
MacSyLib

Release 1.0.3

Bertrand Néron, Sopphe Abby

Aug 20, 2025

CONTENTS

1	User Guide	3
1.1	User Guide	3
2	Modeller Guide	53
2.1	Modeller Guide	53
3	Developer Guide	81
3.1	Developer Guide	81
4	Indices and tables	165
	Python Module Index	167
	Index	169



MacSyLib is a Python package library that allow to model and detect macromolecular systems, genetic pathways... by similarity search in prokaryotes datasets.

Note

If you use MacSyLib please cite:

Néron, Bertrand; Denise, Rémi; Coluzzi, Charles; Touchon, Marie; Rocha, Eduardo P.C.; Abby, Sophie S. MacSyFinder v2: Improved modelling and search engine to identify molecular systems in genomes. Peer Community Journal, Volume 3 (2023), article no. e28. doi : 10.24072/pcjournal.250. <https://peercommunityjournal.org/articles/10.24072/pcjournal.250/>

USER GUIDE

1.1 User Guide

1.1.1 Using MacSyLib

What's new in MacSyLib

V 1.0.3

Fix bug in *msl_data cite*

V 1.0.2

Fix bug in *msl_data* when library used by third partite as *MacSyFinder*.

V 1.0.1

Now MacSyFinder and MacSyLib are two separated packages.

MacSyFinder is build on top of MacSyLib.

This new architecture allow to create new tool on the top of MacSyLib.

provide 2 scripts

- *msl_data* formerly *macsydata*
- *msl_profile* formerly *macsyprofile*

features

- add new subcommand to *msl_data* *msl_data show* to show the structure of an installed package model *Models installation with msl_data*

For older changelog see <https://macsyfinder.readthedocs.io/en/latest/>.

Installation

MacSyLib works with models for macromolecular systems that are not shipped with it, you have to install them separately. See the *msl_data section* below.

MacSyLib dependencies

Python version ≥ 3.10 is required to run MacSyLib: <https://docs.python.org/3.10/index.html>

MacSyLib has one program dependency:

- the *Hmmer* program, version 3.1 or greater (<http://hmmer.org/>).

The *hmmsearch* program should be installed (*e.g.*, in the PATH) in order to use MacSyLib. Otherwise, the paths to this executable must be specified in the command-line: see the *command-line options*.

MacSyLib also relies on six Python library dependencies:

- colorlog
- colorama
- pyyaml
- packaging
- networkx
- pandas

These dependencies will be automatically retrieved and installed when using *pip* for installation (see below).

Note

If you intend to build and distribute new models you will need some other dependencies see modeler guide for installation.

Note

If you want to contribute to the *MacSyLib* code, check the guide lines ([CONTRIBUTING](#)) and specific procedure for *developer installation*.

MacSyLib Installation procedure

It is recommended to use *pip* to install the MacSyLib package.

Archive overview

- **doc** => The documentation in html and pdf
- **test** => All what is needed for unitary tests
- **src/macsylib** => The macsylib python library
- **setup.py** => The installation script (to include documentation)
- **pyproject.toml** => The project installation build tool
- **COPYING** => The licensing
- **COPYRIGHT** => The copyright
- **README.md** => Very brief MacSyLib overview
- **CONTRIBUTORS** => List of people who contributed to the code
- **CONTRIBUTING** => The guide lines to contribute to the code

Installation steps:

Make sure every required dependency/software is present.

By default MacSyLib will try to use *hmmsearch* in your PATH. If *hmmsearch* is not in the PATH, you have to set the absolute path to *hmmsearch* in a *configuration file* or in the *command-line* upon execution. If the tools are not in the path, some test will be skipped and a warning will be raised.

Perform the installation.

```
python3 -m pip install macsylib
```

If you do not have the privileges to perform a system-wide installation, you can either install it in your home directory or use a *virtual environment*.

installation in your home directory

```
python3 -m pip install --user macsylib
```

installation in a virtualenv

```
python3 -m venv macsylib
cd macsylib
source bin/activate
python3 -m pip install macsylib
```

To exit the virtualenv just execute the *deactivate* command. To use *macsylib*, you need to activate the virtualenv:

```
source macsylib/bin/activate
```

Then use *macsylib* as python library or the *msl_data* command line tool.

Note

Super-user privileges (*i.e.*, *sudo*) are necessary if you want to install the program in the general file architecture.

Note

If you do not have the privileges, or if you do not want to install MacSyLib in the Python libraries of your system, you can install MacSyLib in a virtual environment (<http://www.virtualenv.org/>).

Warning

When installing a new version of MacSyLib, do not forget to uninstall the previous version installed !

Uninstalling MacSyLib

To uninstall MacSyLib (the last version installed), run

```
(sudo) pip uninstall macsylib
```

If you have installed it in a virtualenv, just delete the virtual environment. For instance if you create a virtualenv name *macsylib*

```
python3 -m venv macsylib
```

To delete it, remove the directory

```
rm -R macsylib
```

From Conda/Mamba

From version 1.0, MacSyLib is packaged for Conda/Mamba .. code-block:: text

```
mamba install -c macsylib=x.x
```

Where *x.x* is the macsylib version you want to install

Models installation with *msl_data*

Once MacSyLib is installed you have access to an utility program to manage the models: *msl_data*

This script allows to search, download, install and get information from MacSyLib models stored on github (<https://github.com/macsy-models>) or locally installed. The general syntax for *msl_data* is:

```
msl_data <general options> <subcommand> <sub command options> <arguments>
```

To list all models available on *macsy-models*:

```
msl_data available
```

To search for models on *macsy-models*:

```
msl_data search TXSS
```

you can also search in models description:

```
msl_data search -S secretion
```

To install a model package:

```
msl_data install <model name>
```

To install a model when you have not the right to install it system-wide

To install it in your home (*./macsylib/data*):

```
msl_data install --user <model name>
```

To install it in any directory:

```
msl_data install --target <model dir> <model_name>
```

To know how to cite a model package:

```
msl_data cite <model name>
```

To show the name of the models and the structure of installed model package:

```
msl_data show <model package name>
```

for instance `msl_data show TXSScan`

```
TXSScan
├--archaea
│   └--Archaeal-T4P
├--bacteria
│   └--diderm
│       ├──Flagellum
│       ├──MSH
│       ├──T1SS
│       ├──T2SS
│       ├──T3SS
│       ├──T4aP
│       ├──T4bP
│       ├──T5aSS
│       ├──T5bSS
│       ├──T5cSS
│       ├──T6SSi
│       ├──T6SSii
│       ├──T6SSiii
│       ├──T9SS
│       ├──Tad
│       ├──pT4SSi
│       └--pT4SSst
└--monoderm
    └--ComM
```

TXSScan (1.1.3) : 19 models

To show the model definition:

```
msl_data definition <package or subpackage> model1 [model2, ...]
```

for instance to show model definitions T6SSii and T6SSiii in TXSS+/bacterial subpackage:

```
msl_data definition TXSS+/bacterial T6SSii T6SSiii
```

To show all models definitions in TXSS+/bacterial subpackage:

```
msl_data definition TXSS+/bacterial
```

To create a skeleton for your own model package (to access init subcommand check *modeler installation*):

```
msl_data init --pack-name <MY_PACK_NAME> --maintainer <"mantainer name"> --email
↳<maintainer email> --authors <"author1, author2, ..">
```

above `msl_data` with required options. Below I add option but recommended options.

```
msl_data init --pack-name <MY_PACK_NAME> --maintainer <mantainer name> --email  
↪<maintainer email> --authors <"author1, author2, .."> \  
--license cc-by-nc-sa --holders <"the copyright holders"> --desc <"one line package_  
↪description">
```

To list all `msl_data` subcommands:

```
msl_data --help
```

To list all available options for a subcommand:

```
msl_data <subcommand> --help
```

For models not stored in *macsy-models* the commands *available*, *search*, *installation* from remote or *upgrade* from remote are **NOT** available.

For models **NOT** stored in *macsy-models*, you have to manage them semi-manually. Download the archive (do not unarchive it), then use `msl_data` to install the archive.

MacSyLib Quick Start

1. We recommend to install MacSyLib using *pip* in a virtual environment (for further details see *Installation*).

```
python3 -m venv MacSyLib  
cd MacSyLib  
source bin/activate  
pip install macsylib
```

Warning

hmmsearch from the HMMER package (<http://hmmer.org/>) must be installed.

2. Prepare your data. You need a file containing all protein sequences of your genome of interest in a FASTA file (for further details see *Input dataset*). In the best case scenario, they would be ordered as the corresponding genes are ordered along the replicons.
3. You need to install, or make available to MacSyLib the models to search in your input genome data. Please refer to *Macromolecular models* to create your own package of models. Otherwise, *macsy-models* contributed by the community are available here: <https://github.com/macsy-models> and can be retrieved and installed using the *msl_data* command, installed as part of the MacSyLib suite.
4. Install the *macsy-models* of interest from the *Macsy Models repository*:

```
msl_data available  
msl_data install some-public-models
```
5. Use MacSyLib *How to use MacSyLib ?*

Input and Options of MacSyLib

Input dataset

The input dataset must be a set of protein sequences in **Fasta format** (see http://en.wikipedia.org/wiki/FASTA_format). (The fasta file can be compressed in *gzip* format see note below)

The *base section* in the configuration file (see *Configuration file*) can be used to specify the **path** and the **type of dataset** to deal with, as well as the *sequence_db* and *db_type* config options respectively, described in the *Input options*.

Four types of protein datasets are supported:

- *unordered* : a set of sequences corresponding to a complete genome (*e.g.* an unassembled complete genome)
- *ordered_replicon* : a set of sequences corresponding to an ordered complete replicon (*e.g.* an assembled complete genome)
- *gembase* : a set of multiple ordered replicons, which format follows the convention described in *Gembase format*.

For “ordered” (“ordered_replicon” or “gembase”) datasets only, MacSyLib can take into account the **shape of the genome**: “linear”, or “circular” for detection. The default is set to “circular”.

This can be set with the *replicon_topology* parameter from *Input options* or in the configuration in the *base section*.

With the “gembase” format, it is possible to specify a topology per replicon with a topology file (see *Gembase format* and *Topology files*).

Note

MSL can also read .gz compressed files; it will uncompress them on the fly. The compressed files must end with the .gz extension. For the *hmmsearch* step You need to have *gunzip* installed on your system for this to work.

Config object

Config object take 2 parameters:

1. some default values stored in MacSyDefault (these to class are located in config module)
2. and a set of parameters store in an `argparse.Namespace` object

for instance to set up macsylib

```
from argparse import Namespace
import macsylib.config

defaults = macsylib.config.MacsyDefaults()
settings = Namespace(
    db_type='ordered_replicon',
    sequence_db='test.fasta',
    models=['TXSScan', 'all'], # this model must be installed with msl_data scripts
    worker=4,
)
```

The config options available are:

Input options

models

The models to search. A list, where the first element must be the name of family models, followed by the name of the models. If the name ‘all’ is in the list all models from the family will be searched.

- ['TXSScan', 'all'] to search all models in TXSScan.

