalizedWords

Functions Names: mixedCase for instance method, lower_case_with_underscores for other.

Constants: UPPER_CASE_WITH_UNDERSCORE

Comments

They are written in english.

They always start with a capitalized first word.

There is always a space between the # and the begin of the comment.

Docstrings

All modules, functions and classes must have a docstring.

The docstring must be written in the Epytext Markup Language format. We use epydoc to generate the API documentation. See http://epydoc.sourceforge.net/epytext.html and http://epydoc.sourceforge.net/fields.html for more information.

The recommanded epydoc fields are:

```
def foo(a, b, c):
    """
    This methods performs funny things.
    @param a: first parameter of foo
    @type a: int
    @param b: second parameter of foo
    @type b: str
    @param c: third parameter of foo
    @type c: unicode
    @raise ExceptionFoo: raised if b == 'bar'
    @rtype: int
    @return: the result should be 42
    """
```

Remarks:

- Sometimes the method description can be written in @return if the function is simple.
- If you skip @param because the parameter name seems really explicit to you, use at least: @rtype and @return
- Please use a spellchecker for your docstrings

Python module import rules

`from mod import *` is forbidden, because it doesn't allow us to track module dependencies effectively.

The import order should be:

```
# Import standard python module
import os
import sys
# Import external modules (SQLAlchemy, Twisted, python-ldap, etc.)
from sqlalchemy.orm import create_session
# Import internal modules
from mmc.plugins.base import ...
```

SQLAlchemy code convention

Querying with the ORM

Here are the recommended code guidelines when querying using the ORM:

- First select the objects you want as a result:

  ```
  results = session.query(Table1).add_entity(Table2).add_entity(...)
  ```

  If your query will return more than one row, please call the query "results", or "rows". If you are querying for one object only, please use a variable name corresponding to this object.

- Then if needed perform a join between the tables. It is usually done using join in a select_from expression

  ```
  .select_from(table1.join(table2).join(...))
  ```

- Then add filter expressions to filter down the query:

  ```
  .filter(Table1.num == 42)
  .filter(Table2.num == -42)
  ```

  Please use "Table1.num" instead of "table1.c.num", because it's more pythonish.

- At least add the query limit:

  :: .all() # .first() .one(), or count()

Here is the complete query code:

```
results = session.query(Table1).add_entity(Table2).add_entity(...)
.select_from(table1.join(table2).join(...))
.filter(Table1.num == 42)
.filter(Table2.num == -42)
.all()
# Also accepted
results = session.query(Table1).add_entity(Table2).add_entity(...)
select_from(table1.join(table2).join(...))
filter(Table1.num == 42)
filter(Table2.num == -42)
all()
# Also accepted
results = session.query(Table1).add_entity(Table2).add_entity(...)
results = results.select_from(table1.join(table2).join(...))
results = results.filter(Table1.num == 42)
results = results.filter(Table2.num == -42)
results = results.all()
```

If you're looking for one result only (e.g. to get the properties of an object or check its existence) please use "one()" instead of "first()". "one()" will raise an exception if no object or more than one objects if returned, and so it forces you to deal with the exception.

Tools to check Python code

Use the `pyflakes` tool to check your code. The code must be fixed if these messages are displayed:

- "import * used; unable to detect undefined names"
- "'x' undefined variable"
- "'x' imported but unused"

Python language version compatibility

The code must be compatible with Python 2.5. That's a rather old version, but we never had any problems that forced us to use a newer version.

Python additional library compatibility

The code must be compatible with these library versions:

- Python Twisted: 8.1.0
- Python LDAP: 2.0
- Python SQLAlchemy: 0.5

Python code copyright header

Here is the header that must be used:

```
# -*- coding: utf-8; -*-
#
# (c) 2004-2007 Linbox / Free&ALter Soft, http://linbox.com
# (c) 2007-2011 Mandriva, http://www.mandriva.com
#
# This file is part of Mandriva Management Console (MMC).
#
# MMC 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.
#
# MMC 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 MMC.  If not, see <http://www.gnu.org/licenses/>.
```

2.1.7 Style guide for PHP code

Coding conventions for the PHP code of all MMC components.

Introduction

This document sets the coding conventions for the PHP code of all MMC components (like the MMC web interface for example).

Convention from http://pear.php.net/manual/en/standards.php apply too.

Code layout

Indentation: use 4 spaces per indentation level, no tabs allowed.

