How to configure Mobyle

Mobyle 1.0

Contents

Chapter 1

General Configuration

The Mobyle configuration is set in the file **$MOBYLEHOME/Local/Config/Config.py**. It is written in Python, so be very careful to validate the syntax of your file.

1.1 Link Mobyle with a web server

Three values have to be carefully updated in the configuration to integrate it correctly with your web server.
ROOT_URL: the root url and port of the Mobyle server. (mandatory)
HTDOCS_PREFIX: the extra path to the htdocs Mobyle directory. (mandatory)
CGI_PREFIX: the extra path to the cgi-bin Mobyle directory. (mandatory)

You can potentially use any web server, just as long as it is multithreaded and CGI-enabled. Following are two configuration examples that use Apache 2, one that hosts Mobyle in a virtual host, the other in a subdirectory.

1.1.1 Example of Mobyle configuration using a virtual host

1.1.1.1 Apache configuration

```
<VirtualHost 192.168.0.3:83>
        ServerName server.domain.ext:83
        ScriptAlias "/cgi-bin" "/var/www/localhost/cgi-bin/mobyle/"
        DocumentRoot "/var/www/localhost/htdocs/mobyle/"
        DirectoryIndex index.html index.xml
        ErrorLog "/var/log/apache2/mobyle_error_log"
        TransferLog "/var/log/apache2/mobyle_acces_log"
</VirtualHost>
```

1.1.1.2 Mobyle installation

```
python setup.py install --install-htdocs=/var/www/localhost/htdocs/mobyle/ \
                                    --install-cgis=/var/www/localhost/cgi-bin/mobyle/\
                                    --install-core=/any/where
```

1.1.1.3 Mobyle configuration

```
ROOT_URL = ‘‘http://server.domain.ext:83’’
HTDOCS_PREFIX = ‘‘’’
CGI_PREFIX = ‘‘cgi-bin/mobyle’’
```

1.1.2 Example of Mobyle configuration using a web subdirectory

1.1.2.1 Apache configuration

```
ServerName server.domain.ext
ScriptAlias "/cgi-bin" "/var/www/localhost/cgi-bin/"
```

```
DocumentRoot "/var/www/localhost/htdocs/"
DirectoryIndex index.html index.xml
```

1.1.2.2 Mobyle installation

```
python setup.py install --install-htdocs=/var/www/localhost/htdocs/mobyle/ \
                                    --install-cgis=/var/www/localhost/cgi-bin/mobyle/\
                                    --install-core=/any/where
```

1.1.2.3 Mobyle configuration

```
ROOT_URL = ‘‘http:mobyle.mydomain.ext/mobyle’’
HTDOCS_PREFIX = ‘‘mobyle’’
CGI_PREFIX = ‘‘cgi-bin/mobyle’’
```

1.1.3 Additional web server configuration

1.1.3.1 Specific MIME types

In order to be able to upload/visualize some specific data types, such as PDB files, you need to overload their mime type. The default is the `chemical/x-pdb` in `/etc/mime.types`, but this mime type causes most browsers to try opening these files with an external tool.

As a workaround, you can force these files to be sent as “plain text” files by declaring them as `text/plain` files. In Apache, for instance, this is done by adding this directive in the `/etc/apache2/mods-available/mime.conf` Apache configuration file:

```
"AddType text/plain .pdb".
```

1.1.3.2 Download button

The ”save” button that is available in job results and data bookmarks can automatically open a ”save as” prompt (instead of opening the file in the browser), given that you use a little trick to the web server configuration, that modifies the HTTP response headers.