- ['TXSScan/bacteria', 'all'] or ['TXSScan', 'bacteria/all'] to search all models in bacteria.
- ['TXSScan/bacteria/diderm', 'T4aP', 'T4bP'] to search only 'T4ap' and 'T4bP' models.

sequence_db

Path to the sequence dataset in fasta format.

db_type

The type of dataset to deal with.

- “*unordered*” corresponds to a non-assembled genome,
- “*ordered_replicon*” to an assembled genome, where sequence appear in the file as the same order as in the genome. (be careful the *GCF* files from the *NCBI*, most of the time, do not follow the order of the associated *gff*)
- and “*gembase*” to a set of replicons see [Gembase format](#).

replicon_topology

circular or *linear* The topology of the replicons (this option is meaningful only if the db_type is 'ordered_replicon' or 'gembase')

topology_file

Topology file path. The topology file allows to specify a topology (linear or circular) for each replicon (this option is meaningful only if the db_type is 'ordered_replicon' or 'gembase'). A topology file is a tabular file with two columns:

the 1st is the replicon name, and the 2nd the corresponding topology:

RepliconA linear

idx

True / False Forces to build the indexes for the sequence dataset even if they were previously computed and present at the dataset location. (default: False)

Systems detection options

inter_gene_max_space

Co-localization criterion: maximum number of components non-matched by a profile allowed between two matched components for them to be considered contiguous. Option only meaningful for 'ordered' datasets. This option must be provide as list of tuple. The tuple first value must match to a model, the second to a number of components.

```
inter_gene_max_space=[('TXSScan/bacteria/diderm/T2SS', 12), ('TXSScan/bacteria/diderm/Flagellum', 14)],
```

min_mandatory_genes_required

The minimal number of mandatory genes required for model assessment. This option must be provide as list of tuple. The first value must correspond to a model fully qualified name, the second value to an integer.

min_genes_required

The minimal number of genes required for model assessment (includes both 'mandatory' and 'accessory' components). The first value must correspond to a model fully qualified name, the second value to an integer.

```
min_genes_required=[('TXSScan/bacteria/diderm/T2SS', 5), ('TXSScan/bacteria/diderm/Flagellum', 4)],
```

max-nb-genes

The maximal number of genes to consider a system as full. The first value must correspond to a model name, the second value to an integer.

multi-loci

Specifies if the system can be detected as a ‘scattered’ system. The models are specified as a comma separated list of fully qualified name ['TXSScan/bacteria/diderm/T2SS', 'TXSScan/bacteria/diderm/Flagellum']

Options for Hmmer execution and hits filtering:**hmmer**

Path to the hmmsearch program. If it is not specify rely on the PATH. (default: hmmsearch)

e_value_search

Maximal e-value for hits to be reported during hmmsearch search. By default MF set per profile threshold for hmmsearch run (cut_ga option) for profiles containing the GA bit score threshold. If a profile does not contains the GA bit score the e_value_search (-E in hmmsearch) is applied to this profile. To applied the e_value_search to all profiles use the no_cut_ga option. (default: 0.1)

no_cut_ga

By default the MSL try to applied a threshold per profile by using the hmmer -cut-ga option. This is possible only if the GA bit score is present in the profile otherwise MSL switch to use the e_value_search (-E in hmmsearch). If this option is set the e_value_search option is used for all profiles regardless the presence of the a GA bit score in the profiles. (default: False)

cut_ga

By default the MSL try to applied a threshold per profile by using the hmmer cut_ga option. This is possible only if the GA bit score is present in the profile otherwise MSF switch to use the -e-value-search (-E in hmmsearch). But the modeler can override this default behavior to do not use cut_ga but e_value_search instead (-E in hmmsearch). The user can reestablish the general MSL behavior, be sure the profiles contain the GA bit score. (default: True)

i_value_sel

Maximal independent e-value for Hmmer hits to be selected for system detection. (default:0.001)

overage_profile

Minimal profile coverage required in the hit alignment to allow the hit selection for system detection. (default: 0.5)

Options for clusters and systems’ scoring:**mandatory_weight**

the weight (score) of a mandatory component when scoring clusters (default:1.0)

accessory_weight

the weight (score) of an accessory component when scoring clusters (default:0.5)

exchangeable_weight

the weight modifier for the score of a component that is exchangeable (default:0.8)

redundancy_penalty

the weight modifier for the score of a component that is already present in another cluster

loner_multi_system_weight

the weight modifier for the score of a component that is *loner* and *multi-system* at the same time (default:0.7)

Path options:**models_dir**

specify the path to the models if the models are not installed in the canonical place. It gather definitions (xml files) and hmm profiles in a specific structure. A directory with the name of the model with at least two directories

profiles” which contains all hmm profile for gene describe in definitions and models” which contains either xml file of definitions or subdirectories to organize the model in subsystems.

out_dir

Path to the directory where to store results. if out-dir is specified res-search-dir will be ignored. You have to create it.

force

force to run even the out dir already exists and is not empty. Use this option with caution, MSF will erase everything in out dir before to run. (default= False)

index_dir

Specifies the path to a directory to store/read the sequence index when the sequence_db dir is not writable.

res_search_suffix

The suffix to give to Hmmer raw output files. (default: .search_hmm.out)

res_extract_suffix

The suffix to give to filtered hits output files. (default: .res_hmm_extract)

profile_suffix

The suffix of profile files. For each ‘Gene’ element, the corresponding profile is searched in the ‘profile_dir’, in a file which name is based on the Gene name + the profile suffix. For instance, if the Gene is named ‘gspG’ and the suffix is ‘.hmm3’, then the profile should be placed at the specified location and be named ‘gspG.hmm3’ (default: .hmm)

General options:**worker**

Number of workers to be used by MacSyLib. In the case the user wants to run MacSyFLib in a multi-thread mode. (0 mean all threads available will be used (compliant with use of cluster scheduler)). (default: 1)

verbosity

Increases the verbosity level. There are 4 levels: Error messages (default), Warning (-v), Info (-vv) and Debug.(-vvv)

mute

mute the log on stdout. (continue to log on macsylib.log) (default: False)

cfg_file

Path to a MacSyLib configuration file to be used.

previous_run

Path to a previous MacSyLib run directory. It allows to skip the Hmmer search step on same dataset, as it uses previous run results and thus parameters regarding Hmmer detection. The configuration file from this previous run will be used. Conflict with options: *config*, *sequence_db*, *profile_suffix*, *res_extract_suffix*, *e_value_res*, *db_type*, *hmm*

timeout

In some case msf can take a long time to find the best solution (in ‘gembase’ and ‘ordered_replicon mode’). The timeout is per replicon. If this step reach the timeout, the replicon is skipped (for gembase mode the analyse of other replicons continue). NUMBER[SUFFIX] NUMBER seconds. SUFFIX may be ‘s’ for seconds (the default), ‘m’ for minutes, ‘h’ for hours or ‘d’ for days for instance 1h2m3s means 1 hour 2 min 3 sec. NUMBER must be an integer.

Configuration file

Options to run MacSyLib can be specified in a configuration file.

The *Config object* handles all configuration options for MacSyLib. There kind of locations where to put configuration file:

1. System wide configuration (this configuration is used for all macsylib run)
 - `/etc/macsylib/macsylib.conf`
 - or in `${VIRTUAL_ENV}/etc/macsylib.conf` if you installed macsylib in a virtualenv
 - the file pointed by environment variable `MACSY_HOME`
2. User wide configuration (this configuration is used for all run for a user)
 - `~/.macsylib/macsylib.conf`
3. Project configuration
 - `macsylib.conf` in the current directory
 - with command line option `-cfg-file`

Note

The precedence rules from the least to the most important priority are:

System wide configuration < user wide configuration < project configuration < command line option

This means that command-line options will always bypass those from the configuration files. In the same flavor, options altering the definition of systems found in the command-line or the configuration file will always overwhelm values from systems' *XML definition files*.

The configuration files must follow the Python “ini” file syntax. The *Config object* provides some default values and performs some validations of the values.

In MacSyLib, six sections are defined and stored by default in the configuration file:

- **base** : all information related to the protein dataset under study
 - `sequence_db` : the path to the dataset in Fasta format (*no default value*)
 - `db_type` : the type of dataset to handle, four types are supported:
 - * `unordered` : a set of sequences corresponding to a complete replicon (*e.g.* an unassembled complete genome)
 - * `ordered_replicon` : a set of sequences corresponding to a complete replicon ordered (*e.g.* an assembled complete genome)
 - * `gembase` : a set of multiple ordered replicons.*(no default value)*
 - `replicon_topology` : the topology of the replicon under study. Two topologies are supported: ‘linear’ and ‘circular’ (*default* = ‘circular’). This option will be ignored if the dataset type is not ordered (*i.e.* “unordered_replicon” or “unordered”).
- **models** * list of models to search in replicon
- **models_opt**

- *inter_gene_max_space* = list of models' fully qualified names and integer separated by spaces (see example below). These values will supersede the values found in the model definition file.
- *min_mandatory_genes_required* = list of models' fully qualified name and integer separated by spaces. These values will supersede the values found in the model definition file.
- *min_genes_required* = list of models' fully qualified name and integer separated by spaces. These values will supersede the values found in the model definition file.
- *max_nb_genes* = list of models' fully qualified names and integer separated by spaces. These values will supersede the values found in the model definition file.

- **hmmer**

- *hmmer_exe* (default= *hmmsearch*)
- *e_value_res* = (default= 1)
- *i_evalue_sel* = (default= 0.5)
- *coverage_profile* = (default= 0.5)

- **score_opt**

- *mandatory_weight* (default= 1.0)
- *accessory_weight* (default= 0.5)
- *exchangeable_weight* (default= 0.8)
- *redundancy_penalty* (default= 1.5)
- *out_of_cluster* (default= 0.7)

- **directories**

- *res_search_dir* = (default= *./datatest/res_search*)
- *res_search_suffix* = (default= *.search_hmm.out*)
- *system_models_dir* = (default= *./models*)
- *res_extract_suffix* = (default= *.res_hmm_extract*)
- *index_dir* = (default= beside the *sequence_db*)

- **general**

- ***log_level*: (default= *debug*) This corresponds to an integer code:**

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

- *log_file* = (default = *macsylib.log* in directory of the run)

Example of a configuration file

```

[base]
prefix = /path/to/macsylib/home/
file = %(prefix)s/data/base/prru_psae.001.c01.fasta
db_type = gembase
replicon_topology = circular

[models]
models_1 = TFF-SF_final all

[models_opt]
inter_gene_max_space = TXSS/T2SS 22 TXSS/Flagellum 44
min_mandatory_genes_required = TXSS/T2SS 6 TXSS/Flagellum 4
min_genes_required = TXSS/T2SS 8 TXSS/Flagellum 4
max_nb_genes = TXSS/T2SS 12 TXSS/Flagellum 8

[hmmer]
hmmer = hmmsearch
e_value_res = 1
i_evalue_sel = 0.5
coverage_profile = 0.5

[score_opt]
mandatory_weight = 1.0
accessory_weight = 0.5
exchangeable_weight = 0.8
redundancy_penalty = 1.5
loner_multi_system_weight = 0.7

[directories]
prefix = /path/to/macsylib/home/
data_dir = %(prefix)s/data/
res_search_dir = %(prefix)s/dataset/res_search/
res_search_suffix = .raw_hmm
system_models_dir = %(data_dir)/data/models, ~/.macsylib/data
profile_suffix = .fasta-aln.hmm
res_extract_suffix = .res_hmm
index_dir = path/where/I/store/my_indexes

[general]
log_level = debug
worker = 4

```

Note

After a run, the corresponding configuration file (“macsylib.conf”) is generated as a (re-usable) output file that stores every options used in the run. It is stored in the results’ directory (see *the output section*).

