# How to configure Mobyle

Mobyle 0.97

## Contents



# 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 project. ( mandatory )
**HTDOCS_PREFIX**: the extra path to the htdocs mobyle project. ( mandatory )

**CGI_PREFIX**: the extra path to the cgi mobyle project. ( 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

**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"
        CustomLog "/var/log/apache2/mobyle_acces_log" combined
</VirtualHost>
```

**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
```

**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

**Apache configuration**

```
ServerName server.domain.ext
ScriptAlias "/cgi-bin" "/var/www/localhost/cgi-bin/"
DocumentRoot "/var/www/localhost/htdocs/"
DirectoryIndex index.html index.xml
```

**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
```

**Mobyle configuration**

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

## 1.2 Mail

Mobyle occasionally sends emails to users, to validate their email adresses when creating "authenticated" accounts, to send job notifications 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. You can also extend it to use it with your own system, provided that you write the appropriate execution module and place it in the Src/Mobyle/Execution package.

**BATCH** is the value that sets the default execution system used to run jobs. It must be the name of a Python module in Src/Mobyle/Execution package ( except _batch.py and Dummy.py ). There are by default 3 available modules:

- SYS, if you don't have any particular execution system (default).
- SGE, if you use Sun Grid Engine to manage the execution of your jobs.
- PBS, if you use torque to manage the execution of your jobs.

For some programs you may want to use a different Execution system than the default one. You can configure this using the **PARTICULAR_BATCH** value (optional). It is a dictionary, where a key is a program name, and a value the corresponding execution system.

Example:

```
BATCH= ''SGE''
PARTICULAR_BATCH={''golden'' : SYS}
```

If your execution system use queues (SGE, PBS) you can define the default queue with **DEFAULT_Q** and override it for some special programs with **PARTICULAR_Q**.

Example:

```
DEFAULT_Q = 'local'
PARTICULAR_Q = {'golden' : 'short' }
```

### 1.3.1 SGE-specific configuration

**SGE_ROOT** is the path to the GridEngine root installation. **SGE_CELL** is the cluster name to be used. Of course, these attributes are only required if you use SGE.

### 1.3.2 PBS/Torque-specific configuration

**Q_PROPERTIES**={}
This attribute is used by the PBS class. For further information about PBS configurations see: http://mobyle.rpbs.univ-paris-diderot.fr/help/MobylePBSConfiguration.html

### 1.4 Logging

**LOGDIR** is the directory where are located the different file loggers. Beware, default is /dev/null, hence nothing will be logged, which makes it hard to trace potential problems.

- access_log: to log the jobs launched
- error_log : to log the MobyleError
- build_log : to log all the step leading to build 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 tune the execution system.

### 1.5 Sequence and alignment management

Mobyle can use 2 programs helper to manage the sequences and alignments format: readseq (the java version) and squizz. We strongly recommend squizz as we experienced lot of troubles with readseq.
**SEQCONVERTER** defines which converter will be used and where to find it.

Example: here we use both converters. In this case squizz will be used first, and if it fails to detect/convert the format, readseq will be used.

