Modoboa Documentation

Release 0.9.5

Antoine Nguyen

May 24, 2013

CONTENTS

This documentation refers to version 0.9.5.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 Requirements

- Python version 2.6+
- Django version 1.3+
- south version 0.7+
- lxml python module
- pycrypto python module
- rrdtool python binding
- sievelib python module
- chardet python module
- argparse python module

1.1.2 Get Modoboa

You can choose between two options:

- Use the Python package available on the PyPI
- Download the sources tarball

The easiest one is to install it from the *PyPI*. Just run the following command and you're done:

```
$ pip install modoboa
```

If you prefer to use the tarball, download the latest one and run the following procedure:

```
$ tar xzf modoboa-<version>.tar.gz
$ cd modoboa-<version>
$ python setup.py install
```

All dependencies will be installed regardless the way you chose. The only exception concerns the RRDtool binding because there isn't any python package available, it is directly provided with the official tarball.

Fortunately, all major distributions include a ready-to-use package. On Debian/Ubuntu:

```
$ apt-get install python-rrdtool
```

1.1.3 Database

Thanks to *django*, Modoboa supports several databases. Depending on the one you will use, you must install the appropriate python package:

- mysqldb for MySQL
- psycopg2 for PostgreSQL

Then, create a user and a database that will be used by Modoboa. Make sure your database is using UTF8 as a default charset.

1.1.4 Deployment

Automatic

modoboa-admin.py, a command line tool, let you deploy a *ready-to-use* Modoboa site using only one instruction:

```
$ modoboa-admin.py deploy modoboa_example --syncdb --collectstatic [--with-amavis]
```

Just answer the few questions and you're done. You can now go to the *First use* section.

Note: The *–with-amavis* option must be set only if you intend to use the *Amavisd-new frontend*.

Manual

As Modoboa is a set of Django applications, you need to create a new project. Just run the following commands:

```
$ cd /var/www
$ django-admin.py startproject modoboa_example
$ cd modoboa_example
$ rm settings.py
$ rm urls.py
$ wget http://modoboa.org/resources/settings.py
$ wget http://modoboa.org/resources/urls.py
$ mkdir media
```

Then, edit the freshly downloaded *settings.py* file and adjust the database relative information. (see *Database*).

Note: If you plan to serve Modoboa using a URL prefix, you must change the value of the `LOGIN_URL` parameter to `LOGIN_URL = '/<prefix>/accounts/login/'`.

Finally, run the following commands:

```
$ python manage.py collectstatic
$ python manage.py syncdb --migrate --noinput
$ python manage.py loaddata initial_users.json
```

1.1.5 First use

Your installation should now have a default super administrator:

- Username: `admin`
- Password: `password`

It is **strongly** recommended to change this password the first time you log into Modoboa.

To check if your installation works, just launch the embedded HTTP server:

```
$ python manage.py runserver
```

You should be able to access Modoboa at http://locahost:8000/.

For a production environnement, we recommend using a stable webserver like *Apache2* or *Nginx*.

1.2 Upgrading an existing installation

This section contains all the upgrade procedures required to use newest versions of Modoboa.

Note: Before running a migration, we recommend that you make a copy of your existing database.

1.2.1 Latest version

Starting with version 0.9.1, Modoboa comes as a standard django application. Fetch the latest version (see *Get Modoboa*) and install it.

`pip` users, just run the following command:

```
$ pip install --upgrade modoboa
```

Then, follow the common procedure:

```
$ cd <modoboa_instance_dir>
$ python manage.py syncdb --migrate
$ python manage.py collectstatic
```

Finally, refer to this page to check if the version you're installing requires specific operations. If the version you're looking for is not present, it means nothing special is required.

0.9.4: administrative panel performance improved

1. Edit the *settings.py* file and remove `'django.contrib.auth.backends.ModelBackend'` from the `AUTHENTICATION_BACKENDS` variable

0.9.1: standard django application and more

For this version, we recommend to install a new instance (see *Deployment*) in a different directory.

Then, copy the following content from the old installation to the new one:

- The `media` directory

- The directory containing RRD files if you use the *Graphical statistics* plugin

Don't copy the old *settings.py* file, just keep the new one and modify it (see *Database* and *Time zone and language*).

Migrate your database (see *Latest version*).

Finally, check the *Amavisd-new frontend*, *Postifx auto-reply messages* and *Graphical statistics* chapters (depending on those you use) because the provided cron scripts have been changed, you must update the way you call them.

1.2.2 Modoboa 0.9 and prior

First, decompress the new tarball at the same location than your current installation. Then, check if the new version you're installing requires a migration.

0.9: global UI refactoring, new *limits* extension and more

Note: This version requires at least django 1.3. Make sure to update your version before starting to migrate.

Note: Many files have been renamed/removed for this version. I recommend that you backup important files (*settings.py*, etc.) elsewhere (ie. */tmp* for example). Then, remove the *modoboa* directory, extract the new tarball at the same place, rename the new directory to *modoboa* and copy the files you've just backup into it.

Note: If the first super administrator you created is named `admin`, its password will be changed to `password` at the end of this upgrade. Don't forget to modify it!

1. Edit the *settings.py* file and update the following variables (just copy/paste their new content):

   ```
   MIDDLEWARE_CLASSES = (
       'django.middleware.common.CommonMiddleware',
       'django.contrib.sessions.middleware.SessionMiddleware',
       'django.contrib.auth.middleware.AuthenticationMiddleware',
       'django.contrib.messages.middleware.MessageMiddleware',
       'django.middleware.locale.LocaleMiddleware',
       'modoboa.lib.middleware.AjaxLoginRedirect',
       'modoboa.lib.middleware.CommonExceptionCatcher',
       'modoboa.lib.middleware.ExtControlMiddleware',
   )

   AUTHENTICATION_BACKENDS = (
       'modoboa.lib.authbackends.SimpleBackend',
       'django.contrib.auth.backends.ModelBackend',
   )
   ```

2. Add `django.contrib.staticfiles` to `INSTALLED_APPS`

3. Add the following new variables:

   ```
   STATIC_ROOT = os.path.join(MODOBOA_DIR, 'sitestatic')
   STATIC_URL = '/sitestatic/'
   ```

4. Update the following variables (just copy/paste their new values):

   ```
   MEDIA_ROOT = os.path.join(MODOBOA_DIR, 'media')
   MEDIA_URL = '/media/'
   ```

5. **For MySQL users only**, add the following option to your database configuration:

   ```
   DATABASES = {
       "default" : {
           # ...
           # MySQL users only
           "OPTIONS" : {
               "init_command" : "SET foreign_key_checks = 0;",
           },
       }
   }
   ```

6. Add `'modoboa.extensions.limits'` to `INSTALLED_APPS`
7. Update your database (make sure to create a backup before launching the following command):