Warning

The configuration variable *models_dir* cannot be set in general configuration file. *models_dir`* can be

set only in configuration under user control. `\$(HOME)/.macsylib/macsylib.conf < macsylib.conf < "command-line" options` *models_dir* is a single path to a directory where masyfinder can find models.

But the *system_models_dir* can be set in general configuration file

- /etc/macsylib/macsylib.conf
- or \${VIRTUAL_ENV}/etc/macsylib/macsylib.conf
- or anywhere point by \$MACSY_CONF environment variable

system_models_dir manage a list of locations where macsylib can find models. The order of locations is important, it reflects the precedence rule (The models found in last location superseed models found in previous location). By default look for following directories: /share/macsylib/models, or /usr/share/macsylib/models and \$HOME/.macsylib/models and *system_models_dir* uses these directories if they exists.

In-house input files

Gembase format

In order to allow the users to run MacSyLib on **several genomes at once**, we propose to adopt the following convention to fulfill the requirements for the “gembase db_type”.

It consists in providing for each protein, both the replicon name and a protein identifier separated by a “_” in the first field of fasta headers. “_” are accepted in the replicon name, but not in the protein identifier. Hence, the last “_” is the separator between the replicon name and the protein identifier. As such, MacSyLib will be able to treat each replicon separately to assess macromolecular systems’ presence.

For instance:

```
>PlasmidA_0001 YP_003225072.1 | putative stcE protein
MKLKYLSMILASLAMGAFAATAADNNSAIYFNTTQPVNDLQGGLAAEVK
FAQSQILSAHPKEGESQQHLTSLRKSLLLVRLVKADDKTPVQVEARDAND
KILGTLTSPSSLPDTPVYHLDGVPADGIDFTPQNGTKKIINTVAEVNKL
SDASGSSIKSYLANALVEIQTANGRWIRDMYLPQGALEGKMVRVFSYA
GYNSTVFYGD RKVTL SVGNTLLFKYVNGQWFRSGELENNRIAYAQHTWSA
ELPAHWIVPGLNLVIKQGNLSGSLNDINVGAPGELLHTIDIGMLTTPRG
RFDFAKDKEAHREYFQTIPVSRMIVNNYAPLHLKEVMLPTGTLLTDADPG
>PlasmidA_0002 YP_003225073.1 | type II secretion protein EtpC
MLFFLSSRRDRNLFIKDIALKMLTPNWVLCVILLIAGYQLVSVIRHFWLT
PATASDLSHVSSETAVTDEHTEENFVFTLFGTASPPLSEGKVQKTTSS
LSDDLSSGGDLVDVRGILYSSVTEHSVAIFAHNNRQFSLGIGEKVPGYDAT
ISAI FSDHIVINYQGNASLPLRYDNP AKRNAQDDNNLIVGPVTTQANFR
VKNIFDIMSLSPVTVNNTLSGYRLSPGKASSLFYNAGLHDNDLAVLLNGS
ELRDRQAKQIMKQLTELKEIKITVERDGQLYDAFIAVGEN
....
>ChromosomeA_0001 YP_003573410.1 | adhesin-like protein
MKKLFLFAALLMTGFAYSCEDVVDNPAQDPAQSWNYSVSVKFADFDFNG
AVDENSVPYTYKAPTTLYVLNEENTLMGTITTTDAAPAIGDYGTAGTLTG
SIGNNLIITTKIGNDLTKQDGTLSAIENGIVQTAEPVPIKIYNANSGLT
TASAKMDNTAAIAYTSLGYIKGGDKILFVEGNQTFEWTVNEEFDPYTSTD
LYIALPMNTDPETEYTISSDSKDG YTRGGTFKLADYPTLAAGKVSNIYIGG
IPFIQTGVDLTKWDAYMRTPNNTWYMNINNGWPATFSQEVEDGKSFIV
TQSGPTLDSLNVVVGVTGKEVNVTLNNIRLGKDRSINIGDKHGWVEYDG
THDIYGWGAKANVTLIGENECETLYIQCPATKKGEGTLNKNLSIDSYGS
>ChromosomeA_0020 YP_003573411.1 | hypothetical protein
```

(continues on next page)

(continued from previous page)

```

MKRIVLITLVSILTTFFQAIAQVANGFYRVQNNASSRYITLRDNAVGTVDY
SSTNVDSLNIWTSQFQDKVSNPASIIYVEQHDSDYDLKVQGTGIYAITG
GRTYLELRPKDSGYILAVTYNGMEGRLYDSEEDVDGEGYVKRSGNSAYQY
WSFIPVDTENNYIGLQPTVQVGDNYYGTLYASYPFKAASSGIKFYYVDAI

```

```

>NC_001548_0015 YP_003225080.1 | type II secretion protein EtpJ (translation)
MSQQRVKGFLLLEMLLALAVFAALSISAFQVLQSGIRAHLSQDKVRRRLA
ELQRGGSQIERDLMQMIPIRHSRGSEGLLLAAPHLLKSDDWGISFTRNSWL
NPAGMLPRPELQWVGYRLRQKLERLSYFYVDHPSGIAPDVRVVGEGVHA
FRLRFFVNGTWQARWDSTSILPQAVEVTLVMDDFAELTRLFLVSKETAE

```

This input file contains 3 replicons: PlasmidA (which 2 first protein identifiers are 0001 and 0002), ChromosomeA (which 2 first protein identifiers are 0001 and 0020) and NC_001548 (which first protein identifier is 0015). MacSyLib search results will thus be reported for each of these three replicons.

Warning

This *gembase* format is old and not compliant with the *gembase* format produced by [PanACoTA](#). The support of the new *gembase* format is in the road map.

Topology files

To be able to attribute a topology per replicon/genome when using the *Gembase* format, we propose the user to build a “topology file” in the form of a tabular file with two columns separated by a “:”. The 1st column is the replicon name, and the 2nd the corresponding topology. Comments can be written after a “#”.

For example:

```

# comment line
PlasmidA : circular
ChromosomeA : linear
ChromosomeB : circular

```

Note

A topology file can be specified on the command-line with the `--topology-file` parameter.

Output format

MacSyLib allow to produce different types of output files.

There are three types of output files:

1. The main output files for the systems’ search. They differ with the search mode (*ordered* or *unordered*).
2. The *HMMER output files* (search of each systems’ components), located in the *hmm_results* folder.
3. The internal *configuration and log files*.

Note

Each tabular output file contains a header line describing each column in the output.

Output files for the “ordered replicon(s)” search modes

These output files can be generated when MacSyLib search proceeds on a set of proteins that are deemed to follow the order of their genes on replicons. This corresponds to the two search modes *gembase* and *ordered_replicon*.

Systems detection results

Different types of output files can be generated, human-readable files “.txt”, and tabulated files “.tsv”. For the latter, headers are provided with the content of the lines in the file.

- *best_solution.tsv* - This file contains the **best solution found by MacSyLib** in terms of systems detected, under the form of a per-component, tabulated report file. A **solution** consists in a set of compatible systems (no components’ overlap allowed). If multiple solutions showed a maximal score, a *ranking* is established.

To see potential other best solutions (in case several obtained the same highest score), see file *all_best_solutions.tsv*.

To see all possible, candidate systems without further processing, see files *all_systems.txt* and *all_systems.tsv*.

The *best_solution.tsv* file is the most similar to former V1 file *macsylib.report*.

- *best_solution_loners.tsv* and *best_solution_multisystems.tsv* report hits which have been identified as loners or multi-systems which means that the corresponding gene is tagged as a ‘loner’ or ‘multi-system’ in the model definition and the hit is not located in a cluster.
- *best_solution_summary.tsv* is a summary of the *best_solution.tsv* file, containing the number of systems detected in each replicon analysed.
- *all_systems.txt* - This file describes the search process of all possible candidate systems given the definitions in systems’ models - without processing of the potential overlaps between candidate systems. This set of possible candidate systems are also given under the form of a tabulated file in *all_systems.tsv*.
- *rejected_candidates.tsv* and *rejected_candidates.txt* - This file lists candidate clusters (or a combination of clusters) components that were rejected by MacSyLib during the search process, and were thus not assigned to a candidate system. This set of clusters are also given under the form of tabulated file *rejected_candidates.tsv*.
- *all_best_solutions.tsv* - This file contains all possible best solutions under the form of a per-component, tabulated report file. To retrieve a single best solution as proposed by MacSyLib, see file *best_solution.tsv*.
- *all_systems.tsv* - This file contains all possible candidate systems given the definitions - without processing of the potential overlaps between candidate systems, under the form of a per-component, tabulated report file. It corresponds to the tabulated version of the *all_systems.txt* file.

all_systems.txt

The file starts with some comments:

- the version of MacSyLib used
- the name of model package and version used
- the command line used to produce this file

Then for each replicon, the systems detected are listed along with their description:

- **system_id** - the unique identifier of a system
- **model** - the model assigned to this system
- **replicon** - the name of the replicon harbouring the system
- **clusters** - the clusters composition of this system
 - each clusters is a list of tuple

- each tuple is composed of:
 - * the name of the matching gene(s) in the replicon
 - * the name of the corresponding gene profile(s)
 - * the position of the corresponding sequence(s) along the replicon
- **occurrence** - the average number of occurrences of each components of the system (as a potential proxy to estimate whether there's the genetic potential for multiple systems in one)
- **wholeness** - the percentage of the model's components that were found in this system
- **loci nb** - the number of different loci constituting this system
- **score** - the score of the system. See [here](#) for more details
- **systems components** - the number of occurrences of each model components in parenthesis the name of the matching profile in square brackets the name of other putative systems that would involve this gene