For instance, in Apache, you can add these few lines in an `.htaccess` file (if your general Apache configuration permits it) in the directory where the jobs are stored (–install-hdocs value + ’/data/jobs’ by default).

```
RewriteEngine on
RewriteCond    %{REQUEST_URI}  ^HTDOCS_PREFIX/data/jobs(\.*)
RewriteCond %{QUERY_STRING} ^save$
RewriteRule   (.*)/([^/]+)$    $1/$2 [E=SAVEDFILENAME:$2]
Header set Content-Disposition "attachment; filename=\"%{SAVEDFILENAME}e\"" env=SAVEDFILENAME
```

1.1.3.3 Directory indexes

As the jobs are stored in web-accessible subtrees, data confidentiality relies on the use of a unique key as the job identifier. Thus, it is strongly recommended to forbid the directory indexes of the `"[--install-htdocs]/data/jobs"` subtree. You can do it in the Apache general configuration or in `.htaccess` files with the directive `Options -Indexes`.

1.1.3.4 Hidden files

In job directories, Mobyle uses some ”hidden files” for administration purposes. - the unix command generated (the `.command` file) - some information relative to the job (the `.admin` file) - some internal informations to Mobyle execution (the `.forChild.dump` file or the `ADMINDIR` subtree)

These files are not intended to be viewed by the users, and it is thus highly recommended to mask them. You can mask them with a rewrite rule placed in an `.htaccess` file in the `[--install-htdocs]/data/jobs` subtree.

```
# Do not show hidden files content
RewriteCond     %{REQUEST_URI}  /\. [OR]
RewriteCond     %{REQUEST_URI}  ADMINDIR
RewriteRule     .*              - [F,L]
```

1.2 Mail

Mobyle occasionally sends emails to users, to:

- validate their email adresses when creating "authenticated" accounts,
- send job status or help request notifications.

It is mandatory to configure correctly this mail to be able to run Mobyle. The values are the following:
MAILHOST: the mail transfert agent used by Mobyle to send an email. (mandatory)
MAXMAILSIZE: if the results size is over MAXMAILSIZE, job results are not sent. Rather, only a notification of the end of his job is send to the user (containing a link to download it), in bytes. (default=2097152 (2Mo))
MAINTAINER: the emails list which will receive alert emails when problems occur in Mobyle. (mandatory)
HELP: the email adress that receives help requests from users. (mandatory)
SENDER: the email address representing Mobyle which sends long job notifications, results etc... (for further details, see the MailTemplate section)

1.3 Execution system

Mobyle can execute jobs either locally or on the SGE or PBS Distributed Resource Management systems. The execution system has been rewritten to be highly flexible, so its configuration has completely changed. Please refer to the execution_system documentation to learn how to configure it.

1.4 Logging

LOGDIR is the directory where are located the different file loggers. Beware, given that the default directory is `/dev/null`, tracing potential problems can be hard if you do not set this value. The most important log files in this directory are:

- `access_log`: to log the launched jobs,
- `error_log`: to log the errors that occur on the Mobyle server,
- `build_log`: to log all the steps leading to the construction of the command line (when debug >= 2)

If **ACCOUNTING** is set to True an `account_log` will be created to log some statistics about jobs. This log file can be used to fine-tune the execution system.

1.5 Data converter management

You can define with the **DATACONVERTER** variable which converter(s) you want to use to manage the format conversion and verification properties.
For each Datatype, you can provide an ordered list of converter(s) as follow:

```
DATACONVERTER = {
'Datatype1': [ converter1_class('/path/to/bin/converter1'),
  converter2_class('/path/to/bin/converter2')     ] ,
'Datatype2': [ converter3_class('/path/to/bin/converter3') ]
               }
```

Mobyle includes by default two ready-to-use converter classes: squizz_alignment and squizz_sequence to manage respectively the Alignment and Sequence formats with the `squizz` program. The use of textttsquizz is highly recommended in production environments.

The existing converter classes are located in `$MOBYLEHOME/Src/Mobyle/Converter`.

Example to use the provided converters:

```
DATACONVERTER = {
                'Sequence': [ squizz_sequence('/path/to/bin/squizz') ] ,
                'Alignment': [ squizz_alignment('/path/to/bin/squizz') ]
                }
```

1.6 Debug

DEBUG allows to set the default debug level in Mobyle. The default debug level is 0. Beware, only debug levels 0 and 3 allow the execution of a job.