   ```
   $ ./manage.py syncdb --migrate
   ```

8. Run the following command to initialize the directory that contains static files:

   ```
   $ ./manage.py collectstatic
   ```

9. If you are using the *stats* extension, please rename the *<modoboa_dir>/static/stats* directory to *<modoboa_dir>/media/stats* and change the value of the `IMG_ROOTDIR` parameter (go to the adminstration panel)
10. Restart the python instance(s) that serve *Modoboa*
11. Log into Modoboa, go to *Modoboa* > *Extensions*, uncheck all extensions, save. Then, check the extensions you want to use and save again
12. Update your webserver configuration to make static files available (see *Web servers*)
13. **For Dovecot users only**, you need to modify the `password_query` (file */etc/dovecot/dovecot-sql.conf* by default on a Debian system) like this:

   ```
   password_query = SELECT email AS user, password FROM auth_user WHERE email='%u'
   ```

0.8.8: CSV import feature and minor fixes

1. Edit the *settings.py* file and add `'modoboa.lib.middleware.AjaxLoginRedirect'` to the `MIDDLEWARE_CLASSES` variable like this:

   ```
   MIDDLEWARE_CLASSES = (
     'django.middleware.common.CommonMiddleware',
     'django.contrib.sessions.middleware.SessionMiddleware',
     'django.contrib.auth.middleware.AuthenticationMiddleware',
     'django.contrib.messages.middleware.MessageMiddleware',
     'django.middleware.locale.LocaleMiddleware',
     'modoboa.lib.middleware.AjaxLoginRedirect',
     'modoboa.lib.middleware.ExtControlMiddleware',
     'modoboa.extensions.webmail.middleware.WebmailErrorMiddleware',
   )
   ```

2. Still inside *settings.py*, modify the `DATABASE_ROUTERS` variable like this:

   ```
   DATABASE_ROUTERS = ["modoboa.extensions.amavis_quarantine.dbrouter.AmavisRouter"]
   ```

0.8.7: per-user language selection

1. Edit the *settings.py* file and add the `'django.middleware.locale.LocaleMiddleware'` middleware to the `MIDDLEWARE_CLASSES` variable like this:

   ```
   MIDDLEWARE_CLASSES = (
     'django.middleware.common.CommonMiddleware',
     'django.contrib.sessions.middleware.SessionMiddleware',
     'django.contrib.auth.middleware.AuthenticationMiddleware',
     'django.contrib.messages.middleware.MessageMiddleware',
     'django.middleware.locale.LocaleMiddleware',
     'modoboa.lib.middleware.ExtControlMiddleware',
     'modoboa.extensions.webmail.middleware.WebmailErrorMiddleware',
   )
   ```

2. To select a custom language, go to *Options* > *Preferences* and select the `general` section. Choose a value, save and disconnect from Modoboa. On the next login, the desired language will be used.

0.8.6.1: maintenance release

1. If you have tried to create a new mailbox and if you have encountered the following issue, you must run the `dbcleanup.py` script in order to remove orphan records:

   ```
   $ cd <modoboa_dir>
   $ PYTHONPATH=$PWD/.. DJANGO_SETTINGS_MODULE=modoboa.settings ./admin/scripts/dbcleanup.py
   ```

0.8.6: Quarantine plugin refactoring (using Django's ORM)

1. Just update your configuration if you are using the quarantine plugin. Open *settings.py*, move the database configuration from the `DB_CONNECTIONS` variable to the `DATABASES` variable, like this:

   ```
   DATABASES = {
       "default" : {
           # The default database configuration
       },
       #    ...
       "amavis": {
           "ENGINE" : "<your value>",
           "HOST" : "<your value>",
           "NAME" : "<your value>",
           "USER" : "<your value>",
           "PASSWORD" : "<your value>"
       }
   }
   ```

2. Add the new following variable somewhere in the file:

   ```
   DATABASE_ROUTERS = ["modoboa.extensions.amavis_quarantine.dbrouter.AmavisRouter"]
   ```

3. Remove the deprecated `DB_CONNECTIONS` variable from *settings.py*.

0.8.5: new "Sieve filters" plugin, improved admin app

1. Migrate the `lib` and `admin` applications:

```
$ python manage.py migrate lib
$ python manage.py migrate admin
```

2. Add `modoboa.auth` and `modoboa.extensions.sievefilters` to the `INSTALLED_APPS` variable in *settings.py*.
3. Go the *Settings/Extensions* panel, deactivate and activate your extensions, it will update all the symbolic links.

0.8.4: folders manipulation support (webmail) and bugfixes

1. Update the `MIDDLEWARE_CLASSES` variable in *settings.py*:

```
MIDDLEWARE_CLASSES = (
  'django.middleware.common.CommonMiddleware',
  'django.contrib.sessions.middleware.SessionMiddleware',
  'django.contrib.auth.middleware.AuthenticationMiddleware',
  'django.contrib.messages.middleware.MessageMiddleware',
  'modoboa.lib.middleware.ExtControlMiddleware',
  'modoboa.extensions.webmail.middleware.WebmailErrorMiddleware',
)
```

2. Go the *Settings/Extensions* panel, deactivate and activate your extensions, it will update all the symbolic links to the new format.
3. Optional: update the `DATABASES` and `TEMPLATE_LOADERS` variables in *settings.py* to remove warning messages (appearing with Django 1.3):

```
DATABASES = {
  "default" : {
    "ENGINE" : "<your engine>",
    "NAME" : "modoboa",
    "USER" : "<your user>",
    "PASSWORD" : "<your password>",
    "HOST" : "",
    "PORT" : ""
  }
}

TEMPLATE_LOADERS = (
  'django.template.loaders.filesystem.Loader',
  'django.template.loaders.app_directories.Loader',
)
```

0.8.3: admin application refactoring and more

1. Migrate the *admin* application:

```
$ python manage.py migrate admin
```

2. Update SQL queries used in your environnement (see *Postfix* or *Dovecot*).
3. Update *postfix* configuration so that it can handle domain aliases (see *Postfix*).

0.8.2: ckeditor integration and more

1. Migrate the admin applicaton:

```
$ python manage.py migrate admin
```

2. Update your config file and add all extensions to `INSTALLED_APPS` (even those you are not going to use).
3. Inside the *<modoboa_dir>/templates/* directory, remove all symbolic links.
4. Download the latest release of ckeditor and extract it into *<modoboa_dir>/static/js/*. It should create a new directory named *ckeditor*.
5. Update the following variables inside *settings.py*:

   ```
   MEDIA_ROOT = os.path.join(MODOBOA_DIR, 'static')
   MEDIA_URL = '/static/'
   ```

6. Then, add the following variable: `MODOBOA_WEBPATH = 'modoboa/'`
7. Delete the following variables: `STATIC_ROOTDIR` and `TEMPLATE_CONTEXT_PROCESSORS`.
8. Finally, add `modoboa.lib.middleware.ExtControlMiddleware` to `MIDDLEWARE_CLASSES`.