Encoding: the source code must always use the UTF-8 encoding.

Code indentation and organisation

block “for”, “function”, “switch”, “if”, etc... always end with opening braces on the same line.

```
<?php

if ($val == value) {
    echo $val;
} else {
    return -1;
}
foreach ($arrParam as $singleItem) {
    print $singleItem;
}

?>
```

Function with long args (more than one line size)

```
<?php

myFunction($value1,
    $value2,
    $morevalue4,
    $val5);

?>
```

Comments

They are written in english.

They always start with a capitalized first word.

There is always a space between the // and the begin of the comment. // and /* are fine. Don’t use #.

All functions must have a correct doxygen header.

Naming conventions

- ClassName : CapitalizedWords
- functionName : mixedCase for all function name
- _membersValue : member value of a class begin with a “_”

PHP language version compatibility

The code must be compatible with PHP 5.0.

PHP error reporting level

All possible PHP errors, warnings and notices must be fixed in the PHP code. Use these lines in your `php.ini` file when working on the code to find them all:

```
error_reporting = E_ALL
display_errors = On
```

PHP code copyright header

Here is the header that must be used:

```
/**
 * (c) 2004-2007 Linbox / Free&ALter Soft, http://linbox.com
 * (c) 2007-2011 Mandriva, http://www.mandriva.com
 *
 * This file is part of Mandriva Management Console (MMC).
 *
 * MMC 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.
 *
 * MMC 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 MMC.  If not, see <http://www.gnu.org/licenses/>.
 */
```

2.1.8 MMC projects release guidelines

This document explains how to release a new mmc project (mmc-core, mds, pulse2).

Release components

What we release is a single tarball by project called:

- mmc-core-VERSION.tar.gz
- mds-VERSION.tar.gz
- pulse2-VERSION.tar.gz

This tarballs contains:

- the mmc-agent, the core MMC modules (audit framework), the core plugins “base” and “ppolicy” and the MMC web interface framework, with the “base” and “ppolicy” web modules.
- MDS modules (samba, network, sshlpk, mail, bulkimport, userquota...) python and web parts
- Pulse 2 modules (inventory, msc, dyngroup, pkgs...) and services (inventory-server, package-server, imaging-server, scheduler, launcher...)

Preparing a new release

First a release candidate (RC) should be generated to prepare packages and do QA tests.

1. Bump the version of the project

If the current stable version is 1.1.0 and we want to release 1.2.0 bump the version to 1.1.90. This will be the first RC before the final 1.2.0 release.

The version number must be updated in several files:

- `configure.ac` file
- `agent/mmc/agent.py` file (for mmc-core only)
- `agent/mmc/plugin/<PLUGIN_NAME>/__init__.py` files (VERSION attribute)
- `web/modules/<MODULE_NAME>/infoPackage.inc.php` files

2. Prepare the changelog

The Changelog file must be updated. If an entry in the changelog is a bugfix of a bug reported in the bug tracking system, the ticket number must be written.

3. Documentation update

All the installation/configuration manuals must be updated and checked.

The upgrade procedure is updated:

- http://projects.mandriva.org/projects/mmc/wiki/Pulse2_Upgrade_Procedure
- http://projects.mandriva.org/projects/mmc/wiki/MDS_Upgrade_Procedure

4. Making the tarball

```
# Clean all generated/untracked files
$ git clean -fdx
# Do a fresh configure
$ ./configure --disable-python-check ...
# Make the tarball
$ make dist
```

5. Packaging and tests

Packages are published on a testing repository. The installation/upgrade is validated by the QA team and developers.

- All python unit tests of the project runs succesfully
- Selenenium tests runs succesfully
- Manual tests are succesfull

> **Warning:** If bugs are found a new RC release must be issued and tests re-run (in our example it would be 1.1.91 for the next RC)
> Fix the bugs then go back to step 1.

6. Publishing the release

Final Bump

- If all tests are successfull the version is bump to the final release number (1.2.0 in our example).
- A git tag is created after the version bump commit (MMC-CORE-XXX, MDS-XXX, Pulse2-XXX)
- The final tarball is generated.