Here is an example of the *all_systems.txt* file:

```
# macsylib 20200217.dev
# models: TFF-SF_final-0.1
# macsylib --sequence-db DATA_TEST/sequences.prt --db-type=gembase --models-dir data/
↳models/ --models TFF-SF_final all -w 4
# Systems found:

system id = VICH001.B.00001.C001_MSH_1
model = TFF-SF_final/MSH
replicon = VICH001.B.00001.C001
clusters = [('VICH001.B.00001.C001_00406', 'MSH_mshI', 366), ('VICH001.B.00001.C001_00407
↳', 'MSH_mshJ', 367), ('VICH001.B.00001.C001_00408', 'MSH_mshK', 368), ('VICH001.B.
↳00001.C001_00409', '
MSH_mshL', 369), ('VICH001.B.00001.C001_00410', 'MSH_mshM', 370), ('VICH001.B.00001.C001_
↳00411', 'MSH_mshN', 371), ('VICH001.B.00001.C001_00412', 'MSH_mshE', 372), ('VICH001.B.
↳00001.C001_0041
3', 'MSH_mshG', 373), ('VICH001.B.00001.C001_00414', 'MSH_mshF', 374), ('VICH001.B.00001.
↳C001_00415', 'MSH_mshB', 375), ('VICH001.B.00001.C001_00416', 'MSH_mshA', 376), (
↳'VICH001.B.00001.C001
_00417', 'MSH_mshC', 377), ('VICH001.B.00001.C001_00418', 'MSH_mshD', 378), ('VICH001.B.
↳00001.C001_00419', 'MSH_mshO', 379), ('VICH001.B.00001.C001_00420', 'MSH_mshP', 380), (
↳'VICH001.B.00001
.C001_00421', 'MSH_mshQ', 381)]
occ = 1
wholeness = 0.941
loci nb = 1
score = 10.500

mandatory genes:
  - MSH_mshA: 1 (MSH_mshA)
  - MSH_mshE: 1 (MSH_mshE)
  - MSH_mshG: 1 (MSH_mshG)
  - MSH_mshL: 1 (MSH_mshL)
  - MSH_mshM: 1 (MSH_mshM)

accessory genes:
  - MSH_mshB: 1 (MSH_mshB)
```

(continues on next page)

(continued from previous page)

```

- MSH_mshC: 1 (MSH_mshC)
- MSH_mshD: 1 (MSH_mshD)
- MSH_mshF: 1 (MSH_mshF)
- MSH_mshI: 1 (MSH_mshI)
- MSH_mshI2: 0 ()
- MSH_mshJ: 1 (MSH_mshJ)
- MSH_mshK: 1 (MSH_mshK)
- MSH_mshN: 1 (MSH_mshN)
- MSH_mshO: 1 (MSH_mshO)
- MSH_mshQ: 1 (MSH_mshQ)
- MSH_mshP: 1 (MSH_mshP)

```

neutral genes:

```

=====
system id = VICH001.B.00001.C001_T4P_14
model = TFF-SF_final/T4P
replicon = VICH001.B.00001.C001
clusters = [('VICH001.B.00001.C001_00476', 'T4P_pilT', 427), ('VICH001.B.00001.C001_00477
↳ ', 'T4P_pilU', 428)], [('VICH001.B.00001.C001_00847', 'T4P_pilO', 778), ('VICH001.B.
↳ 00001.C001_00850',
↳ 'T4P_pilE', 781), ('VICH001.B.00001.C001_00851', 'T4P_fimT', 782), ('VICH001.B.00001.
↳ C001_00852', 'T4P_pilW', 783), ('VICH001.B.00001.C001_00853', 'T4P_pilX', 784), (
↳ 'VICH001.B.00001.C001_00
854', 'T4P_pilV', 785)], [('VICH001.B.00001.C001_02305', 'T4P_pilA', 2202), ('VICH001.B.
↳ 00001.C001_02306', 'T4P_pilB', 2203), ('VICH001.B.00001.C001_02307', 'T4P_pilC', 2204),
↳ ('VICH001.B.000
01.C001_02308', 'T4P_pilD', 2205)], [('VICH001.B.00001.C001_02502', 'MSH_mshM', 2391), (
↳ 'VICH001.B.00001.C001_02505', 'T4P_pilQ', 2394), ('VICH001.B.00001.C001_02506', 'T4P_
↳ pilP', 2395), ('VI
CH001.B.00001.C001_02507', 'T4P_pilO', 2396), ('VICH001.B.00001.C001_02508', 'T4P_pilN',
↳ 2397), ('VICH001.B.00001.C001_02509', 'T4P_pilM', 2398)]
occ = 1
wholeness = 0.944
loci nb = 4
score = 12.000

```

mandatory genes:

```

- T4P_pilE: 1 (T4P_pilE)
- T4P_pilB: 1 (T4P_pilB)
- T4P_pilC: 1 (T4P_pilC)
- T4P_pilO: 2 (T4P_pilO, T4P_pilO)
- T4P_pilQ: 1 (T4P_pilQ)
- T4P_pilN: 1 (T4P_pilN)
- T4P_pilT: 1 (T4P_pilT)
- T4P_pilD: 1 (T4P_pilD [VICH001.B.00001.C001_T2SS_4])

```

accessory genes:

```

- T4P_pilA: 1 (T4P_pilA)
- T4P_pilV: 1 (T4P_pilV)
- T4P_pilY: 0 ()
- T4P_pilW: 1 (T4P_pilW)

```

(continues on next page)

(continued from previous page)

```

- T4P_pilX: 1 (T4P_pilX)
- T4P_fimT: 1 (T4P_fimT)
- T4P_pilM: 1 (T4P_pilM)
- T4P_pilP: 1 (T4P_pilP)
- T4P_pilU: 1 (T4P_pilU)
- MSH_mshM: 1 (MSH_mshM)

```

neutral genes:

all_systems.tsv

This corresponds to the tabulated version of the systems listed in *all_systems.txt*. Each line corresponds to a “hit” that has been assigned to a detected system. It includes:

- **replicon** - the name of the replicon it belongs to
- **hit_id** - the unique identifier of the hit
- **gene_name** - the name of the component identified by the hit
- **hit_pos** - the position of the sequence in the replicon
- **model_fqn** - the model fully-qualified name
- **sys_id** - the unique identifier attributed to the detected system
- **sys_loci** - the number of loci
- **locus_num** - the number of the locus where is located this gene. Loners gene have a negative locus_num
- **sys_wholeness** - the wholeness of the system
- **sys_score** - the system score
- **sys_occ** - the estimated number of system occurrences that could be potentially “filled” with this system’s occurrence, based on the average number of each component found. A proxy for the genetic potential ton encode several systems from the set of components found in this one occurrence.
- **hit_gene_ref** - the gene in the model whose this hit plays the role of
- **hit_status** - the status of the component in the assigned system’s definition
- **hit_seq_len** - the length of the protein sequence matched by this hit
- **hit_i_eval** - Hmmer statistics, the independent-evalue
- **hit_score** - Hmmer score
- **hit_profile_cov** - the percentage of the profile covered by the alignment with the sequence
- **hit_seq_cov** - the percentage of the sequence covered by the alignment with the profile
- **hit_begin_match** - the position in the sequence where the profile match begins
- **hit_end_match** - the position in the sequence where the profile match ends
- **counterpart** - the hit id of some other hit which are equivalent. Only loners and multi-systems hits have counterparts
- **used_in** - whether the hit could be used in another system’s occurrence

This file can be easily parsed using the Python [pandas](#) library.

```
import pandas as pd
```

```
systems = pd.read_csv("path/to/systems.tsv", sep='\t', comment='#')
```

Note

Each system reported is separated from the others with a blank line to ease human reading. These lines are ignored during the parsing with pandas.

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
→type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH_
→T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
```

replicon	hit_id	gene_name	hit_pos	model_fqn	sys_	
→id	sys_loci	locus_num	sys_wholeness	sys_score	sys_	
→occ	hit_gene_ref	hit_status	hit_seq_len	hit_i_		
→eval	hit_score	hit_profile_cov	hit_seq_cov	hit_begin_		
→match	hit_end_match	counterpart	used_in			
GCF_000005845	GCF_000005845_000970		T4P_pilC	97	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	1	0.556	7.	
→260	1	T4P_pilC	mandatory	400	2.2e-105	353.
→100	0.991	0.830	62	393		
GCF_000005845	GCF_000005845_000980		T4P_pilB	98	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	1	0.556	7.	
→260	1	T4P_pilB	mandatory	461	8.9e-152	506.
→100	0.948	0.850	62	453		
GCF_000005845	GCF_000005845_000990		T4P_pilA	99	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	1	0.556	7.	
→260	1	T4P_pilA	accessory	146	1.1e-19	71.
→200	0.859	0.473	5	73		
GCF_000005845	GCF_000005845_025680		T4P_pilW	2568	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	2	0.556	7.	
→260	1	T4P_pilW	accessory	187	3.3e-08	34.
→500	0.625	0.401	6	80		
GCF_000005845	GCF_000005845_025690		T4P_fimT	2569	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	2	0.556	7.	
→260	1	T4P_fimT	accessory	156	2.5e-06	28.
→500	0.939	0.397	5	66		
GCF_000005845	GCF_000005845_030590		T4P_pilQ	3059	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	3	0.556	7.	
→260	1	T4P_pilQ	mandatory	412	5.9e-51	173.
→100	0.919	0.408	244	411		
GCF_000005845	GCF_000005845_030620		T4P_pilN	3062	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	3	0.556	7.	
→260	1	T4P_pilN	mandatory	179	3.8e-09	37.
→500	0.986	0.765	5	141		
GCF_000005845	GCF_000005845_030630		T4P_pilM	3063	TFF-SF/	
→T4P	GCF_000005845_T4P_14	3	3	0.556	7.	
→260	1	T4P_pilM	accessory	259	1.1e-09	39.

(continues on next page)

(continued from previous page)

```

→300      0.988      0.598      8      162
GCF_000005845      GCF_000005845_026740      T4P_pilT      2674      TFF-SF/
→T4P      GCF_000005845_T4P_14      3      -1      0.556      7.
→260      1      T4P_pilT      mandatory      326      1.1e-117      393.
→600      0.944      0.979      3      321
GCF_000005845      GCF_000005845_026930      T2SS_gsp0      2693      TFF-SF/
→T4P      GCF_000005845_T4P_14      3      -2      0.556      7.
→260      1      T4P_pilD      mandatory      269      1.3e-87      294.
→000      1.000      0.859      30      260      GCF_000005845_
→030080      GCF_000005845_T2SS_2

```

Note

If a loner component is not clustered with other genes, it will not be considered as part of a locus. Thus, its locus number will be a negative value (numbered from -1) and will not be counted in the variable *sys_loci* (number of loci for a system). See above lines for more details.