Level 0 used in production:

- the command line is built
- the build log is NOT filled
- the job is executed

Level 1 to test a program definition XML (e.g. python syntax in code, precond. . .):

- the command line is built
- the build log is NOT filled
- the job is NOT executed

Level 2 to debug a program definition XML:

- the command line is built
- the build log is filled
- the job is NOT executed

Level 3 to test the program definition XML, the job execution and its results:

- the command line is built
- the build log is filled
- the job is executed

To test/debug a program or its interface, the debug level can be overloaded for this service using the feature **PARTICULAR_DEBUG** in conjunction with **AUTHORIZED_SERVICES**, which allows to define restricted access by IP address (see 1.13.3). To test the execution and results, set the **PARTICULAR_DEBUG** to 2 or 3 for the program in question and restrict the access to this program to your own machine. This way, you will be the only one who can access the interface in the portal, and the `build_log` will register all steps of the command line generation, which can be useful to debug an interface.

In the following example the general debug level is 0 except for `clustalw` which has a debug level set to 2.

```
DEBUG= 0
PARTICULAR_DEBUG={ 'clustalw' : 2 }
```

Warnings:

1. during the debugging phase, don't forget to reinstall the XML file after each modification with `mobdeploy`:

   ```
   python mobdeploy.py -s local -p myNewProgram deploy
   ```

2. once the debugging phase has been completed, don't forget to remove the restricted access and set the debug level to 0.

1.7 Binary path

BINARY_PATH is a list of strings representing the paths where the program binaries can be found. Each element of the list must be a valid path. The order of the elements is kept to build the final path. These paths are added before the canonical PATH. By default, it is empty.

Example: here we add `/usr/local/bin` to $PATH:

```
BINARY_PATH = ['/usr/local/bin']
```

1.8 Data banks

DATABANKS_CONFIG describes the locally available databanks to fetch entries from, using the "databox". It is a dictionary of dictionaries, such that:

- the **key** is the **name of the bank**, and the value is a dictionary where
- **'dataType'** is the Mobyle dataType of the data stored in the bank,
- **'bioTypes'** is the list of Mobyle bioTypes for the data stored in the bank,
- **'label'** is the label of the bank, as shown to the user in the portal,
- and **'command'** is a string template that describes how to generate a command line that will retrieve the bank entry(ies). The **db** key is the key of the bank (if it is necessary to specify it in the command line), and **id** key is the requested value.

Example:

```
DATABANKS_CONFIG = { 'WGS' : {  'dataType' : 'Sequence',
                                'bioTypes' : [ 'Nucleic' ],
                                'label' : 'Genbank - Whole Genome Shotgun',
                                'command' : [ 'golden', '%(db)s:%(id)s' ] },
                     'PDB' : { 'dataType' : '3DStructure',
                                'bioTypes' : [ 'Protein' ],
                                'label' : 'Protein Data Bank',
                                'command' : [ 'PDBGet.py', '%(id)s' ] }
                   }
```

1.9 Statistics

GACODE : Google Analytics. Setting this code will automatically configure Mobyle tracking on Google Analytics. For more information about how to set up a Google Analytics account, please refer to http://www.google.com/analytics/

1.10 Welcome page configuration

The welcome page of the Mobyle portal can now be customized, to display the contents of an "external" HTML page or an ATOM news feed.

WELCOME_CONFIG is a dictionary with 2 entries:

- **'url'** : the url of the document to include in the portal welcome page,
- **'format'**: with 2 possible values:
  - **'html'** if the url is the location of an html page,
  - **'atom'** if the url is the location of an ATOM feed.

An example file, `news_ex.atom`, is available in the Example/Local directory.

1.11 User authentication and management

User actions in the portal, such as job submissions, can be configured to require some informations such as a valid email, to be able to contact the user if necessary.