0.8.1 : project renamed

1. First, rename the *mailng* directory to *modoboa* and copy all the content from *modoboa-0.8.1* to *modoboa*.
2. Edit *settings.py* and replace all occurences of mailng by modoboa. Make sure you don't modify the `DATABASE` section as you're not going to rename your database.
3. Rename the `MAILNG_DIR` variable to `MODOBOA_DIR`.
4. Add `'django.contrib.messages.middleware.MessageMiddleware'` to `MIDDLEWARE_CLASSES` and `'django.contrib.messages'` to `INSTALLED_APPS`. Save your modifications.
5. Run the following command:

   ```
   $ python manage.py syncdb
   ```

6. For all activated extensions, run the following command:

   ```
   $ export PYTHONPATH=<modoboa_dir>/..=
   $ DJANGO_SETTINGS_MODULE=modoboa.settings <modoboa_dir>/scripts/extension.py <extension> on
   ```

7. Update your webserver configuration and restart it.

0.8 : SQL migration needed

Before you start the migration, make sure you have updated your `INSTALLED_APPS` variable and that it contains at least:

```
INSTALLED_APPS = (
   # Django's stuff before

   'south',
   'mailng',
   'mailng.lib',
   'mailng.admin',
   'mailng.userprefs',
)
```

Starting with 0.8, mailng.main doesn't exist anymore. You must remove it from your `INSTALLED_APPS`.

Finally, run the following commands:

```
$ python manage.py syncdb
$ python manage.py convert_to_south
$ python manage.py migrate --all 0001 --fake
$ python manage.py migrate --all 0002
```

1.3 Configuration

1.3.1 Online parameters

Modoboa provides online panels to modify internal parameters. There are two available levels:

- Application level: global parameters, define how the application behaves. Available at *Modoboa* > *Parameters*
- User level: per user customization. Available at *User* > *Settings* > *Preferences*

Regardless level, parameters are displayed using tabs, each tab corresponding to one application.

admin application parameters

The *admin* application exposes several parameters, mainly to control how mailboxes are created and how messages are stored on the file system.

Default values are presented below:

```
The authentication type
AUTHENTICATION_TYPE = Local

Does Modoboa need to handle the creation of directories on the file system?
CREATE_DIRECTORIES = Yes

Where Modoboa stores mailboxes
STORAGE_PATH = /var/vmail

Which user:group Modoboa uses to assign permissions on mailboxes
VIRTUAL_UID = vmail
VIRTUAL_GID = vmail

Which format is used to store messages in a mailbox
MAILBOX_TYPE = maildir

If maildir is in use, tells Modoboa that mailbox content is under a
potential sub-directory
MAILDIR_ROOT = .maildir

When a mailbox is removed, also remove the associated account
AUTO_ACCOUNT_REMOVAL = No

The encryption method used to store passwords
PASSWORD_SCHEME = crypt

The number of displayed items per page for listing pages
ITEMS_PER_PAGE = 30
```

1.3.2 Host configuration

Note: This section is only relevant when the `CREATE_DIRECTORIES` parameter is set to `Yes`.

To let Modoboa create mailboxes and store emails on the filesystem, you must create a group and a user (UNIX ones). There is only one group/user needed because we are in a virtual hosting configuration (ie. users with non-UNIX accounts).

The following examples are based on the default values presented in *admin application parameters*.

For example, create a vmail group:

```
$ groupadd vmail
```

Then create a vmail user:

```
$ useradd -g vmail -d /var/vmail -m vmail
```

At last, the system user used to run modoboa will need permissions to manipulate directories in vmail's homedir. To do so, edit the */etc/sudoers* file and add the following inside:

```
<user_that_runs_modoboa> ALL=(vmail) NOPASSWD: ALL
```

1.3.3 Time zone and language

Modoboa is available in english, french, german and spanish.

To specify the default language to use, edit the *settings.py* file and modify the `LANGUAGE_CODE` variable:

```
LANGUAGE_CODE = 'en-US' # English
# or
LANGUAGE_CODE = 'fr-FR' # French
# or
LANGUAGE_CODE = 'de-DE' # German
# or
LANGUAGE_CODE = 'es-ES' # Spanish
```

Note: Each user has the possibility to define the language he prefers.

In the same configuration file, specify the timezone to use by modifying the `TIME_ZONE` variable. For example:

```
TIME_ZONE = 'Europe/Paris'
```

1.3.4 Sessions management

Modoboa uses Django's session framework to store per-user information.

Few parameters need to be set in the *settings.py* configuration file to make Modoboa behave as expected:

```
SESSION_EXPIRE_AT_BROWSER_CLOSE = False # Default value
SESSION_COOKIE_AGE = 600 # time in seconds
```

The first parameter is optional but you must ensure it is set to `False` (the default value).

The second one tells Django that a session inactive for this time should be considered as closed. You are free to adjust it according to your need.

The default configuration file provided by the *modoboa-admin.py* command is properly configured.

Clearing the session table

Django does not provide automatic purging. Therefore, it's your job to purge expired sessions on a regular basis.

Django provides a sample clean-up script: `django-admin.py cleanup`. That script deletes any session in the session table whose `expire_date` is in the past.

For example, you could setup a cron job to run this script every night:

```
0 0 * * * <modoboa_site>/manage.py cleanup
```

1.3.5 External authentication

LDAP

Modoboa supports external LDAP authentication using the following extra components:

- Python LDAP client
- Django LDAP authentication backend

If you want to use this feature, you must first install those components:

```
$ pip install python-ldap django-auth-ldap
```

Then, all you have to do is to modify the *settings.py* file:

- Add a new authentication backend to the *AUTHENTICATION_BACKENDS* variable, like this:

  ```
  AUTHENTICATION_BACKENDS = (
    'modoboa.lib.authbackends.LDAPBackend',
    'modoboa.lib.authbackends.SimpleBackend',
  )
  ```

- Set the required parameters to establish the communication with your LDAP server, for example:

  ```
  import ldap
  from django_auth_ldap.config import LDAPSearch

  AUTH_LDAP_SERVER_URI = "ldap://<ldap server address>"
  AUTH_LDAP_BIND_DN = ""
  AUTH_LDAP_BIND_PASSWORD = ""
  LDAP_USER_BASE = "ou=users,dc=example,dc=com"
  LDAP_USER_FILTER = "(mail=%(user)s)"
  AUTH_LDAP_USER_SEARCH = LDAPSearch(LDAP_USER_BASE,
      ldap.SCOPE_SUBTREE, LDAP_USER_FILTER)

  AUTH_LDAP_USER_ATTR_MAP = {
      "first_name": "givenName",
      "email": "mail",
      "last_name": "sn"
  }
  ```

You will find a detailled documentation here.

Finally, go to *Modoboa* > *Parameters* > *admin*, set the `AUTHENTICATION_TYPE` parameter to `LDAP` and click on the *Save* blue button.

Once the authentication is properly configured, the users defined in your LDAP directory will be able to connect to *Modoboa*, the associated domain and mailboxes will be automatically created if needed.

Users will also be able to update their LDAP password directly from Modoboa.

Note: Modoboa doesn't provide any synchronization mechanism once a user is registered into the database. Any modification done from the directory to a user account will not be reported to Modoboa (an email address change for example). Currently, the only solution is to manually delete the Modoboa record, it will be recreated on the next user login.

Available settings

- `LDAP_USER_BASE` : the distinguish name of the search base
- `LDAP_USER_FILTER` : the filter used to retrieve users distinguish name
- `LDAP_PASSWORD_ATTR` : the attribute used to store the password (default: `userPassword`)
- `LDAP_ACTIVE_DIRECTORY` : used to indicate if your directory is an Active Directory one (default: `False`)

1.4 Using plugins

1.4.1 Enable or disable a plugin

Modoboa provides an online panel to control plugins activation. You will find it at *Modoboa* > *Extensions*.