```

GCF_000005845      GCF_000005845_026740      T4P_pilT      2674      TFF-SF/T4P      GCF_
→000005845_T4P_25      3      -1      0.556      7.800
GCF_000005845      GCF_000005845_026930      T2SS_gsp0      2693      TFF-SF/T4P      GCF_
→000005845_T4P_25      3      -2      0.556      7.800

```

best_solution.tsv and all_best_solutions.tsv

Since MacSyLib 2.0, a combinatorial exploration of solutions using sets of systems found is performed. We call best solution, the combination of systems offering the highest score.

The *best_solution.tsv* and *all_best_solutions.tsv* files have the same structure as the file *all_systems.tsv*, except that there is an extra column **sol_id** which is a solution identifier added to the file *all_best_solutions.tsv*. The systems that have the same “sol_id” belong to a same solution.

As the files have the same structure as *all_systems.tsv*, they can also be parsed with pandas as shown above.

For the description of the fields of *best_solution.tsv*, see [above](#) those of the *all_systems.tsv* file.

For the *all_best_solutions.tsv*, each line corresponds to a “hit” that has been assigned to a detected system. It includes:

- **sol_id** - the name of the solution it is part of (**only in *all_best_solutions.tsv* files**)
- **replicon** - the name of the replicon it belongs to
- **hit_id** - the unique identifier of the hit
- **gene_name** - the name of the component identified by the hit
- **hit_pos** - the position of the sequence in the replicon
- **model_fqn** - the model fully-qualified name
- **sys_id** - the unique identifier attributed to the detected system
- **sys_loci** - the number of loci
- **locus_num** - the number of the locus where is located this gene. Loners gene have negative locus_num
- **sys_wholeness** - the wholeness of the system
- **sys_score** - the system score

- **sys_occ** - the estimated number of system occurrences that could be potentially “filled” with this system’s occurrence, based on the average number of each component found. A proxy for the genetic potential to encode several systems from the set of components found in this one occurrence.
- **hit_gene_ref** - the gene in the model whose this hit plays the role of
- **hit_status** - the status of the component in the assigned system’s definition
- **hit_seq_len** - the length of the protein sequence matched by this hit
- **hit_i_eval** - Hmmer statistics, the independent-evalue
- **hit_score** - Hmmer score
- **hit_profile_cov** - the percentage of the profile covered by the alignment with the sequence
- **hit_seq_cov** - the percentage of the sequence covered by the alignment with the profile
- **hit_begin_match** - the position in the sequence where the profile match begins
- **hit_end_match** - the position in the sequence where the profile match ends
- **counterpart** - the hit id of some other hit which are equivalent. Only loners and multi-systems hits have counterparts
- **used_in** - whether the hit could be used in another system’s occurrence

Note

Each system reported is separated from the others with a blank line to ease human reading. These lines are ignored during the parsing with pandas.

Example of *best_solution.tsv* files

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
→type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH_
→T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
replicon      hit_id      gene_name      hit_pos      model_fqn      sys_
→id      sys_loci      locus_num      sys_wholeness      sys_score      sys_
→occ      hit_gene_ref      hit_status      hit_seq_len      hit_i_
→eval      hit_score      hit_profile_cov      hit_seq_cov      hit_begin_
→match      hit_end_match      counterpart      used_in
GCF_000005845      GCF_000005845_000970      T4P_pilC      97      TFF-SF/
→T4P      GCF_000005845_T4P_9      1      1      0.278      3.
→760      1      T4P_pilC      mandatory      400      2.2e-105      353.
→100      0.991      0.830      62      393
GCF_000005845      GCF_000005845_000980      T4P_pilB      98      TFF-SF/
→T4P      GCF_000005845_T4P_9      1      1      0.278      3.
→760      1      T4P_pilB      mandatory      461      8.9e-152      506.
→100      0.948      0.850      62      453
GCF_000005845      GCF_000005845_000990      T4P_pilA      99      TFF-SF/
→T4P      GCF_000005845_T4P_9      1      1      0.278      3.
→760      1      T4P_pilA      accessory      146      1.1e-19      71.
→200      0.859      0.473      5      73
GCF_000005845      GCF_000005845_026740      T4P_pilT      2674      TFF-SF/
```

(continues on next page)

(continued from previous page)

→T4P	GCF_000005845_T4P_9	1	-1	0.278	3.	
→760	1	T4P_pilT	mandatory	326	1.1e-117	393.
→600	0.944	0.979	3	321		
GCF_000005845	GCF_000005845_026930		T2SS_gspO	2693		TFF-SF/
→T4P	GCF_000005845_T4P_9	1	-2	0.278	3.	
→760	1	T4P_pilD	mandatory	269	1.3e-87	294.
→000	1.000	0.859	30	260	GCF_000005845_	
→030080	GCF_000005845_T2SS_2					
GCF_000005845	GCF_000005845_025680		T4P_pilW	2568		TFF-SF/
→T4P	GCF_000005845_T4P_13	2	1	0.389	4.	
→760	1	T4P_pilW	accessory	187	3.3e-08	34.
→500	0.625	0.401	6	80		
GCF_000005845	GCF_000005845_025690		T4P_fimT	2569		TFF-SF/
→T4P	GCF_000005845_T4P_13	2	1	0.389	4.	
→760	1	T4P_fimT	accessory	156	2.5e-06	28.
→500	0.939	0.397	5	66		
GCF_000005845	GCF_000005845_030590		T4P_pilQ	3059		TFF-SF/
→T4P	GCF_000005845_T4P_13	2	2	0.389	4.	
→760	1	T4P_pilQ	mandatory	412	5.9e-51	173.
→100	0.919	0.408	244	411		
GCF_000005845	GCF_000005845_030620		T4P_pilN	3062		TFF-SF/
→T4P	GCF_000005845_T4P_13	2	2	0.389	4.	
→760	1	T4P_pilN	mandatory	179	3.8e-09	37.
→500	0.986	0.765	5	141		

Example of *all_best_solutions.tsv* files

```
# macsylib 20250121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsylib --db-type=gembase -
→-models-dir=tests/data/models/ --models TFF-SF Archaeal-T4P ComM MSH T2SS T4bP T4P_
→Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
sol_id      replicon      hit_id      gene_name      hit_pos      model_
→fqn        sys_id        sys_loci     locus_num      sys_wholeness sys_
→score      sys_occ       hit_gene_ref hit_status      hit_seq_
→len        hit_i_eval    hit_score    hit_profile_cov hit_seq_
→cov        hit_begin_match hit_end_match counterpart     used_in
1           GCF_000005845 GCF_000005845_000970 T4P_pilC       97           TFF-
→SF/T4P      GCF_000005845_T4P_9 1 1 0.278 3.
→760        1            T4P_pilC     mandatory      400          2.2e-105     353.
→100        0.991        0.830        62            393
1           GCF_000005845 GCF_000005845_000980 T4P_pilB       98           TFF-
→SF/T4P      GCF_000005845_T4P_9 1 1 0.278 3.
→760        1            T4P_pilB     mandatory      461          8.9e-152     506.
→100        0.948        0.850        62            453
1           GCF_000005845 GCF_000005845_000990 T4P_pilA       99           TFF-
→SF/T4P      GCF_000005845_T4P_9 1 1 0.278 3.
→760        1            T4P_pilA     accessory      146          1.1e-19      71.
→200        0.859        0.473        5             73
1           GCF_000005845 GCF_000005845_026740 T4P_
```

(continues on next page)

(continued from previous page)

→pilT	2674	TFF-SF/T4P	GCF_000005845_T4P_9	1	-	
→1	0.278	3.760	1	T4P_		
→pilT	mandatory	326	1.1e-117	393.600	0.944	0.
→979	3	321				
1	GCF_000005845	GCF_000005845_026930	T2SS_			
→gsp0	2693	TFF-SF/T4P	GCF_000005845_T4P_9	1	-	
→2	0.278	3.760	1	T4P_		
→pilD	mandatory	269	1.3e-87	294.000	1.000	0.
→859	30	260	GCF_000005845_030080	GCF_000005845_T2SS_2		
1	GCF_000005845	GCF_000005845_025680	T4P_			
→pilW	2568	TFF-SF/T4P	GCF_000005845_T4P_			
→13	2	1	0.389	4.760	1	T4P_
→pilW	accessory	187	3.3e-08	34.500	0.625	0.
→401	6	80				
1	GCF_000005845	GCF_000005845_025690	T4P_			
→fimT	2569	TFF-SF/T4P	GCF_000005845_T4P_			
→13	2	1	0.389	4.760	1	T4P_
→fimT	accessory	156	2.5e-06	28.500	0.939	0.
→397	5	66				
1	GCF_000005845	GCF_000005845_030590	T4P_			
→pilQ	3059	TFF-SF/T4P	GCF_000005845_T4P_			
→13	2	2	0.389	4.760	1	T4P_
→pilQ	mandatory	412	5.9e-51	173.100	0.919	0.
→408	244	411				
1	GCF_000005845	GCF_000005845_030620	T4P_			
→pilN	3062	TFF-SF/T4P	GCF_000005845_T4P_			
→13	2	2	0.389	4.760	1	T4P_
→pilN	mandatory	179	3.8e-09	37.500	0.986	0.
→765	5	141				
1	GCF_000005845	GCF_000005845_030630	T4P_			
→pilM	3063	TFF-SF/T4P	GCF_000005845_T4P_			
→13	2	2	0.389	4.760	1	T4P_
→pilM	accessory	259	1.1e-09	39.300	0.988	0.
→598	8	162				
1	GCF_000005845	GCF_000005845_026740	T4P_			
→pilT	2674	TFF-SF/T4P	GCF_000005845_T4P_13	2	-	
→1	0.389	4.760	1	T4P_		
→pilT	mandatory	326	1.1e-117	393.600	0.944	0.
→979	3	321				
1	GCF_000005845	GCF_000005845_026930	T2SS_			
→gsp0	2693	TFF-SF/T4P	GCF_000005845_T4P_13	2	-	
→2	0.389	4.760	1	T4P_		
→pilD	mandatory	269	1.3e-87	294.000	1.000	0.
→859	30	260	GCF_000005845_030080	GCF_000005845_T2SS_2		
1	GCF_000005845	GCF_000005845_029970	T2SS_			
→gspC	2997	TFF-SF/T2SS	GCF_000005845_T2SS_			
→1	1	1	0.857	9.000	1	T2SS_
→gspC	mandatory	271	2.3e-19	70.400	0.897	0.
→358	47	143				
1	GCF_000005845	GCF_000005845_030050	T2SS_			

(continues on next page)

(continued from previous page)

```

→gspK      3005      TFF-SF/T2SS      GCF_000005845_T2SS_
→1          1          1          0.857      9.000      1          T2SS_
→gspK      accessory      327      1e-16      61.500      1.000      0.
→180       6          64
1          GCF_000005845      GCF_000005845_030060      T2SS_
→gspL      3006      TFF-SF/T2SS      GCF_000005845_T2SS_
→1          1          1          0.857      9.000      1          T2SS_
→gspL      accessory      387      1.5e-37      129.300      1.000      0.
→351       6          141
1          GCF_000005845      GCF_000005845_030070      T2SS_
→gspM      3007      TFF-SF/T2SS      GCF_000005845_T2SS_
→1          1          1          0.857      9.000      1          T2SS_
→gspM      accessory      153      2.8e-29      102.900      0.985      0.
→804       13          135
1          GCF_000005845      GCF_000005845_030080      T2SS_
→gsp0      3008      TFF-SF/T2SS      GCF_000005845_T2SS_
→1          1          1          0.857      9.000      1          T2SS_
→gsp0      mandatory      225      4e-65      220.400      0.978      0.
→840       26          214

# WARNING Loner: there is only 1 occurrence(s) of loner 'T4P_pilT' and 2 potential_
→systems [GCF_000005845_T4P_9, GCF_000005845_T4P_13]

2          GCF_000005845      GCF_000005845_000970      T4P_pilC      97      TFF-
→SF/T4P      GCF_000005845_T4P_11      2          1          0.389      4.
→760       1          T4P_pilC      mandatory      400      2.2e-105      353.
→100       0.991      0.830      62      393

```

Note

If a loner component is not clustered with other genes, it will not be considered as part of a locus. Thus, its locus number will be a negative value (numbered from -1) and will not be counted in the variable *sys_loci* (number of loci for a system). See above lines for more details.