OPT_EMAIL: allow users to run jobs without having to provide an email. (default = False)
PARTICULAR_OPT_EMAIL: per-program override of the previous option.

Example: the email is mandatory to run any program, except for golden (a very short program).

```
OPT_EMAIL = False
PARTICULAR_OPT_EMAIL = { 'golden' : True }
```

1.11.1 General e-mail validation configuration

User-provided emails can be validated beyond syntax check by performing a DNS resolution on the domain part of the user email, to ensure that a valid corresponding mail server exists, in order to limit fake user email addresses. This is done by setting **DNS_RESOLVER** to True. In this case the **dnspython** library must be installed. By default, domain name resolution is not performed.

1.11.2 User accounts management

Each user has a space to store his data. This space can be temporary (during the working session) or persistent. The temporary space is created when a user connects to the portal and is accessible during the work session. These temporary accounts are refered to in the configuration as **anonymous sessions**. Persistent accounts are refered to as authenticated sessions, as they should require a higher level of validation for user informations (such as emails). Such accounts allow the user to access his workspace across several browser sessions, by signin in in the portal.

Administrators can enable/disable the use of anonymous and authenticated sessions by setting the **ANONYMOUS_SESSION** and **AUTHENTICATED_SESSION** values.

ANONYMOUS_SESSION can take 3 values :

- 'no' : anonymous sessions are not allowed,
- 'yes' : anonymous sessions are allowed, without any verification,
- 'captcha' : anonymous sessions are allowed, but with a captcha challenge (default value).

AUTHENTICATED_SESSION can also take 3 values :

- 'no' : the authenticated sessions are not allowed,
- 'yes' : the authenticated sessions are allowed and activated, without any restriction,
- 'email' : the authenticated sessions are allowed but an email confirmation is needed to activate it (default value).

User workspaces are allocated a given amount of disk space. This disk space stores mainly their data bookmarks, which they can reuse across multiple tools. Note that this disk space **does not** include the job files, which are stored separately, and only "linked" from the user account. This size can be set using **SESSIONLIMIT**, which has a default value of 50Mib.

OPENID : activate OpenID authentication. When the OpenID identification system is enabled, users can connect without any registration: their authenticated session is automatically created and activated (default=False).

1.11.3 Jobs management

To limit disk space problems generated by excessively large jobs, or by programming bugs (e.g., execution loops which output ever-growing files), the **FILELIMIT** value allows to define the maximum size for a file generated by a job **if and only if the job is executed using the OS batch system, "SYS"**. If you use a distributed resource management system such as SGE or Torque/PBS, please refer to its documentation.

Jobs can be configured to have a limited "storage" time, meaning that once they are finished, they are kept on the server for a limited number of days before being removed. Jobs should be removed using the mobclean tool (see Tools/mobclean and Tools/README), which can be started periodically from a cron. To configure this "cleaning" tasks, please set the **RESULT_REMAIN** value which stores the lifetime of a finished job in days (default value=10 days).

Users are notified by email that their job is finished only if its execution time is superior to a given limit. This limit is set using **EMAIL_DELAY** in seconds, and the default value is 20s.

1.12 Portal settings

The Mobyle portal includes a javascript-based polling mechanism which regularly updates the user workspace by getting its contents from the server. The frequency of these requests can be set with **REFRESH_FREQUENCY**, in seconds. Its default value is 240 seconds, meaning the automatic workspace refresh is launched every 4 minutes. User data (parameter values or job results) are displayed in textarea elements in the portal (if text-based). However, this preview/editing mechanism can be problematic if the data file to be displayed is too large. Hence, the portal does not display the contents of a data file if its size is above a given limit. The size limit can be set with **PREVIEW_DATA_LIMIT**, and the default size is 1048576 octets (1 Mib).

1.13 Services management

1.13.1 Deployment

Before deployment, the XML descriptions of the services used in Mobyle are located in two directories:

- `$MOBYLEHOME/Services/` intended to store service definitions provided by Mobyle maintainers,
- `$MOBYLEHOME/Local/Services/` intended to store your own service definitions.