To activate a plugin, check the corresponding box and click on the *Apply* button.

To deactivate a plugin, uncheck the corresponding box and click on the *Apply* button.

1.4.2 Per-admin limits

This plugin offers a way to define limits about how many objects (aliases, mailboxes) a domain administrator can create.

It also brings a new administrative role: `Reseller`. A reseller is a domain administrator that can also manipulates domains and assign permissions to domain administrators.

If you don't want to limit a particular object type, just set the associated value to -1.

1.4.3 Amavisd-new frontend

This plugin provides a simple management frontend for amavisd-new. The supported features are:

- SQL quarantine management : available to administrators or users, possibility to delete or release messages
- Per domain customization (using policies): specify how amavisd-new will handle traffic

---

Note: The per-domain policies feature only works for new installations. Currently, you can't use modoboa with an existing database (ie. with data in *users* and *policies* tables).

Note: This plugin requires *amavisd-new* version **2.7.0** or higher. If you're planning to use the *Self-service mode*, you'll need version **2.8.0**.

Database

You must tell to Modoboa where it can find the amavis database. Inside *settings.py*, add a new connection to the `DATABASES` variable like this:

```
DATABASES = {
  # Stuff before
  #
  "amavis": {
    "ENGINE" : "<your value>",
    "HOST" : "<your value>",
    "NAME" : "<your value>",
    "USER" : "<your value>",
    "PASSWORD" : "<your value>"
  }
}
```

Replace values between <> with yours.

Note: Modoboa doesn't create *amavis* tables. You need to install them following the official documentation.

Cleanup

Storing quarantined messages to a database can quickly become a perfomance killer. Modoboa provides a simple script to periodically purge the quarantine database. To use it, add the following line inside root's crontab:

```
0 0 * * * <modoboa_site>/manage.py qcleanup
```

Replace `modoboa_site` with the path of your Modoboa instance.

By default, messages older than 14 days are automatically purged. You can modify this value by changing the `MAX_MESSAGES_AGE` parameter in the online panel.

Release messages

To release messages, first take a look at this page. It explains how to configure *amavisd-new* to listen somewhere for the AM.PDP protocol. This protocol is used to send requests.

Below is an example of a working configuration:

```
$interface_policy{'SOCK'} = 'AM.PDP-SOCK';
$interface_policy{'9998'} = 'AM.PDP-INET';

$policy_bank{'AM.PDP-SOCK'} = {
  protocol => 'AM.PDP',
  auth_required_release => 0,
};
$policy_bank{'AM.PDP-INET'} = {
  protocol => 'AM.PDP',
  inet_acl => [qw( 127.0.0.1 [::1] )],
};
```

Don't forget to update the `inet_acl` list if you plan to release from the network.

Once *amavisd-new* is configured, just tell Modoboa where it can find the *release server* by modifying the following parameters in the online panel:

```
# "unix" or "inet"
AM_PDP_MODE = "inet"

# "unix" mode only
AM_PDP_SOCKET = "/var/amavis/amavisd.sock"

# "inet" mode only
AM_PDP_HOST = "127.0.0.1"
AM_PDP_PORT = 9998
```

Deferred release

By default, simple users are not allowed to release messages themselves. They are only allowed to send release requests to administrators.

As administrators are not always available or logged into Modoboa, a notification tool is available. It sends reminder e-mails to every administrators or domain administrators. To use it, add the following example line to root's crontab:

```
0 12 * * * <modoboa_site>/manage.py amnotify --baseurl='<modoboa_url>'
```

You are free to change the frequency.

Note: If you want to let users release their messages alone (not recommended), change the value of the `USER_CAN_RELEASE` parameter into the admin panel.

Self-service mode

The *self-service* mode let users act on quarantined messages without beeing authenticated. They can:

- View messages
- Remove messages
- Release messages (or send release requests)

To access a specific message, they only need the following information:

- Message's unique identifier
- Message's secret identifier

This information is controlled by *amavis*, which is in charge of notifying users when new messages are put into quarantine. Each notification (one per message) must embark a direct link containing the required identifiers.

To activate this feature, go the administration panel and set the `SELF_SERVICE` paramater to yes.

The last step is to customize the notification messages *amavis* sends. The most important is to embark a direct link. Take a look at the README.customize file to learn what you're allowed to do.

Here is a link example:

```
http://<modoboa_url>/quarantine/%i/?rcpt=%R&secret_id=[:secret_id]
```

1.4.4 Graphical statistics

This plugin collects various statistics about emails traffic on your server. It parses a log file to collect information, store it into RRD files (see rrdtool)and then generates graphics in PNG format.

To use it, go to the online parameters panel and adapt the following ones to your environnement:

```
# Path to mail log file
LOGFILE = "/var/log/mail.log"

# Path to directory where rrd files are stored
RRD_ROOTDIR = "/tmp/modoboa"

# Path to directory where png files are stored
IMG_ROOTDIR = "<modoboa_site>/media/stats"
```

Make sure the directory that will contain RRD files exists (`RRD_ROOTDIR`). If not, create it before going further. For example (according to the previous parameters):

```
$ mkdir /tmp/modoboa
```

To finish, you need to collect information periodically in order to feed the RRD files. Add the following line into root's crontab:

```
*/5 * * * * <modoboa_site>/manage.py logparser &> /dev/null
```

Replace `<modoboa_site>` with the path of your Modoboa instance.

Graphics will be automatically created after each parsing.

1.4.5 Postifx auto-reply messages

This plugin let users define an auto-reply message (*vacation*). It is based on *postfix* capabilities.

The user that executes the autoreply script needs to access *settings.py*. You must apply proper permissions on this file. For example, if *settings.py* belongs to *www-data:www-data*, you can add the *vmail* user to the *www-data* group and set the read permission for the group.

To make *postfix* use this feature, you need to update your configuration files as follow:

`/etc/postfix/main.cf`:

```
transport_maps = mysql:/etc/postfix/maps/sql-transport.cf
virtual_alias_maps = mysql:/etc/postfix/maps/sql-aliases.cf
        mysql:/etc/postfix/maps/sql-autoreplies.cf
```

`/etc/postfix/master.cf`:

```
autoreply unix        -       n       n       -       -       pipe
          flags= user=vmail:<group> argv=<modoboa_site>/manage.py autoreply $sender $mailbox
```

`<modoboa_site>` is the path of your Modoboa instance.

Then, create the requested map files:

```
$ modoboa-admin.py postfix_maps mapfiles --categories autoreply
```

mapfiles is the directory where the files will be stored. Answer the few questions and you're done.

Note: Auto-reply messages are just sent one time per sender for a pre-defined time period. By default, this period is equal to 1 day (86400s), you can adjust this value by modifying the `AUTOREPLY_TIMEOUT` parameter available in the online panel.

1.4.6 Sieve filters

This plugin let users create server-side message filters, using the sievelib module (which provides Sieve and ManageSieve clients).