Redmine updates

- The final tarballs are put in the public download place: http://projects.mandriva.org/projects/mmc/files
- Close the version we are going to release with the date of the release.
- Open a new version for the next release.
- If the release provide new plugins new Redmine components must be created
- Make a news for the new release (details of the news can be taken from the Changelog file)

Packages updates

- The Debian packages repository is updated, for Lenny and Squeeze.
- The RPMs packages repository for Mandriva MES5 and Mandriva Cooker are updated.
- A bug is open on https://qa.mandriva.com for MES5 official updates (eg: https://qa.mandriva.com/show_bug.cgi?id=65463)

Communication

- A mail is sent to the XXX-announce mailing list
- The freshmeat entry is updated
- A blog entry can be post on http://blog.mandriva.com
- A news can be proposed on http://www.linuxfr.org and other revelant websites.

2.2 Specifications

2.2.1 MMC audit framework specification

This document specifies the MMC audit framework.

Introduction

An administrator needs to know who did what and when. This feature is highly needed for the MMC, as it manages users and critical network ressources.

This document defines the perimeter of the MMC audit system, the data structure and the Python API that will be used to log events.

We won't talk in this document about any audit GUI design.

Perimeter

First, what is not the MMC audit framework:

- a debugging log for developers. Auditing is not logging.
- a log for all LDAP directory modifications. That's the role of the LDAP directory to provide this kind of data. Moreover, the MMC API also works on non LDAP data, like SAMBA shares for example.

We will record only "atomic" events. For example, If we have a method that remove a user from all her/his group, we will create a log record for each removed group instead of a single log telling that the user has been removed from all groups. This precision is needed to follow the exact life-cycle of the objects.

Structure of an audit record

An audit record is made of the following datas:

- timestamp: when does this event occured ?
- source: which host is performing the action related to the event and reporting the event ? This can be the host name of the MMC agent reporting the event, or the host name where a script using the MMC API is used.
- initiator: who triggered this event ? This can be the user id of the user connected to the MMC agent, or the effective user id of a script using the MMC API.
- initiator: which application on which host initiated this event ? The typical cases: the MMC web interface on computer laptop.example.net, the python script `/root/populate_ldap.py` on computer mds.example.net.
- event: what happened ? To designate an event, we will use a simple label. For example, the "add a user action" from the MMC "base" API is called "BASE_ADD_USER".
- action result: was the operation triggering the event succesfull ?
- target: which were the object affected by the event ? An object could be a user, a group, a user LDAP attribute, etc.
- what were the previous values of the affected targets, and what are the current values ? For example, if we modified the value of a LDAP attribute, we will save its previous and current value in the audit record.

Audit database schema

We will provide a MySQL and a PostgreSQL database backend, thanks to the python SQLAlchemy library.

Here is the database schema:

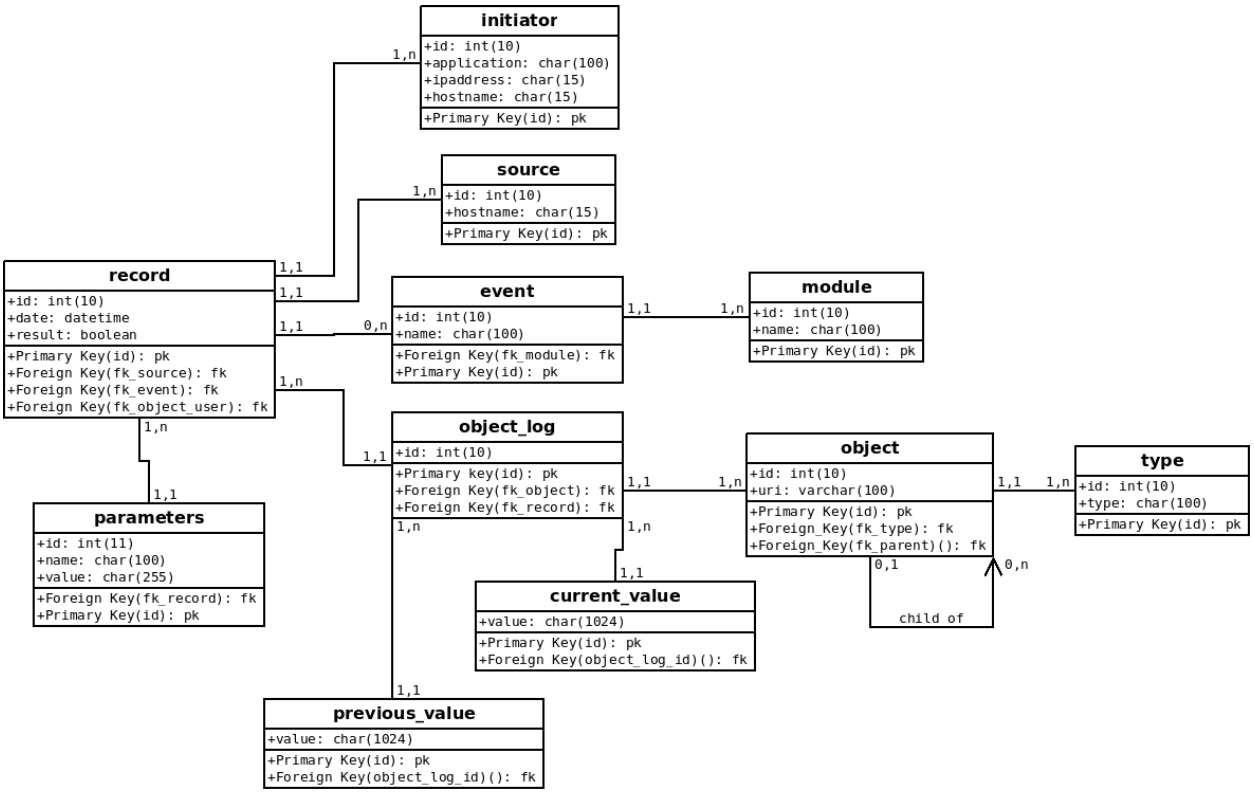


Table description:

- initiator: application (MMC web interface for example), IP address and hostname of the server that initiated the connection;
- source: hostname of the machine that received the action that triggered the event, and which is reporting the event;
- module: module name owner of the event. If the event is linked to the MMC SAMBA module, the name will be `MMC-SAMBA`;
- event: name of the reported event, for example “SAMBA_LOCK_USER”. Each event is linked to its module;
- type: name of a object type, for example “USER”, “GROUP”, ...
- object: URI of a object affected by the audit record. If the object is a LDAP object, the URI is the LDAP DN of the object. For other type of object, a URI system must be found. Each object has a type. For example, the object representing the LDAP user “foo” has the URI “uid=foo,ou=Users,dc=mds” and the type “USER”.
- object_log: link a log entry to object entries;
- previous_value: the previous value (if applicable) of the object before the audit record;
- current_value: the value (if applicable) of the object after the audit record;
- parameters: additional event parameters (optional);
- log: a log is linked to an initiator, a source, an event, one or multiple object_log rows, and zero or multiple parameters. The first object_log row is always linked to the object representing the user that triggered the event. The “result” column is a boolean, which value is false if the operation linked to the event failed.

How do we address object in a log ?

There is a little problem when addressing object in a log. For example, we want to record that the LDAP attribute called "fooattr" from the user "foo" has been deleted. How do we implement that using this database structure ?

When storing the log data into the database, we will simply connect the object representing the LDAP attribute "fooattr" to the object representing the user "foo" thanks to the fk_parent field.

How do we know the current execution context ?

All XML-RPC calls received by the MMC agent are executed in threads. Each time a new thread is started, the current user session is attached to the thread. From the session the MMC agent knows which user triggered an event and the initiator (the MMC web interface in most case).

When using the MMC API without the MMC agent (no XML-RPC calls), the initiator is the current host and the current application (sys.argv[0]), and the user is the current effective user id number.

Python API

AuditFactory singleton class

We provide a Singleton class called "AuditFactory" that allows to access the audit framework. It reads the "audit" section of `/etc/mmc/plugins/base.ini` file that defines the database connection.

For compatibility, the audit framework can be disabled.

Logging an event

The AuditFactory class owns this method to log an event:

```
def log(self, module, event, objects = None, current=None, previous=None, parameters = None)
```

- module: module name owner of the event
- event: event name
- objects: objects affected by the event. the object is represented by a couple (object name, object type). For example, the user "foo" is `("foo", "USER")`. If the object is a child of another object, its parent must be prepended in a list. For example, the attribute "fooattr" of the user "foo" is `[("foo", "USER"), ("fooattr", "ATTRIBUTE")]`
- previous: previous value of the object affected by the event;
- current: current value of the object affected by the event;
- parameters: parameters used when performing the action that triggered the event;

This method creates all needed rows into the audit database. It should be called just before an action is performed. It sets the log database result field to False to define that the action has not been performed or has failed.