The use of two directories allows to maintain a list of service definitions from an external source located in `$MOBYLEHOME/Services`, without any risk to loose local modifications located in `$MOBYLEHOME/Local/Services`. Beware, if there are 2 services, one in `$MOBYLEHOME/Local/Services` and the other in `$MOBYLEHOME/Services`, with the same name, only the description in `$MOBYLEHOME/Local/Services` will be deployed.

`$MOBYLEHOME/Services/` contains 4 subdirectories:

- `Programs`: to store program definitions,
- `Workflows`: to store workflow definitions,
- `Viewers`: to store viewer definitions and dependencies.
- `Entities`: to store pieces of XML shared by several XML files, such as package information. (see `how_to_write_a_xml.pdf`)

`$MOBYLEHOME/Local/Services/` contains 5 subdirectories:

- `Programs`: to store program definitions
- `Workflows`: to store workflow definitions
- `Viewers`: to store viewer definitions and dependencies
- `Entities`: to store pieces of XML shared by several XML, like packages or nucleic databanks available
- `Env`: to store some environment variables required by some programs (BLASTDB ...)

1.13.1.1 Configuration

There are 2 kinds of services:

local services corresponding to services executed on the local Mobyle server

imported services corresponding to services appearing on the local Mobyle portal but executed on another Mobyle server

Whatever the service is, it must be deployed to be invoked in the Mobyle portal. This deployment operation will:

1. get the XML files (on your computer or the distant server for imported services) and resolve the entities
2. validate of the service descriptions
3. transform the descriptions to make them ready to use by Mobyle (to lower the burden of the portal)
4. copy the descriptions in the HTTP-accessible area for web publication

The `mobdeploy` script located in `$MOBYLEHOME/Tools` allows to deploy or undeploy a service. It needs the Mobyle configuration file `$MOBYLEHOME/Local/Config/Config.py` to determine:

- which services to deploy,
- which services are imported ones.

1.13.1.2 Locals services

The local service descriptions are store in 2 directories (as describe in 1.13.1). However Mobyle can be configured to deploy only a subset of these descriptions by using the following 2 directives:

LOCAL_DEPLOY_INCLUDE indicating which descriptions must be deployed

LOCAL_DEPLOY_EXCLUDE indicating which descriptions must not be deployed

These 2 directives are python dictionaries with the 3 following keys: 'programs', 'workflows','viewers' and the values are lists of corresponding services to deploy or not. Joker can be used to defined a set of services. By default all definitions found are deployed because of the use of the wildcard * in each list of each entry of the **LOCAL_DEPLOY_INCLUDE** dictionary.

Beware, Mobyle evaluates **LOCAL_DEPLOY_INCLUDE** first meaning that if the same service appears in the two lists, it will be excluded because **LOCAL_DEPLOY_EXCLUDE** has the final say.

```
LOCAL_DEPLOY_INCLUDE = { 'programs' : [ '*' ] ,
                         'workflows': [ '*' ] ,
                         'viewers'  : [ '*' ]
                       }

LOCAL_DEPLOY_EXCLUDE = { 'programs' : [ '' ] ,
                         'workflows': [ '' ] ,
                         'viewers'  : [ '' ]
                       }
```

To illustrate how these two directives work, let's imagine that a portal proposes `hmmalign`, `hmmconvert`, `hmmemit`, `hmmfetch` and `muscle` as programs. The definitions of `hmmalign`, `hmmconvert`, `hmmemit`, `hmmfetch` `hmmscan`, `hmmsim` and `hmmstat` are provided in the `pasteur-programs` package.

```
LOCAL_DEPLOY_INCLUDE = { 'programs' : [ 'hmm*' , 'muscle' ] ,# will deploy all
                                                             # programs beginning
                                                             # with hmm plus
                                                             # muscle
                         'workflows': [ '*' ] ,
                         'viewers'  : [ '*' ] }

LOCAL_DEPLOY_EXCLUDE = { 'programs' : [ 'hmms*' ] , # all programs beginning with
                                                    # hmms will be removed from
                                                    # the list of programs to
```

```
                                                # deploy so 'hmmscan',
                                                # 'hmmsim', 'hmmstat' won't
                                                # appear in the portal
                        'workflows': [ '' ] ,
                        'viewers'  : [ '' ]  }
```