Two working modes are available:

- A raw mode: you create filters using the Sieve language directly (advanced users)
- An assisted mode: you create filters using an intuitive form

To use this plugin, your hosting setup must include a *ManageSieve* server and your local delivery agent must understand the *Sieve* language. Don't panic, *Dovecot* supports both :-) (refer to *Dovecot* to know how to enable those features).

Note: The sieve filters plugin requires that the *Webmail* plugin is activated and configured.

Go the online panel and modify the following parameters in order to communicate with the *ManageSieve* server (default values are displayed below):

```
SERVER = localhost
PORT = 2000
STARTTLS = no
AUTHENTICATION_MECH = plain
```

1.4.7 Webmail

Modoboa provides a simple webmail:

- Browse, read and compose messages, attachments are supported
- HTML messages are supported
- CKeditor integration
- Manipulate mailboxes (create, move, remove)
- Quota display

To use it, go to the online panel and modify the following parameters in order to communicate with your *IMAP* and *SMTP* servers (default values are displayed below):

```
IMAP_SECURED = no
IMAP_SERVER = 127.0.0.1
IMAP_PORT = 143

SMTP_SECURED_MODE = None
SMTP_AUTHENTICATION = no
SMTP_SERVER = 127.0.0.1
SMTP_PORT = 25
```

The size of each attachment sent with messages is limited. You can change the default value by modifying the `MAX_ATTACHMENT_SIZE` parameter.

Using CKeditor

Modoboa supports CKeditor to compose HTML messages. To use it, first download it from the official website, then extract the tarball:

```
$ cd <modoboa_site_dir>
$ tar xzf /path/to/ckeditor/tarball.tag.gz -C sitestatic/js/
```

And you're done!

Now, each user has the possibility to choose between CKeditor and the raw text editor to compose their messages. (see *User* > *Settings* > *Preferences* > *Webmail*)

CHAPTER

TWO

INTEGRATION WITH OTHER SOFTWARES

2.1 SMTP servers

2.1.1 Postfix

This section gives an example about building a simple virtual hosting configuration with *Postfix*. Refer to the official documentation for more explanation.

Map files

You first need to create configuration files (or map files) that will be used by *postfix* to lookup into Modoboa tables.

Automatic generation

To automaticaly generate the requested map files and store them in a directory, run the following command:

```
$ modoboa-admin.py postfix_maps --dbtype <mysql|postgres> mapfiles
```

mapfiles is the directory where the files will be stored. Answer the few questions and you're done.

Manual generation

Note: The SQL queries presented below are working only with MySQL only. To make them work with PostgreSQL, just replace (when needed) the *concat* function call by the standard concatenation operator (ie. ||).

Define the following map files on your server (it should work with *postfix* versions >= 2.2):

`/etc/postfix/sql-domains.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = SELECT name FROM admin_domain WHERE name='%s' AND enabled=1
```

`/etc/postfix/sql-domain-aliases.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = SELECT dom.name FROM admin_domain dom INNER JOIN admin_domainalias domal ON dom.id=domal.targ
```

`/etc/postfix/sql-mailboxes.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = SELECT concat(dom.name, '/', mb.path, (SELECT value FROM lib_parameter WHERE name='admin.MAII
```

`/etc/postfix/sql-aliases.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = (SELECT concat(mb.address, '@', dom.name) FROM admin_mailbox mb INNER JOIN admin_domain dom (
```

`/etc/postfix/sql-domain-aliases-mailboxes.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = SELECT DISTINCT concat(mb.address, '@', dom.name) FROM admin_alias al INNER JOIN admin_domair
```

`/etc/postfix/sql-email2email.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = SELECT concat(mb.address, '@', dom.name) FROM admin_mailbox mb INNER JOIN admin_domain dom ON
```

`/etc/postfix/sql-catchall-aliases.cf`:

```
user = <user>
password = <password>
dbname = <database>
hosts = 127.0.0.1
query = (SELECT concat(mb.address, '@', dom.name) FROM admin_mailbox mb INNER JOIN admin_domain dom (
```

Configuration

Use the following configuration in the */etc/postfix/main.cf* file (this is just one possible configuration):

```
# Stuff before
mailbox_transport = virtual
maildrop_destination_recipient_limit = 1
virtual_minimum_uid = <vmail user id>
virtual_gid_maps = static:<vmail group id>
virtual_uid_maps = static:<vmail user id>
virtual_mailbox_base = /var/vmail
```

```
relay_domains =
virtual_mailbox_domains = mysql:/etc/postfix/sql-domains.cf
virtual_alias_domains = mysql:/etc/postfix/sql-domain-aliases.cf
virtual_mailbox_maps = mysql:/etc/postfix/sql-mailboxes.cf
virtual_alias_maps = mysql:/etc/postfix/sql-aliases.cf,
      mysql:/etc/postfix/sql-domain-aliases-mailboxes.cf,
      mysql:/etc/postfix/sql-email2email.cf,
      mysql:/etc/postfix/sql-catchall-aliases.cf

# Stuff after
```

Note: Modoboa supports both maildir and mbox formats. You can specify which format to use by modifying the MAILBOX_TYPE parameter available in the admin panel.

Recommended: using Dovecot's LDA

If you are using *Dovecot* in your environnement, we recommend to use its LDA. Doing so, you'll will be able to use extra functionalities such as sieve filters and more.

First, edit the */etc/postfix/main.cf* file and define (or modify if they already exist) the following parameters:

```
virtual_transport = dovecot
dovecot_destination_recipient_limit = 1
```

Then, edit the */etc/postfix/master.cf* file and add the following definition at the end:

```
dovecot   unix  -       n       n       -       -       pipe
  flags=DRhu user=vmail:vmail argv=/usr/lib/dovecot/deliver -f ${sender} -d ${recipient}
```

If you have followed the *Manual generation* section to install your environnement, you need to modify the SQL query corresponding to the `virtual_mailbox_maps` parameter. Edit the */etc/postfix/maps/sql-mailboxes.cf* and modify the `query` parameter as follow:

```
query = SELECT concat(dom.name, '/', mb.path) FROM admin_mailbox mb INNER JOIN admin_domain dom ON m
```

Restart *Postfix*.

2.2 IMAP servers

2.2.1 Dovecot

If you are using the *maildir* format to store mailboxes, add the following line into Dovecot's main config file (*/etc/dovecot/dovecot.conf*):

```
mail_location = maildir:<path_to_mailboxes>/%h/.maildir
```

The `.maildir` part is the previous example must match the value of the `MAILDIR_ROOT` parameter. See *admin application parameters*.

If you are using the mbox format, add the following:

```
mail_location = mbox:<path_to_mailboxes>/%h/:INBOX=<path_to_mailboxes>/%h/Inbox
```

To make the authentication work, edit *dovecot.conf* and add the following content inside:

```
auth default {
  # ... stuff before

  passdb sql {
    # Path for SQL configuration file, see /etc/dovecot/dovecot-sql.conf for
    #  example
    args = /etc/dovecot/dovecot-sql.conf
  }

  userdb sql {
    # Path for SQL configuration file
    args = /etc/dovecot/dovecot-sql.conf
  }

  # ... stuff after
}
```

Make sure to activate only one backend (per type) inside your configuration (just comment the other ones).