Note

If several systems from same model use a loner (same gene) *msf* check that there is at least one occurrence of this hit for each system. If there are fewer hits than systems occurrence a warning is displayed in *best_solution.tsv* or *all_best_solution.tsv* as comment. So the file can be parsed with pandas without problem.

```

1          GCF_000005845      GCF_000005845_030080      T2SS_gsp0      3008      TFF-SF/T2SS
→ GCF_000005845_T2SS_1      1          1          0.857      9.000      1          T2SS_gsp0
→mandatory      225      4e-65      220.400      0.978      0.840      26      214

# WARNING Loner: there is only 1 occurrence(s) of loner 'T4P_pilT' and 2 potential_
→systems [GCF_000005845_T4P_9, GCF_000005845_T4P_13]

2          GCF_000005845      GCF_000005845_000970      T4P_pilC      97      TFF-SF/T4P
→ GCF_000005845_T4P_11      2          1          0.389      4.760      1          T4P_pilC
→mandatory      400      2.2e-105      353.100      0.991      0.830      62      393

```

Note

In case multiple solutions have the exact same score, a sorting is performed among the best solutions, and the solution ranked 1st is reported in the *best_solution.tsv* and *best_solution.txt* files. The ranking is performed as follow:

1. by the number of systems' components (hits) constituting the solution (most components first)
2. by the number of systems (most systems in first)
3. by the average of systems' wholeness
4. by hits position. This criterion is mostly introduced to produce reproducible results between two runs.

best_solution_summary.tsv

This file is a concise view of which systems have been found in your replicons and how many per replicon. It is based on **best_solution.tsv**. The first two lines are comments that indicate the version of MacSyLib and the command line used to generate the results. Then a table represented by tabulated text to separate columns, with the searched models in columns and the replicons scanned for the models in row.

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsfinder --db-
→type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH-
→T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
```

replicon	TFF-SF/MSH	TFF-SF/T2SS	TFF-SF/T4P	TFF-SF/
→T4bP	TFF-SF/Tad	TFF-SF/Archaeal-T4P	TFF-SF/ComM	
GCF_000005845	0	1	2	0
GCF_000006725	0	1	2	0
GCF_000006745	1	1	2	1
GCF_000006765	0	3	1	0
GCF_000006845	0	0	1	0
GCF_000006905	0	1	0	0
GCF_000006925	0	0	1	0
GCF_000006945	0	0	2	0

as a *tsv* file it can be parsed easily using pandas:

```
import pandas as pd
solution = pd.read_csv('path to best_solution_summary.tsv', sep='\t', comment='#', index_
→col=0)
```

Note

If you want to do the same operation but based on the *all_best_solutions.tsv* file, you can do it with the few lines of pandas below:

```
import pandas as pd

all_best_sol = '<macsylib_results_dir>/all_best_solutions.tsv'

# read data from best_solution file
data = pd.read_csv(all_best_sol, sep='\t', comment='#')
```



```
# remove useless columns
selection = data[['sol_id', 'replicon', 'sys_id', 'model_fqn']]

# keep only one row per replicon, sys_id
dropped = selection.drop_duplicates(subset=['sol_id', 'replicon', 'sys_id'])

# count for each replicon which models have been detected and their
→ occurrences
summary = pd.crosstab(index=[dropped.sol_id, dropped.replicon],
→ columns=dropped['model_fqn'])
```

if you are not fluent in *pandas*, we provide you a tiny script *msf_summary.py* based on few lines above to do the job *msf_summary.py*.

Then you can run the script

```
python msf_summary.py <path_to_all_best_solutions.tsv>
```

below an example of summary of *all_best_solutions.tsv*

sol_id	replicon	TFF-SF/MSH		TFF-SF/T2SS		TFF-SF/
→ T4P	TFF-SF/T4bP	TFF-SF/Tad				
1	GCF_0000005845	0	1	1	0	0
2	GCF_0000006725	0	1	1	0	0
3	GCF_0000006725	0	1	1	0	0
4	GCF_0000006745	1	1	2	1	0
5	GCF_0000006745	1	1	2	1	0
6	GCF_0000006745	1	1	1	1	0
7	GCF_0000006765	0	3	1	0	1
8	GCF_0000006845	0	0	1	0	0
9	GCF_0000006905	0	1	0	0	1
10	GCF_0000006925	0	0	1	0	0
11	GCF_0000006945	0	0	1	0	0

best_solution_loners.tsv

This file give an overview of all hits identified as Loner in the best_solution

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsfinder --db-
→ type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P
→ ComM MSH T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/
→ gembase.fasta -w 12
# Loners found:
```

replicon	model_fqn	function	gene_name	hit_
→ id	hit_pos	hit_status	hit_seq_len	hit_i_
→ eval	hit_score	hit_profile_cov	hit_seq_cov	hit_
→ begin_match	hit_end_match			
GCF_0000005845	TFF-SF/T4P	T4P_pilT	T4P_pilT	GCF_
→ 0000005845_026740	2674	mandatory	326	1.100e-
→ 117	393.600	0.944	0.979	3 321
GCF_0000005845	TFF-SF/T4P	T4P_pilD	T2SS_gspO	GCF_
→ 0000005845_026930	2693	mandatory	269	1.300e-

(continues on next page)

(continued from previous page)

↪87	294.000	1.000	0.859	30	260	
GCF_000005845	TFF-SF/T4P	T4P_pild	T2SS_gsp0	GCF_		
↪000005845_030080	3008	mandatory	225	4.000e-		
↪65	220.400	0.978	0.840	26	214	
GCF_000006725	TFF-SF/T4P	T4P_pild	T4P_pild	GCF_		
↪000006725_000270	4269	mandatory	344	1.800e-		
↪172	573.700	0.994	0.985	2	340	
GCF_000006725	TFF-SF/T4P	T4P_pila	T4P_pila	GCF_		
↪000006725_003680	4610	accessory	187	9.000e-		
↪10	39.500	0.667	0.278	6	57	
GCF_000006725	TFF-SF/T2SS	T2SS_gsp0	T4P_pild	GCF_		
↪000006725_014570	5699	mandatory	287	7.400e-		
↪77	258.600	1.000	0.836	28	267	
GCF_000006725	TFF-SF/T2SS	T2SS_gspE	T2SS_gspE	GCF_		
↪000006725_018700	6112	mandatory	566	1.800e-		
↪171	571.000	0.936	0.701	165	561	
GCF_000006725	TFF-SF/T4P	T4P_pila	T4P_pila	GCF_		
↪000006725_022640	6506	accessory	178	2.000e-		
↪10	41.600	0.603	0.264	5	51	
GCF_000006745	TFF-SF/T2SS	T2SS_gsp0	T4P_pild	GCF_		
↪000006745_021980	8766	mandatory	291	3.100e-		
↪88	295.800	1.000	0.832	28	269	
GCF_000006765	TFF-SF/T2SS	T2SS_gsp0	T4P_pild	GCF_		
↪000006765_044730	14545	mandatory	290	1.100e-		
↪88	297.200	1.000	0.828	31	270	
GCF_000006925	TFF-SF/T4P	T4P_pild	T4P_pild	GCF_		
↪000006925_026070	23874	mandatory	341	6.600e-		
↪118	394.300	0.950	0.941	18	338	
GCF_000006945	TFF-SF/T4P	T4P_pild	T4P_pild	GCF_		
↪000006945_030160	28596	mandatory	326	3.400e-		
↪113	378.800	0.933	0.966	3	317	
GCF_000006945	TFF-SF/T4P	T4P_pild	T2SS_gsp0	GCF_		
↪000006945_033450	28925	mandatory	155	2.900e-		
↪35	122.700	0.588	0.871	9	143	

best_solution_multisystems.tsv

This file give an overview of all hits identified as multi-systems in the best_solution

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type ordered_replicon --replicon-topology linear --models-dir tests/data/
↪models/ -m functional T12SS-multisystem --relative-path --sequence-db tests/
↪data/base/test_13.fasta -w 15
# Multisystems found:
replicon      model_fqn      function      gene_name      hit_
↪id          hit_pos      hit_status      hit_seq_len      hit_i_
↪eval        hit_score      hit_profile_cov      hit_seq_cov      hit_
↪begin_match      hit_end_match
UserReplicon      functional/T12SS-multisystem      T1SS_omf      T1SS_
↪omf          VICH001.B.00001.C001_
```

(continues on next page)

(continued from previous page)

↪01360	20	mandatory	484	3.200e-28	90.
↪000	0.985	0.820	80	476	
UserReplicon	functional/T12SS-multisystem			T1SS_omf	T1SS_
↪omf	VICH001.B.00001.C001_				
↪01506	35	mandatory	419	9.100e-35	111.
↪500	0.998	0.912	25	406	

rejected_candidates.txt

This file records all clusters or cluster combinations (if the “multi_loci” search mode is on) which have been discarded and the reason why they were not selected as systems.

The header is composed of the MacSyLib version and the command line used followed by the description of the cluster(s). The list of the hits composing the cluster is presented at the end of the cluster or clusters’ combination, followed by the reason why it has been discarded.

Note

This file is in human readable format. If you need to parse the information about rejected candidates, use the tsv formatted file rejected_candidates.tsv

```
# <header of my choice see FAQ>
# Rejected candidates:

Cluster:
- model: T4P
- hits: (GCF_000005845_025680, T4P_pilW, 2568), (GCF_000005845_025690, T4P_fimT, ↪
↪2569)
Cluster:
- model: T4P
- hits: (GCF_000005845_026930, T2SS_gsp0, 2693)
Cluster:
- model: T4P
- hits: (GCF_000005845_030080, T2SS_gsp0, 3008)
This candidate has been rejected because:
The quorum of mandatory genes required (4) is not reached: 1
The quorum of genes required (5) is not reached: 3
=====
Cluster:
- model: Archaeal-T4P
- hits: (GCF_000005845_019260, Archaeal-T4P_arCOG00589, 1926), (GCF_000005845_019310,
↪ Archaeal-T4P_arCOG02900, 1931)
This candidate has been rejected because:
The quorum of mandatory genes required (3) is not reached: 0
The quorum of genes required (3) is not reached: 2
=====
```

rejected_candidates.tsv

This file contains same information as *rejected_candidates.txt* but in tsv format, so it's more convenient to parse it. for instance with python and `pandas` library.:

```
import pandas as pd
pd.read_csv("path/to/rejected_candidates.tsv", sep='\t', comment='#')
```

As other file the first lines are comments and provides informations to indicate how this file has been produced.

- the macsylib version
- the model package and version used
- the command line used

then the following information separated by 'tabulation' character 't'

- **candidate_id** - An unique identifier of the candidate (for this run)
- **replicon** - The name of the replicon
- **model_fqn** - The model fully-qualified name
- **cluster_id** - An unique identifier for the cluster constituting the candidate
- **hit_id** - The identifier of the hit (as indicate in hmmer output)
- **hit_pos** - The position of the sequence in the replicon
- **gene_name** - The name of the component identified by the hit
- **function** - The name of the gene for which it it fulfill the function.
- **reasons** - The reasons why this cluster has been discarded. ther can be several reasons, in this case each reason are separated by '/

Note

A rejected candidate can be constituted of

- clusters (can have several clusters if the model is multi loci),
- loners

Example of *rejected_candidates.tsv*

```
# macsyfinder 20220805.dev
# models : TFF-SF-None
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsfinder --sequence-db_
↳data/base/GCF_000006745.fasta --models TFF-SF all --models-dir data/models/ --db-type_
↳gembase -w 15
# Rejected candidates found:
candidate_id      replicon      model_fqn      cluster_id      hit_id      hit_
↳pos      gene_name      function      reasons
GCF_000006745_Archaeal-T4P_1      GCF_000006745      TFF-SF/Archaeal-
↳T4P      c3      GCF_000006745_018740      1874      Archaeal-T4P_
↳arCOG00589      Archaeal-T4P_arCOG00589      The quorum of mandatory genes_
↳required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1
GCF_000006745_Archaeal-T4P_1      GCF_000006745      TFF-SF/Archaeal-
```

(continues on next page)

(continued from previous page)

```

→T4P      c3      GCF_000006745_018800      1880      Archaeal-T4P_
→arCOG00589      Archaeal-T4P_arCOG00589      The quorum of mandatory genes_
→required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1

GCF_000006745_Archaeal-T4P_2      GCF_000006745      TFF-SF/Archaeal-
→T4P      c4      GCF_000006745_026670      2667      Archaeal-T4P_
→arCOG02900      Archaeal-T4P_arCOG02900      The quorum of mandatory genes_
→required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1
GCF_000006745_Archaeal-T4P_2      GCF_000006745      TFF-SF/Archaeal-
→T4P      c4      GCF_000006745_026680      2668      Archaeal-T4P_
→arCOG02900      Archaeal-T4P_arCOG02900      The quorum of mandatory genes_
→required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1

GCF_000006745_ComM_4      GCF_000006745      TFF-SF/ComM      c11      GCF_
→000006745_017080      1708      ComM_comEC      ComM_comEC      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (4) is not_
→reached: 1

GCF_000006745_ComM_5      GCF_000006745      TFF-SF/ComM      c12      GCF_
→000006745_032430      3243      ComM_comEB      ComM_comEB      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (4) is not_
→reached: 2
GCF_000006745_ComM_5      GCF_000006745      TFF-SF/ComM      c13      GCF_
→000006745_017080      1708      ComM_comEC      ComM_comEC      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (4) is not_
→reached: 2

GCF_000006745_ComM_3      GCF_000006745      TFF-SF/ComM      c10      GCF_
→000006745_032430      3243      ComM_comEB      ComM_comEB      The quorum of_
→mandatory genes required (4) is not reached: 0/The quorum of genes required (4) is not_
→reached: 1

GCF_000006745_MSH_6      GCF_000006745      TFF-SF/MSH      c18      GCF_
→000006745_004600      460      MSH_mshA      MSH_mshA      The quorum of_
→mandatory genes required (3) is not reached: 1/The quorum of genes required (4) is not_
→reached: 1

GCF_000006745_T2SS_7      GCF_000006745      TFF-SF/T2SS      c25      GCF_
→000006745_021980      2198      T4P_pilD      T2SS_gsp0      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (6) is not_
→reached: 1

GCF_000006745_T4P_8      GCF_000006745      TFF-SF/T4P      c30      GCF_
→000006745_004240      424      T4P_pilT      T4P_pilT      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (5) is not_
→reached: 2
GCF_000006745_T4P_8      GCF_000006745      TFF-SF/T4P      c30      GCF_
→000006745_004250      425      T4P_pilU      T4P_pilU      The quorum of_
→mandatory genes required (4) is not reached: 1/The quorum of genes required (5) is not_
→reached: 2

GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c34      GCF_

```

(continues on next page)

(continued from previous page)

```

↪000006745_004240      424      T4P_pilT      T4P_pilT      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c34      GCF_
↪000006745_004250      425      T4P_pilU      T4P_pilU      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c35      GCF_
↪000006745_007820      782      T4P_pilE      T4P_pilE      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c35      GCF_
↪000006745_007830      783      T4P_fimT      T4P_fimT      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c35      GCF_
↪000006745_007840      784      T4P_pilW      T4P_pilW      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c35      GCF_
↪000006745_007850      785      T4P_pilX      T4P_pilX      The quorum of_
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12      GCF_000006745      TFF-SF/T4P      c35      GCF_
↪000006745_007860      786      T4P_pilV      T4P_pilV      The quorum of_
↪mandatory genes required (4) is not reached: 2

```

Note

If a timeout is set to limit the time spent in best solution resolution. This timeout is applied per replicon. If the best solution resolution reach the timeout for a replicon, a WARNING is raised in *macysfinder.log*. The warning is also report in the following files:

- *best_solution.tsv*
- *best_solution_summary.tsv*
- *all_best_solutions.tsv*
- *all_systems.tsv* and *all_systems.txt*
- *rejected_candidates.tsv* and *rejected_candidates.txt*

for instance

```

# <header of my choice see FAQ>
#
# WARNING: The replicon 'GCF_000006765' has been SKIPPED. Cannot be solved before_
↪timeout.
#
replicon hit_id ...

```

Output files for the “unordered replicon” search mode**Systems detection results**

As for ordered replicons, several output files are provided.

- *all_systems.txt* - This file contains the description of candidate systems found.
- *all_systems.tsv* - The same information as in *all_systems.txt* but in the tabulated tsv format.

- *uncomplete_systems.txt* - This file contains occurrences for systems that did not complete models' definitions and that were therefore not kept as candidate systems.

Note

In this *unordered* search mode, there is no notion of order or distance of the components along the replicon. The clustering step is skipped by MacSyLib, and it is therefore “only” checked for each type of system being searched whether there is the genetic potential to fulfil its model definition.

all_systems.txt

This file contains potential systems for unordered replicon in human readable format.

In this file, for each component of each searched system's model, we report the number of hits found. For the description of the fields, see *above*.

Warning

In this mode the *forbidden* genes are reported here to the user. As we do not know if they co-localize (cluster) with the other genes they could be present in the replicon, yet far away - or very close on the contrary - to the potential system.

```
# <header of my choice see FAQ>
# Systems found:

This replicon contains genetic materials needed for system TFF-SF/T4P_single_locus

system id = Unordered_T4P_single_locus_1
model = TFF-SF/T4P_single_locus
replicon = Unordered
hits = [('GCF_000006845_000250', 'T4P_pilY', 25), ('GCF_000006845_000700', 'T4P_pilY', 70), ('GCF_000006845_001030', 'T4P_pilQ', 103), ('GCF_000006845_001040', 'T4P_pilP', 104), ('GCF_000006845_001050', 'T4P_pilO', 105), ('GCF_000006845_001060', 'T4P_pilN', 106), ('GCF_000006845_001070', 'T4P_pilM', 107), ('GCF_000006845_003200', 'T4P_pilU', 320), ('GCF_000006845_004190', 'T4P_fimT', 419), ('GCF_000006845_004200', 'T4P_pilV', 420), ('GCF_000006845_004210', 'T4P_pilW', 421), ('GCF_000006845_004220', 'T4P_pilX', 422), ('GCF_000006845_004230', 'T4P_pilA', 423), ('GCF_000006845_010160', 'T4P_pilA', 1016), ('GCF_000006845_012440', 'T4P_pilA', 1244), ('GCF_000006845_014270', 'T4P_pilC', 1427), ('GCF_000006845_014280', 'T4P_pilD', 1428), ('GCF_000006845_014310', 'T4P_pilB', 1431), ('GCF_000006845_016430', 'T4P_pilT', 1643), ('GCF_000006845_016440', 'T4P_pilU', 1644)]
wholeness = 0.889

mandatory genes:
- T4P_pilE: 0 ()
- T4P_pilB: 1 (T4P_pilB)
- T4P_pilC: 1 (T4P_pilC)
- T4P_pilO: 1 (T4P_pilO)
- T4P_pilQ: 1 (T4P_pilQ)
- T4P_pilN: 1 (T4P_pilN)
```

(continues on next page)

(continued from previous page)

```
- T4P_pilT: 1 (T4P_pilT)
- T4P_pilD: 1 (T4P_pilD)
```

accessory genes:

```
- T4P_pilA: 3 (T4P_pilA, T4P_pilA, T4P_pilA)
- T4P_pilV: 1 (T4P_pilV)
- T4P_pilY: 2 (T4P_pilY, T4P_pilY)
- T4P_pilW: 1 (T4P_pilW)
- T4P_pilX: 1 (T4P_pilX)
- T4P_fimT: 1 (T4P_fimT)
- T4P_pilM: 1 (T4P_pilM)
- T4P_pilP: 1 (T4P_pilP)
- T4P_pilU: 2 (T4P_pilU, T4P_pilU)
- MSH_mshM: 0 ()
```

neutral genes:

forbidden genes:

Use ordered replicon to have better prediction.

all_systems.tsv

This file contains the same information as in *all_systems.txt* but in *tsv* format. For the description of the fields, see *above*.

Note

This file can be easily parsed with pandas:

```
import pandas as pd
pot_systems = pd.read_csv('all_systems.tsv', sep='\t', comment='#')
```