```
SEQCONVERTER = {'SQUIZZ': '/path/to/bin/squizz' ,
                'READSEQ': '/path/to/bin/jreadseq'
                }
```

### 1.6 Debug

**DEBUG** allows to set the default debug level in Mobyle. It can be overloaded on a per-program basis with **PARTICULAR_DEBUG**. There are 4 debug levels:

- 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

The default debug level is 0. Beware, only debug levels 0 and 3 allow the execution of a job.
To test/debug a program or its interface, the feature **PARTICULAR_DEBUG** is used in conjunction with **AUTHORIZED_SERVICES** which allows to define restricted access by IP address (See section Restriction programs access). To do this, set the **PARTICULAR_DEBUG** to 2 or 3 ( if you want to test the execution and results) for your program and restrict the access to this program to your own machine. You will then 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.

Example: here, all services have a debug level set to 0 except clustalw which is set to 2.

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

### 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 various utilities. It is a dictionnary of dictionnaries, such that:

- the **key** is the **name of the bank**, and the value is a dictionnary where
- **'dataType'** is the Mobyle dataType of the data which are stored in the bank,
- **'bioTypes'** is the list of Mobyle bioTypes for the data which are 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 requested bank, and **id** 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 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.

**Requiring e-mails to enable job submissions** **OPT_EMAIL**: set this to True to allow the users to run a job without specify any email. ( default = False )
**PARTICULAR_OPT_EMAIL**: you can override the general **OPT_EMAIL** option for a specific program.

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

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

**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.

**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.

**Jobs management** To limit disk space problems generated by very hungry users, or by execution loops which output ever-growing files, the **FILELIMIT** value lets you define what is the maximum size for a file generated by a job **if and only if the job is executed using the OS batch system**. 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 have finished they will be available on the server for a given number of days before to be removed. To configure this "cleaning" tasks, please set the **RESULT_REMAIN** value which stores the lifetime of a finished job in days (default value of 10 days). Jobs should be removed using the mobclean tool (see Tools/mobclean and Tools/README), which can be started periodically from a cron.

Users are notified by email that their job has been finished if this job is considered as "long". The "long" job limit is set using **TIMEOUT** n seconds, and the default value is 60s. For technical reasons, the "floor" value is 10 seconds, meaning that it cannot be lower than 10 seconds: if you set it lower, Mobyle will set it to 10 seconds.

## 1.10 Portal settings

The Mobyle portal includes a javascript-based polling mechanism which regularly launches user workspace updates 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).

## 1.11 Programs management

### 1.11.1 Deployment

Basically, the xml descriptions of the programs that can be used in Mobyle are located in two directories: `Mobyle/Local/Programs` and `Mobyle/Programs`. To be able to use these programs from Mobyle, you need to "deploy" them, an operation which will copy them in the HTTP-accessible area and also index them for use in Mobyle.

While you can tune the program descriptions which are deployed from the `Programs` directory, all the ones from `Local/Programs` are deployed. The following configuration values are used to select the program descriptions from the `Programs` directory, :

- **LOCAL_DEPLOY_ORDER**: The order in which INCLUDE and EXCLUDE directives are evaluated (default=['include','exclude']).
- **LOCAL_DEPLOY_INCLUDE**: The list of programs descriptions to install (default=['*']).
- **LOCAL_DEPLOY_EXCLUDE**: The list of programs descriptions to not install (default=[]).

By default all program descriptions are deployed (include all, exclude nothing). For INCLUDE and EXCLUDE directives, shell jokers can be used. For instance, 'dna*' refers to all programs descriptions beginning by 'dna'....

Example: if you want deploy only "blast family" programs, you can configure mobyle like this:

```
LOCAL_DEPLOY_ORDER = ['exclude' ,'include']
LOCAL_DEPLOY_INCLUDE = ['blast*']
LOCAL_DEPLOY_EXCLUDE = ['*']
```

Example: if you want all programs except the blast family programs:

```
LOCAL_DEPLOY_ORDER = ['include','exclude']
LOCAL_DEPLOY_INCLUDE = ['*']
LOCAL_DEPLOY_EXCLUDE = ['blast*']
```

The mobdeploy tool, located in Tools subfolder, is the script to use to deploy the xml programs descriptions (for more details see associated README).

### 1.11.2 Disabling programs execution

It is sometimes necessary to disable the portal or a program for maintenance operations for instance. You can disable a program by adding its complete url in the **DISABLED_SERVICES** list or disable all programs by setting **DISABLE_ALL** to True to forbid to run any program. This mechanism prevents all new job submissions, portal-wide or program-specific, without any need to remove it from the portal. Conversely, new submissions can be enabled back by setting **DISABLE_ALL** to False or removing them from the **DISABLED_SERVICES** list. 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.

### 1.11.3 Restriction programs access

The access to some programs can be restricted based on their IP. By default all the programs available are usable by all users who can access your web server. But sometimes, due to some license restrictions for example, you need to filter this access. **AUTHORIZED_SERVICES** is a dictionary with the service names of programs to restrict as keys and the list of IPs which can access these programs as values. It associates the program URL to a list of IP adresses or simplified IP masks.

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']
                       }
```

### 1.12 *MobyleNet* functionality

Mobyle includes a set of functionalities which allow the use of programs which are 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 programs imported from this server will be labeled with this name in the programs panel in the portal.
  - url: the url of the portal.
  - help: the email adress to send help messages (should probably be the email set in the **HELP** configuration value of the corresponding server).
  - repository: the url where the program descriptions (XML files) are stored.
  - programs: the list of programs you want to import from this server.
  - jobsBase: the url of the jobs repository.
- **EXPORTED_SERVICES**: The list of local programs for which you want to enable remote call from other portals.

Beware, such a functionality does not mean you can import programs/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' ]
```

# 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 explained 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 **TIMEOUT**, we consider this job as a long job and a notification is sent to the user to notify him that his job keeps running.
- **RESULTS_FILES** : when a job is finished, all results are sent to the user as a zip archive (if an email was specified).
- **RESULTS_TOOBIG** : if the results size is bigger than **MAXMAILSIZE**, a notification of the end of the job and the URL of the results is send to the user.
- **RESULTS_NOTIFICATION** : if a trouble occurs during the results zipping, a notification of the end of the job and 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 **HELP**. 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.

# 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.

# 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 IPs for which you want 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 IPs 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.*' ]
```

# 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. Briefly, 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 CGIs.