For *MySQL* users, edit your */etc/dovecot/dovecot-sql.conf* and modify following lines:

```
driver = mysql
connect = host=<mysqld socket> dbname=<database> user=<user> password=<password>
default_pass_scheme = CRYPT
password_query = SELECT email AS user, password FROM auth_user WHERE email='%u' and is_active=1
user_query = SELECT concat(dom.name, '/', mb.path) AS home, uid, gid FROM admin_mailbox mb INNER JOIN
```

Enable quota support

Put the following lines inside the *dovecot.conf* file:

```
protocol imap {
  mail_plugins = quota imap_quota
}
```

Before continuing, you need to know which quota backend must be used (function of mailboxes format):

- mbox : backend=dirsize,
- maildir : backend=maildir.

If you use version prior to 1.1, add also the following configuration:

```
plugin {
  # 10 MB default quota limit
  quota = <backend>:storage=10240
}
```

For *MySQL* users, modify the above query inside *dovecot-sql.conf* as follow to activate per-user quotas:

```
user_query = SELECT concat(dom.name, '/', mb.path) AS home, uid, gid, concat('<backend>:storage=', mb
```

For version >= 1.1, put the following configuration inside the *dovecot.conf* file:

```
plugin {
  # Default 10M storage limit with an extra 5M limit when saving to Trash.
  quota = <backend>:User quota
  quota_rule = *:storage=10M
  quota_rule2 = Trash:storage=5M
}
```

For *MySQL* users, modify the above query inside *dovecot-sql.conf* to activate per-user quotas:

```
user_query = SELECT concat(dom.name, '/', mb.path) AS home, uid, gid, concat('*:storage=', mb.quota,
```

Enable ManageSieve/Sieve support

Note: The following configuration example should work with versions 1.X. For versions >= 2, please refer to Dovecot's wiki.

Edit the */etc/dovecot/dovecot.conf* file and make the following modifications:

- Add `managesieve` to the `protocols` variable:

  ```
  protocols = imap imaps managesieve
  ```

- Uncomment the `managesieve` section:

  ```
  protocol managesieve {
    # ...
  }
  ```

- Configure the `lda` protocol as follow:

  ```
  protocol lda {
    postmaster_address = postmaster@<your domain>
    mail_plugins = sieve # + your other plugins
    # ...
  }
  ```

- In the `plugin` section, uncomment the following content:

  ```
  plugin {
    # stuff before

    # Location of the active script. When ManageSieve is used this is actually
    # a symlink pointing to the active script in the sieve storage directory.
    sieve=~/.dovecot.sieve

    #
    # The path to the directory where the personal Sieve scripts are stored. For
    # ManageSieve this is where the uploaded scripts are stored.
    sieve_dir=~/sieve
  }
  ```

Restart *Dovecot*.

Note: If you're using *Postfix* as MTA, you have to use *Dovecot*'s local delivery agent otherwise your emails won't get filtered. See *Recommended: using Dovecot's LDA* to get information on how to activate it.

2.3 Web servers

2.3.1 Apache2

mod_wgsi

First, make sure that *mod_wsgi* is installed on your server.

Create a new virtualhost in your Apache configuration and put the following content inside:

```
<VirtualHost *:80>
  ServerName <your value>
  DocumentRoot <path to your site's dir>

  Alias /media/ <path to your site's dir>/media/
  <Directory <path to your site's dir>/media>
    Order deny,allow
    Allow from all
  </Directory>

  Alias /sitestatic/ <path to your site's dir>/sitestatic/
  <Directory <path to your site's dir>/sitestatic>
    Order deny,allow
    Allow from all
  </Directory>

  WSGIScriptAlias / <path to your site's dir>/wsgi.py
</VirtualHost>
```

This is just one possible configuration.

Note: *Django* 1.3 users, please consult this page, it contains an example *wsgi.py* file.

Note: You will certainly need more configuration in order to launch Apache.

mod_python

First, make sure that *mod_python* is installed on your server.

Create a new virtualhost in your Apache configuration and put the following content inside:

```
<VirtualHost *:80>
  ServerName <your value>
  DocumentRoot <path to your site's dir>

  <Location "/">
    SetHandler mod_python
    PythonHandler django.core.handlers.modpython
    PythonPath "['<path to directory that contains your site's dir>'] + sys.path"
    SetEnv DJANGO_SETTINGS_MODULE <your site's name>.settings
  </Location>

  Alias "/sitestatic" "<path to your site's dir>/sitestatic"
  <Location "/sitestatic/">
```

```
    SetHandler None
  </Location>

  Alias "/media" "<path to your site's dir>/media"
  <Location "/media/">
    SetHandler None
  </Location>
</VirtualHost>
```

This is just one possible configuration.

Note: You will certainly need more configuration in order to launch Apache.

2.3.2 Nginx

Nginx is a really fast HTTP server. Associated with Green Unicorn, it gives you one of the best setup to serve python/Django applications. Modoboa's performances are really good with this configuration.

To use this setup, first download and install those softwares (Install gunicorn and install nginx).

Then, use the following sample *gunicorn* configuration (create a new file named *gunicorn.conf.py* inside Modoboa's root dir):

```
backlog = 2048
bind = "unix:/var/run/gunicorn/modoboa.sock"
pidfile = "/var/run/gunicorn/modoboa.pid"
daemon = True
debug = False
workers = 2
logfile = "/var/log/gunicorn/modoboa.log"
loglevel = "info"
```

To start gunicorn, execute the following commands:

```
$ cd <modoboa dir>
$ gunicorn_django -c gunicorn.conf.py
```

Now the *nginx* part. Just create a new virtual host and use the following configuration:

```
upstream modoboa {
      server      unix:/var/run/gunicorn/modoboa.sock fail_timeout=0;
}

server {
      listen 443;
      server_name <host fqdn>;
      root <modoboa's root dir>;

      access_log  /var/log/nginx/<host fqdn>.access.log;
      error_log /var/log/nginx/<host fqdn>.error.log;

      location /sitestatic/ {
              autoindex on;
      }

      location /media/ {
              autoindex on;
```

```
        }

        location / {
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_redirect off;
                proxy_set_header X-Forwarded-Protocol ssl;
                proxy_pass http://modoboa;
        }
}
```

Paste this content to your configuration (replace values between <> with yours), restart *nginx* and enjoy a really fast application!

CHAPTER

THREE

EXTENDING MODOBOA

3.1 Adding a new plugin

3.1.1 Introduction

Modoboa offers a plugin API to expand its capabilities. The current implementation provides the following possibilities:

- Expand navigation by adding entry points to your plugin inside the GUI
- Access and modify administrative objects (domains, mailboxes, etc.)
- Register callback actions for specific events

Plugins are nothing more than Django applications with an extra piece of code that integrates them into Modoboa. Usually, the *__init__.py* file will contain a complete description of the plugin:

- Admin and user parameters
- Observed events
- Custom menu entries

The communication between both applications is provided by *Available events*. Modoboa offers some kind of hooks to let plugin add custom actions.