1.13.1.3 Imported services

Contrary to the local services, there is no implicit mechanism to deploy imported services. Every imported service to deploy must be explicitly specified. Additional information is also required, such as the name of the portal, where to find the definitions .... The following example gives the configuration of a hypothetical Mobyle server called portal1 importing services from Institut Pasteur.

```
PORTALS={
        'portal1': {                            # this is a free label which
                                                # will appear in the portal
            'url': 'http://mydomain.ext/cgi-bin', # the url of the portal.py
                                                #
            'help' : 'user@mydomain.ext',       # where to have help about
                                                # jobs.
            'repository':'http://rita.sis.pasteur.fr/', # where are located the
                                                        # sessions, jobs and
                                                        # services.
            'services':{
                  'programs': ['program1' , 'program2' , ... ], # the list of programs
                                                                # to import
                                                                # from portal1
                  'workflows': [ 'workfA' , 'workfB' , ... ]    # the list of workflows
                                                                # to import
                                                                # from portal1
                        } #the viewers cannot be imported.
             } ,
             # import from the Institut Pasteur's portal
             'pasteur': {
                        'url': 'http://mobyle.pasteur.fr/cgi-bin',
                        'help': 'mobyle@pasteur.fr',
                        'repository': 'http://mobyle.pasteur.fr',
                        'services': { 'programs' : [ 'pars', 'protopars', 'mix']}
        }
}
```

A more complete documentation about Mobyle network is available in `mobyle_network_overview.pdf`

1.13.1.4 mobdeploy usage

Before running `mobdeploy`, make sure that the current user (e.g. apache user) has permissions to read and write the files belonging to the Mobyle user.

`Usage: mobdeploy [options] cmd`

Here are the available commands:

- **deploy**: deploy services available options [-s –server, -p –programs , -w –workflows , -v –viewers , -f –force , -V –verbose]
- **clean**: clean the repository available options [-s –server, -p –programs , -w –workflows , -v –viewers , -f –force , -V –verbose]
- **index**: generate indexes available options [-V –verbose]

And options:

- `-s --server` specifies the names (comma separated list) of the servers on which the command is applied. The keyword `all` means all servers as defined in the **PORTALS** dictionay 1.13.1.3 in `$MOBYLEHOME/Local/Conf`
- `-p --programs` specifies the names (comma separated list) of the programs the command is applied to. The keyword `all` means all programs as defined in `$MOBYLEHOME/Local/Config/Config.py`

- `-w --workflows` specifies the names (comma separated list) of the workflows the command is applied to. The keyword `all` means all workflows as defined in `$MOBYLEHOME/Local/Config/Config.py`
- `-v --viewers` specifies the names (comma separated list) of the viewers on which the command is applied. The keyword `all` means all viewers as defined in `$MOBYLEHOME/Local/Config/Config.py`
- `-V --verbose` increases the verbosity. 3 levels are available: `warning`, `info` and `debug`. Default is `warning`.

After deployment, the services are stored in a directory hierarchy with the following schema:

- the root directory is `$MOBYLE_ROOT_URL/HTDOC_PREFIX/data/services/`
- inside the root directory, there is one directory for each server:
  - your Mobyle server is called "local"
  - all other directories are named as specified in the **PORTALS** dictionary set in the `$MOBYLEHOME/Local/Co`
- in each server directory, there are the following subdirectories: programs, workflows and viewers (local server only) if some programs, workflows and viewers have respectively been deployed for this server. The descriptions of the corresponding services are inside those directories.