This method returns an AuditRecord object, that has only one method called "commit", that should be use when the action is done:

Example

```
from mmc.core.audit import AuditFactory
# Record to the audit database the action being performed
r = AuditFactory().log("MODULE_TEST", "TEST_AUDIT")
# Do domething
...
# Flag the action has successfull
r.commit()
```

Declaring module events and type

Each MMC API module have a `audit.py` Python file that defines all events and types managed by the module.

Here is an extract of what contains the `audit.py` file for the "base" MMC module:

```
class AuditActions:
    BASE_ADD_USER = u'BASE_ADD_USER'
    BASE_ENABLE_USER = u'BASE_ENABLE_USER'
    ...

class AuditTypes:
    USER = u'USER'
    GROUP = u'GROUP'
    ...

AA = AuditActions
AT = AuditTypes
PLUGIN_NAME = u'MMC-BASE'
```

Remarks:

- All the strings must be unicode strings, in uppercase.
- Actions (events) name starts with the name of the plugin

Defining object URI

An object URI must allow us to identify and address a unique object in the audit database, to record and track all its changes.

For LDAP objects, it is logical to use the object DN as the URI to store into the database.

But the MMC allows to modify objects which are not into the LDAP, for example SAMBA shares. For this kind of objects, a method to build an URI must be found.

2.3 QA

2.3.1 MDS QA

This page describes how to test the MDS interface before a release.

Selenium tests

Install the Selenium IDE plugin for Firefox at http://seleniumhq.org.

Running the test suite

- Open Selenium IDE (Ctrl+Alt+S)
- Open the test suite from the repository (`mmc/mds/tests/selenium/suite/all_test.html`)
- Browse to the MMC login page on your test server. The MDS installation must be clean with samba, mail and network modules. Root password must be `secret`.
- Play the full test suite

Manual tests

Manual tests validate basic usage of MSS (Mandriva Server Setup) and MDS on MES 5.2. We basically check that the setup of SAMBA, bind9, dhcpd, postfix, dovecot, OpenLDAP with MDS done by MSS is OK.

Environment setup

A private network with one MBS 1.0 server and one Windows XP or 7 client.

The network and the machines can of course be virtualized (it's easy to setup with VirtualBox).

- The MBS server is a base installation from the DVD + all updates
- The private network will be `192.168.220.0/24` in this document
- The MBS server has a static IP of `192.168.220.10`

Enable the testing repo to be able to get the MDS test packages.

Installation & configuration

From mss (https://192.168.220.10:8000) select and install the following modules: Samba, DNS & DHCP, Mail, Webmail.

- MDS domain: `test.local`
- MDS password: `test$!`
- SAMBA password: `smbTest!`
- Mail networks: `192.168.220.0/255.255.255.0`
- DNS networks: `192.168.220.0/255.255.255.0`

The configuration must be successfull.

MDS tests

1. Login in MDS at http://192.168.220.10/mmc/ with `root/test$!`
2. Add a user:
    - Login: `user1`

- Password: `test1`
- Mail: `user1@example.com`

3. Edit the user and set some other fields:
   - Last name, phone...
4. Put the user in a secondary group then remove if from the group.
5. Add a second user:
   - Login: `user2`
   - Password: `test2`
   - Mail: `user2@example.com`
   - Alias: `contact@example.com`
6. Add the mail domain `example.com`
7. Login in roundcube (http://192.168.220.10/roundcubemail) with `user1@example.com`.
   - send a mail to `user2@example.com`
   - send a mail to `contact@example.com`
8. Login in roundcube with `user2@example.com` and check the mails
9. Edit the MMC ACLs of user1 and check the "Change password" page
   - Login the MMC with `user1` and change his password
10. Create a DNS zone
    - FQDN: `example.com`
    - Server IP: `192.168.220.10`
    - Network address: `192.168.220.0`
    - Network mask: `24`
    - Create DHCP subnet and reverse zone
11. Edit the DHCP subnet and add a dynamic pool from `192.168.220.50` to `192.168.220.60`
12. Restart both services in `Network Services Management`
13. Boot the Windows client and check if it gets an IP
14. Convert the dynamic lease of the client to a static lease. Set the DNS name to `win.example.com`.
15. Renew the lease on the windows client (`ipconfig /renew`) then ping `win.example.com`.
16. Join the computer to the `MES5DOMAIN` domain (`admin/smbTest!`)
17. Login with `user1` on the Windows client
18. Change `user1` password on the Windows client
19. Login the MMC with `user1` new password
20. Change `user1` password on the MMC interface
21. Logout/Login with `user1` on the Windows client