The following subsections describe plugin architecture and explain how you can create your own plugin.

3.1.2 The required glue

To create a new plugin, just start a new django application like this (into modoboa's directory):

```
$ python manage.py startapp
```

Then, you need to register this application using the provided API. Just copy/paste the following example into the *__init__.py* file of the future extension:

```
from modoboa.extensions import ModoExtension, exts_pool

class MyExtension(ModoExtension):
    name = "myext"
    label = "My Extension"
    version = "0.1"
    description = "A description"
    url = "myext_root_location" # optional, name is used if not defined
    def init(self):
        """This method is called when the extension is activated.
        """
        pass

    def load(self):
        """This method is called when Modoboa loads available and activated plugins.

        Declare parameters and register events here.
        """
        pass

    def destroy(self):
        """This function is called when a plugin is disabled from the interface.

        Unregister parameters and events here.
        """
        pass

exts_pool.register_extension(MyExtension)
```

Once done, simply add your plugin's module name to the `INSTALLED_APPS` variable located inside *settings.py*. Optionaly, run `python manage.py syncdb` if your plugin provides custom tables and `python manage.py collectstatic` to update static files.

3.1.3 Parameters

A plugin can declare its own parameters. There are two levels available:

- 'Administration' parameters : used to configure the plugin, editable inside the *Admin* > *Settings* > *Parameters* page,
- 'User' parameters : per-user parameters (or preferences), editable inside the *Options* > *Preferences* page.

Playing with parameters

To declare a new administration parameter, use the following function:

```
from modoboa.lib import parameters

parameters.register_admin(name, **kwargs)
```

To declare a new user parameter, use the following function:

```
parameter.register_user(name, **kwargs)
```

Both functions accept extra arguments listed here:

- `type` : parameter's type, possible values are : `int`, `string`, `list`, `list_yesno`,
- `deflt` : default value,
- `help` : help text,
- `values` : list of possible values if `type` is `list`.

To undeclare parameters (for example when a plugin is disabled is disabled from the interface), use the following function:

```
parameters.unregister_app(appname)
```

`appname` corresponds to your plugin's name, ie. the name of the directory containing the source code.

3.1.4 Custom administrative roles

Modoboa uses Django's internal permission system. Administrative roles are nothing more than groups (`Group` instances).

If an extension needs to add new roles, it needs to follow those steps:

1. Create a new `Group` instance. It can be done by providing fixtures or by creating it into the extension `init` function
2. A a new listener for the *GetExtraRoles* event that will return the group's name

3.2 Available events

3.2.1 Introduction

Modoboa provides a simple API to interact with events. It understands two kinds of events:

- Those that return a value
- Those that return nothing

Listening to a specific event is achieved as follow:

```
from modoboa.lib import events

def callback(*args):
  pass

events.register('event', callback)
```

You can also use the custom decorator `events.observe`:

```
@events.observe('event')
def callback(*args):
  pass
```

`event` is the event's name you want to listen to, `callback` is the function that will be called each time this event is raised. Each event impose to callbacks a specific prototype to respect. See below for a detailled list.

To stop listening to as specific event, you must use the `unregister` function:

```
events.unregister('event', callback)
```

The parameters are the same than those used with `register`.

Read further to get a complete list and description of all available events.

3.2.2 Supported events

AccountCreated

Raised when a new account is created.

Callback prototype:

```
def callback(account): pass
```

- `account` is the newly created account (`User` instance)

AccountDeleted

Raised when an existing account is deleted.

Callback prototype:

```
def callback(account): pass
```

- `oldaccount` is the account that is going to be deleted

AccountModified

Raised when an existing account is modified.

Callback prototype:

```
def callback(oldaccount, newaccount): pass
```

- `oldaccount` is the account before it is modified
- `newaccount` is the account after the modification

AdminMenuDisplay

Raised when an admin menu is about to be displayed.

Callback prototype:

```
def callback(target, user): pass
```

The `target` argument indicates which menu is being displayed. Possible values are:

- `admin_menu_box` : corresponds to the menu bar available inside administration pages
- `top_menu` : corresponds to the top black bar

See *UserMenuDisplay* for a description of what callbacks that listen to this event must return.

CanCreate

Raised just before a user tries to create a new object.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance

Return `True` or `False` to indicate if this user can respectively create or not create a new `Domain` object.

CheckExtraAccountForm

When an account is being modified, this event lets extensions check if this account is concerned by a specific form.

Callback prototype:

```
def callback(account, form): pass
```

- `account` is the `User` instance beeing modified
- `form` is a dictionnary (same content as for `ExtraAccountForm`)

Callbacks listening to this event must return a list containing one Boolean.

CreateDomain

Raised when a new domain is created.

Callback prototype:

```
def callback(user, domain): pass
```

- `user` corresponds to the `User` object creating the domain (its owner)
- `domain` is a `Domain` instance

CreateMailbox

Raised when a new mailbox is created.

Callback prototype:

```
def callback(user, mailbox): pass
```

- `user` is the new mailbox's owner (`User` instance)
- `mailbox` is the new mailbox (`Mailbox` instance)

DeleteDomain

Raised when an existing domain is about to be deleted.

Callback prototype:

```
def callback(domain): pass
```

- `domain` is a `Domain` instance

DeleteMailbox

Raised when an existing mailbox is about to be deleted.

Callback prototype:

```
def callback(mailbox): pass
```

- `mailbox` is a `Mailbox` instance

DomainAliasCreated

Raised when a new domain alias is created.

Callback prototype:

```
def callback(user, domain_alias): pass
```

- `user` is the new domain alias owner (`User` instance)
- `domain_alias` is the new domain alias (`DomainAlias` instance)

DomainAliasDeleted

Raised when an existing domain alias is about to be deleted.

Callback prototype:

```
def callback(domain_alias): pass
```

- `domain_alias` is a `DomainAlias` instance

DomainModified

Raised when a domain has been modified.

Callback prototype:

```
def callback(domain): pass
```

- `domain` is the modified `Domain` instance, it contains an extra `oldname` field which contains the old domain name

ExtDisabled

Raised just after an extension has been disabled.

Callback prototype:

```
def callback(extension): pass
```

- `extension` is an `Extension` instance

ExtEnabled

Raised just after an extension has been activated.

Callback prototype:

```
def callback(extension): pass
```

- `extension` is an `Extension` instance

ExtraAccountForm

Let extensions add new forms to the account edition form (the one with tabs).

Callback prototype:

```
def callback(user, account): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `account` is the account beeing modified (`User` instance)

Callbacks listening to the event must return a list of dictionnaries, each one must contain at least three keys:

```
{"id" : "<the form's id>",
 "title" : "<the title used to present the form>",
 "cls" : TheFormClassName}
```

ExtraAdminContent

Let extensions add extra content into the admin panel.

Callback prototype:

```
def callback(user, target, currentpage): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `target` is a string indicating the place where the content will be displayed. Possible values are : `leftcol`
- `currentpage` is a string indicating which page (or section) is displayed. Possible values are : `domains`, `identities`

Callbacks listening to this event must return a list of string.

ExtraDomainForm

Let extensions add new forms to the domain edition form (the one with tabs).