A full description of `mobdeploy` is available in `$MOBYLEHOME/Tools/README`.

1.13.2 Disabling services execution

It is sometimes necessary to disable jobs execution, for maintenance operations for instance. You can disable a given service by adding it in the **DISABLED_SERVICES** list or disable all programs by setting **DISABLE_ALL** to True to forbid the execution of any service. This mechanism prevents all new job submissions, portal-wide or service-specific, without any need to remove it from the portal. Conversely, new submissions can be enabled back by setting **DISABLE_ALL** to False or updating the services listed in **DISABLED_SERVICES**. By default, nothing is disabled (default DISABLE_ALL=False, DISABLED_SERVICES=[ ]). Please note that even though no job submission is accessible, the form itself can still be displayed.

1.13.3 Services access restriction

The access to some services can be restricted based on their IP address. By default, all the programs available are usable by any user. If you need to filter this access, based on some license restrictions for example, you can use **AUTHORIZED_SERVICES**. This dictionary defines the list of IPs and IP "masks" which can access specific services.

Example: here toppred can be accessed by any machine with IP 125.234.60.18 or any IP with "mask" 125.234.60.*

```
AUTHORIZED_SERVICES = {
   'http://myMobyle.mydomain.fr/data/programs/toppred.xml' : [ '125.234.60.18', '125.234.60.*']
                       }
```

Example: you have a new version of neighor program and you want to test it so you restrict the access to your IP 125.234.60.18 and set the debug level of the program to 2 doing this:

```
PARTICULAR_DEBUG={ 'neighbor' : 2 }

AUTHORIZED_SERVICES = {
   'http://myMobyle.mydomain.fr/data/programs/neighbor.xml' : [ '125.234.60.18']
                       }
```

1.14 *MobyleNet* functionality

Mobyle includes a set of functionalities allowing the use of services available on remote Mobyle servers. *MobyleNet*-ed servers are connected using HTTP, to enable remote job submissions or results retrievals for instance.

The configuration of the MobyleNet functionality is set in the following values:

- **PORTALS**: The Mobyle server from which you want to import services. To define a server, you need to specify this 6 fields:
  - name: the nickname you give to a Mobyle server. The services imported from this server will be labeled with this name in the Services panels of the portal.
  - url: the url of the portal (minus '/portal.py').
  - help: the email address to which help requests should be sent (should probably be the email set in the **HELP** configuration value of the corresponding server).
  - repository: the container url where the remote data (service descriptions, jobs) are stored.
  - services: the dictionary of services you want to import from this server, defining two lists, one with the 'programs' key, the other with the 'workflows' key.
- **EXPORTED_SERVICES**: The list of local services you want to enable from other portals.
- **PORTAL_NAME** is the name you want to give to your own portal, when seen from other MobyleNet nodes.

Beware, such a functionality does not mean you can import services from publicly available Mobyle portals without asking, as it can significantly impact the remote server load. Please consider asking the maintainer of the portal you wish to import programs from before doing so.

Example:

```
PORTALS={
         'mobyleA': {
            'url': 'http://mobyle.domain.ex/cgi-bin/MobylePortal',
            'help' : 'user@domain.ex',
            'repository': 'http://mobyle.domain.ex/data/programs/',
            'programs': ['clustalw-multialign'],
            'jobsBase': 'http://mobyle.domain.ex/data/jobs'
            } ,
        'univB': {
            'url': 'http://bio.univB.fr/cgi-bin/',
            'help' : 'mobyle-help@univB.fr',
            'repository': 'http://bio.univB.fr/Mobyle/programs',
            'programs': ['muscle' , 'sspro' ],
            'jobsBase': 'http://bio.univB.fr/Mobyle/Results'
            }
         }

EXPORTED_SERVICES = [ 'protpars' , 'dnapars', 'neighbor' ]

PORTAL_NAME = 'me'
```

Chapter 2

Mail Templates

Mobyle uses emails in several circumstances. You can tune the contents of each mail by modifying the corresponding template in `Local/mailTemplate.py`.