Callback prototype:

```
def callback(user, domain): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `domain` is the domain beeing modified (`Domain` instance)

Callbacks listening to the event must return a list of dictionnaries, each one must contain at least three keys:

```
{"id" : "<the form's id>",
 "title" : "<the title used to present the form>",
 "cls" : TheFormClassName}
```

FillAccountInstances

When an account is beeing modified, this event is raised to fill extra forms.

Callback prototype:

```
def callback(user, account, instances): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `account` is the `User` instance beeing modified
- `instances` is a dictionnary where the callback will add information needed to fill a specific form

FillDomainInstances

When a domain is beeing modified, this event is raised to fill extra forms.

Callback prototype:

```
def callback(user, domain, instances): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `domain` is the `Domain` instance beeing modified
- `instances` is a dictionnary where the callback will add information needed to fill a specific form

GetAnnouncement

Some places in the interface let plugins add their own announcement (ie. message).

Callback prototype:

```
def callback(target): pass
```

- `target` is a string indicating the place where the announcement will appear:
- `loginpage` : corresponds to the login page

Callbacks listening to this event must return a list of string.

GetExtraRoles

Let extensions define new administrative roles.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance corresponding to the currently logged in user

Callbacks listening to this event must return a list of 2uple (two strings) which respect the following format: `(value, label)`.

GetStaticContent

Let extensions add static content (ie. CSS or javascript) to default pages. It is pretty useful for functionalities that don't need a template but need javascript stuff.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance corresponding to the currently logged in user

Callbacks listening to this event must return a list of string.

MailboxAliasCreated

Raised when a new mailbox alias is created.

Callback prototype:

```
def callback(user, mailbox_alias): pass
```

- `user` is the new domain alias owner (`User` instance)
- `mailbox_alias` is the new mailbox alias (`Alias` instance)

MailboxAliasDeleted

Raised when an existing mailbox alias is about to be deleted.

Callback prototype:

```
def callback(mailbox_alias): pass
```

- `mailbox_alias` is an `Alias` instance

ModifyMailbox

Raised when an existing mailbox is modified.

Callback prototype:

```
def callback(newmailbox, oldmailbox): pass
```

- `newmailbox` is a `Mailbox` instance containing the new values
- `oldmailbox` is a `Mailbox` instance containing the old values

PasswordChange

Raised just before a *password change* action.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance

Callbacks listening to this event must return a list containing either `True` or `False`. If at least one `True` is returned, the *password change* will be cancelled (ie. changing the password for this user is disabled).

TopNotifications

Let extensions add custom content into the top bar.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance corresponding to the currently logged in user

Callbacks listening to this event must return a list of string.

UserLogin

Raised when a user logs in.

Callback prototype:

```
def callback(request, username, password): pass
```

UserLogout

Raised when a user logs out.

Callback prototype:

```
def callback(request): pass
```

UserMenuDisplay

Raised when a user menu is about to be displayed.

Callback prototype:

```
def callback(target, user): pass
```

The `target` argument indicates which menu is being displayed. Possible values are:

- `options_menu`: corresponds to the top-right user menu
- `uprefs_menu`: corresponds to the menu bar available inside the *User preferences* page
- `top_menu`: corresponds to the top black bar

All the callbacks that listen to this event must return a list of dictionnaries (corresponding to menu entries). Each dictionnary must contain at least the following keys:

```
{"name" : "a_name_without_spaces",
 "label" : _("The menu label"),
 "url" : reverse("your_view")}
```

CHAPTER

FOUR

ADDITIONAL RESOURCE

4.1 Migrating from other software

4.1.1 PostfixAdmin

Since version 0.8.5, Modoboa provides a simple script to migrate an existing PostfixAdmin (version 2.3.3+) database to a Modoboa one.

Note: This script is only suitable for a new installation.

First, you must follow the *Installation* step to create a fresh Modoboa database.

Once done, edit the *setting.py* file. First, add a new database connection named `pfxadmin` into the `DATABASES` variable corresponding to your PostfixAdmin setup:

```
DATABASES = {
  "default" : {
    # default connection definition
  },
  "pfxadmin" : {
    "ENGINE" : "<engine>",
    "NAME" : "<database name>",
    "USER" : "<database user>",
    "PASSWORD" : "<user password>",
  }
}
```

This connection should correspond to the one defined in PostfixAdmin's configuration file.

You are now ready to start the migration. Enter Modoboa's root directory and execute the following command:

```
$ PYTHONPATH=$PWD/.. DJANGO_SETTINGS_MODULE=modoboa.settings \
    ./tools/pfxadmin_migrate/migrate.py -r -p <directory that stores mailboxes>
```

Depending on how many domains/mailboxes your existing setup contains, the migration can be long. Just wait for the script's ending.

Once the migration has succeed, go the *Admin* > *Configuration* panel, click on the *admin* row and modify the value of `MAILDIR_ROOT` as follow:

```
MAILDIR_ROOT =
```

The corresponding field must be empty. Don't touch other fields except `PASSWORD_SCHEME`, if needed. (set it to the same method as the one used by PostfixAdmin, check its configuration file if you're not sure)

Click on the *Save* button.

The procedure is over, edit the *settings.py* file and:

- remove the `pfxadmin` database connection from the `DATABASES` variable
- remove the `'modoboa.tools.pfxadmin_migrate',` from the `INSTALLED_APPS` variable

You should be able to connect to Modoboa using the same credentials you were using to connect to PostfixAdmin.

4.2 Using the virtual machine

4.2.1 Introduction

A virtual machine with a ready-to-use *Modoboa* setup is available here. It is composed of the following components:

- Debian 6.0 (squeeze)
- Modoboa and its prerequisites
- MySQL
- Postfix
- Dovecot
- nginx and gunicorn

Actually, it is the result you obtain if you follow the official documentation.

The disk image is using the VMDK format and is compressed using bzip2. To decompress it, just run the following command:

```
$ bunzip2 modoboa.vmdk.bz2
```

If you can't use the vmdk format, you can use qemu to convert it to another one. For example:

```
$ qemu-img convert modoboa.vmdk -O qcow2 modoboa.qcow2
```

Then, just use your prefered virtualization software (qemu, kvm, virtualbox, etc.) to start the machine. You'll need to configure at least one bridged network interface if you want to be able to play with *Modoboa*, ie. your machine must be visible from your network.

The default network interface of the machine (`eth0`) is configured to use the DHCP protocol.

4.2.2 Connect to the machine

The following UNIX users are available if you want to connect to the system:

Login	Password	Description
root	demo	the root user
demo	demo	an unpriviliged user

To connect to *Modoboa*, first connect to the system and retrieve its current network address like this:

```
$ /sbin/ifconfig eth0
```

Once you know its address, open a web browser and go to this url:

```
http://<ip_address>/admin/
```

You should see the login page. Here are the users available by default:

Login	Password	Capabitilies
admin	password	Default super administrator. Can do anything on the admin but can't access applications
admin@demo.local	admin	Administrator of the domain *demo.local*. Can administrater its domain and access to applications.
user@demo.local	user	Simple user. Can access to applications.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*