A template is composed of two parts, the header and the body. The header must contain the fields "From" and "Subject", and can contain the fields "Cc", "Bcc", "Reply-To" and "Organization". For each template, you can use some variables which will be expanded at runtime. All available variables are documented in `Local/mail.template.py`.

- **CONFIRM_SESSION** : if **AUTHENTICATED_SESSION** is set to 'email', an email is sent to the user to confirm his registration.
- **LONG_JOB_NOTIFICATION** : when a job is longer than **EMAIL_DELAY**, we consider this job as a "long" job, and an email is sent to the user to notify him that his job keeps running.
- **RESULTS_FILES** : once a "long" job is finished, all results are sent to the user as a zip archive (if an email was specified).
- **RESULTS_TOOBIG** : once a "long" job is finished, if the size of the results is bigger than **MAXMAILSIZE**, the URL of the results is sent to the user.
- **RESULTS_NOTIFICATION** : once a "long" job is finished, if the compression of the results cannot be performed, the URL of the results is sent to the user.
- **HELP_REQUEST** : whereas the previous emails are sent by "Mobyle" **SENDER** to the user, this email is sent by the user to the email address set in the **HELP** directive. The user triggers this action by clicking on the "ask for help" button in the results page.
- **HELP_REQUEST_RECEIPT**: this is a receipt sent to the user who asks for help, containing a copy of his request.
- **NEW_PASSWD**: text of the password modification notification sent by the `mobpasswd` tool.

Chapter 3

Local Policy

The `Local/Policy.py` module is used to overload internal Mobyle functions, to adapt them to your administration needs.

3.1 emailCheck

This function checks if the email is valid, according to the local rules. This function takes one argument, the *email* address, and must return one of the following values:

- **Mobyle.Net.EmailAddress.VALID** : if the email is valid,
- **Mobyle.Net.EmailAddress.INVALID** : if the email is rejected,
- **Mobyle.Net.EmailAddress.CONTINUE** : to continue further the email validation process.

This function is called after the syntax checking, black-list filter and, if enabled by your configuration, before the "DNS" checking.

3.2 authenticate

This function overloads the Mobyle authentication session method. You can provide here any custom authentication mechanism. This function takes 2 arguments, *login* and *password*, and must return one of the following values:

- **Mobyle.AuthenticatedSession.AuthenticatedSession.CONTINUE**: the method did not authenticate this login/passwd. The fall back method must be applied.
- **Mobyle.AuthenticatedSession.AuthenticatedSession.VALID**: the login/password has been authenticated and is valid.
- **Mobyle.AuthenticatedSession.AuthenticatedSession.REJECT**: the login/password is invalid.

Chapter 4

Black list

The file `Local/black_list.py` allows to forbid job submission to "non-desired" users. In this file 2 structures are defined to store emails or IP addresses for which you want to forbid job submissions.

4.1 users

The first structure, `users`, is a list of emails which are not allowed to run a job on your Mobyle server.
Example:

```
users = [ 'foo@bar.com', 'spam@eggs.fr']
```

4.2 host

The second structure, `host`, is a list of IP addresses from which users are not allowed to run a job on your Mobyle server. You can use * as a joker to black list a whole subnet. Example:

```
host = [ '192.168.3.1' , '192.168.2.*' ]
```

Chapter 5

Portal referencing in search engines

If you manage a publicly accessible Mobyle portal, you may be interested in having it referenced by various search engines. To facilitate this task, the Mobyle portal includes a sitemap-generating CGI. You can "declare" your sitemap to different search engines (refer to their respective webmaster help pages). This sitemap will facilitate the crawling of your Mobyle portal, by declaring a different page for each available program. For more detailed information about sitemaps, refer to http://www.sitemaps.org/. In Mobyle, this sitemap, `sitemap.py` is located in the same directory as the other cgi scripts.