```
# <header of my choice see FAQ>
```

```
# Likely Systems found:
```

replicon	hit_id	gene_name	hit_pos	model_fqn	sys_id	sys_wholeness	hit_
↪gene_ref	hit_status	hit_seq_len	hit_i_eval	hit_score	hit_		
↪profile_cov	hit_seq_cov	hit_begin_match	hit_end_match	used_in			
Unordered	GCF_0000006845_014310	T4P_pilB	1431	TFF-SF/T4P_single_locus_			
↪Unordered_T4P_single_locus_1	0.889	T4P_pilB	mandatory	558	3.8e-		
↪178	589.000	0.964	0.731	146	553		
Unordered	GCF_0000006845_014270	T4P_pilC	1427	TFF-SF/T4P_single_locus_			
↪Unordered_T4P_single_locus_1	0.889	T4P_pilC	mandatory	410	1.9e-		
↪131	434.800	0.997	0.817	72	406		
Unordered	GCF_0000006845_014280	T4P_pilD	1428	TFF-SF/T4P_single_locus_			
↪Unordered_T4P_single_locus_1	0.889	T4P_pilD	mandatory	286	2.8e-		
↪82	272.300	1.000	0.829	28	264		
Unordered	GCF_0000006845_001060	T4P_pilN	106	TFF-SF/T4P_single_locus_			
↪Unordered_T4P_single_locus_1	0.889	T4P_pilN	mandatory	199	2.3e-		

(continues on next page)

(continued from previous page)

↪33	112.200	0.986	0.714	7	148				
Unordered	GCF_000006845_001050				T4P_pilO	105	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilO		mandatory	215	2.9e-
↪37	124.800	0.980	0.693	23	171				
Unordered	GCF_000006845_001030				T4P_pilQ	103	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilQ		mandatory	723	1.9e-
↪62	206.600	0.935	0.238	548	719				
Unordered	GCF_000006845_016430				T4P_pilT	1643	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilT		mandatory	347	6.9e-
↪167	551.400	0.997	0.983	2	342				
Unordered	GCF_000006845_004190				T4P_fimT	419	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_fimT		accessory	221	2.7e-
↪23	78.900	0.985	0.294	7	71				
Unordered	GCF_000006845_004230				T4P_pilA	423	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilA		accessory	162	8.6e-
↪20	67.800	0.744	0.389	9	71				
Unordered	GCF_000006845_010160				T4P_pilA	1016	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilA		accessory	149	1.3e-
↪15	54.300	0.821	0.430	5	68				
Unordered	GCF_000006845_012440				T4P_pilA	1244	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilA		accessory	129	1.5e-
↪19	67.000	0.859	0.519	6	72				
Unordered	GCF_000006845_001070				T4P_pilM	107	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilM		accessory	371	3.3e-
↪43	144.300	0.988	0.429	30	188				
Unordered	GCF_000006845_001040				T4P_pilP	104	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilP		accessory	181	2.7e-
↪34	115.600	1.000	0.735	13	145				
Unordered	GCF_000006845_003200				T4P_pilU	320	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilU		accessory	376	2.2e-
↪170	562.600	0.985	0.896	16	352				
Unordered	GCF_000006845_016440				T4P_pilU	1644	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilU		accessory	408	1.5e-
↪127	421.800	0.994	0.833	40	379				
Unordered	GCF_000006845_004200				T4P_pilV	420	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilV		accessory	203	9.6e-
↪16	54.600	1.000	0.276	14	69				
Unordered	GCF_000006845_004210				T4P_pilW	421	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilW		accessory	326	1.7e-
↪10	38.000	0.517	0.190	17	78				
Unordered	GCF_000006845_004220				T4P_pilX	422	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilX		accessory	203	2.8e-
↪18	62.600	0.983	0.286	17	74				
Unordered	GCF_000006845_000250				T4P_pilY	25	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilY		accessory	1006	2.2e-
↪57	191.700	0.728	0.389	463	853				
Unordered	GCF_000006845_000700				T4P_pilY	70	TFF-SF/T4P_single_locus_		
↪Unordered_T4P_single_locus_1		0.889			T4P_pilY		accessory	1047	1.9e-
↪57	191.900	0.721	0.362	516	894				

uncomplete_systems.txt

This file is created when a search is performed in the *unordered replicon* mode. This file lists models that probably do not have full systems in the replicon(s). For each model, the reason why it is not fulfilled is reported, followed by the model description and the components found.

```
# <header of my choice see FAQ>
# Unlikely Systems found:

This replicon probably not contains a system TFF-SF/T2SS:
The quorum of mandatory genes required (4) is not reached: 1
The quorum of genes required (6) is not reached: 2

system id = Unordered_T2SS_3
model = TFF-SF/T2SS
replicon = Unordered
hits = [('GCF_000006845_002600', 'Tad_tadD', 260), ('GCF_000006845_014280', 'T4P_pilD', 1428), ('GCF_000006845_016430', 'T4P_pilT', 1643)]
wholeness = 0.143

mandatory genes:
  - T2SS_gspD: 0 ()
  - T2SS_gspE: 0 ()
  - T2SS_gspF: 0 ()
  - T2SS_gspG: 0 ()
  - T2SS_gspC: 0 ()
  - T2SS_gspO: 1 (T4P_pilD)

accessory genes:
  - T2SS_gspM: 0 ()
  - T2SS_gspH: 0 ()
  - T2SS_gspI: 0 ()
  - T2SS_gspJ: 0 ()
  - T2SS_gspK: 0 ()
  - T2SS_gspN: 0 ()
  - T2SS_gspL: 0 ()
  - Tad_tadD: 1 (Tad_tadD)

neutral genes:

forbidden genes:
  - T4P_pilT: 1 (T4P_pilT)

Use ordered replicon to have better prediction.
```

Hammer results' output files

Raw Hammer outputs are provided, as long with processed tabular outputs that include hits filtered as specified by the user. For instance, the Hammer search for SctC homologs with the corresponding profile will result in the creation of two output files: “sctC.search_hmm.out” for the raw HMMER output file and “sctC.res_hmm_extract” for the output file after processing/filtering of the HMMER results by MacSyLib.

The processed output file “sctC.res_hmm_extract” recalls on the first lines the parameters used for hits filtering and relevant information on the matches, as in this example:

```
# gene: sctC extract from /Users/bob/macsylib_results/
      macsylib-20130128_08-57-46/sctC.search_hmm.out hmm output
# profile length= 544
# i_evalue threshold= 0.001000
# coverage threshold= 0.500000
# hit_id replicon_name position_hit hit_sequence_length gene_name gene_system i_evalue
↪score
      profile_coverage sequence_coverage begin end
PSAE001c01_006940      PSAE001c01      3450      803      sctC      T3SS      1.1e-41 141.6
      0.588235 0.419676      395      731
PSAE001c01_018920      PSAE001c01      4634      776      sctC      T3SS      9.2e-48 161.7
      0.976103 0.724227      35      596
PSAE001c01_031420      PSAE001c01      5870      658      sctC      T3SS      2.7e-52 176.7
      0.963235 0.844985      49      604
PSAE001c01_051090      PSAE001c01      7801      714      sctC      T3SS      1.9e-46 157.4
      0.571691 0.463585      374      704
```

Logs and configuration files

Three specific output files are systematically built, whatever the search mode, to store information on MacSyLib’s execution:

- **macsylib.conf** - contains the configuration information of the run. It is useful to recover all the parameters used for the run.
- **macsylib.log** - the log file, contains raw information on the run. Please send it to us with any **bug report**.

1.1.2 MacSyLib functioning

Macromolecular models

MacSyLib relies on the definition of models of macromolecular systems as a **set of models’ components** to be searched by similarity search, and a **set of rules** regarding their genomic organization and their requirement level to make a complete system (mandatory, accessory components, number of components required).

See [below](#) for more details on MacSyLib’s modelling scheme and the section on [Functioning](#) for the principles of the MacSyLib’s search engine.

A **MacSyLib model** (masy-model for short) is the association of several elements:

- a **definition** which describes the system to detect with a specific **XML grammar** that is described [below](#).
- a set of **HMM profiles** (one per component/gene in the model) to enable the similarity search of the systems’ components with the HMMER program.

The models are grouped by *family* possibly gathering *sub-families* (multiple levels allowed), for instance *Secretion*, *Cas-proteins*... A set of models from a same family (coherent set) of systems to detect is called hereafter a **masy-model package**.

Note

For details on how to create your own masy-models, have a look at the [Modeller Guide](#).

Installing models

How to install new models

MacSyLib does not provide models. You must install models before using it. The `msl_data` utility tool is shipped with *MacSyLib* to deal with macsy-models:

```
msl_data <subcommand> [options]
```

The main sub-commands are

- `msl_data available` to get the list of macsy-models available
- `msl_data search` to search a model given its name or a pattern in its description
- `msl_data install` to install a macsy-model package (the installed version can be set see `-help`)
- `msl_data cite` to retrieve information on how to cite the model
- `msl_data show` to show the structure of model package
- `msl_data definition` to display one or a set of model definition
- `msl_data --help` to get the extended list of available subcommands
- `msl_data <subcommand> --help` to get help about the specified subcommand

Where the models are located

MacSyLib looks at several locations to find macsy-models.

system-wide installation

By default `msl_data` installs models in a shared location (set by `-install-data` option) that is `/usr/share/macsylib/` or `/usr/local/share/macsylib` depending on your Operating System distribution. If you use a *virtualenv*, the shared resources are located in the `<virtualenv>/share/macsylib` directory.

user-wide installation

If you don't own rights to install system-wide, you can install models in the MacSyLib's cache located in your home: `$HOME/macsylib/data/`. `msl_data` installs packages in this location when you use the `-user` option. The packages installed in user land is added to the system-wide packages.

Note

If two packages have the same name, the package in the user land supersedes the system-wide package.

project-wide installation

If you cannot install macsy-model packages in system or user land locations, you can install models in specific directory with the `-target` option.:

```
msl_data install --target <my_models>
```

The specify this specific location with the `--models-dir` *command-line option*.:

```
MacSyLib --db-type ordered_replicon --models-dir=my_models --models TFF-SF all --  
↪ sequence-db my_genome.fasta
```

The path must point at a directory that contains macsy-model packages as described [above](#).

MacSyLib's search engine

Functioning overview

MacSyLib is able to use a variety of input files and options. See [Input dataset](#) for more details. Below follows a description of its overall functioning.

A. Searching for Systems' components

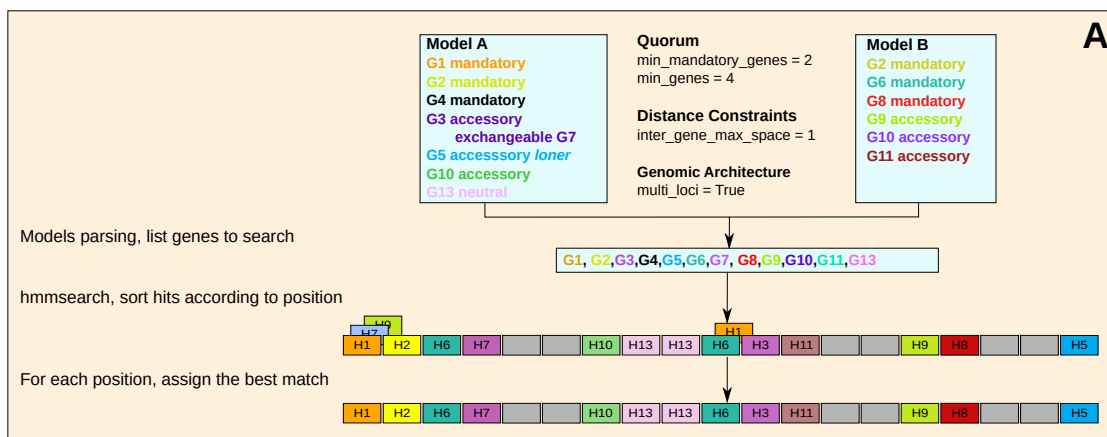
Initially, MacSyLib **searches for the components** of the *System(s)* to detect by sequence similarity search (`macsylib.search_systems.search_systems()`).

1. From the list of *System(s)* to detect, a **non-redundant list of components to search** is built. For each system, the list can include:

- mandatory components
- accessory components
- neutral components
- forbidden components
- exchangeable components that can be functionally replaced by other components (usually by analogs or homologs). These other components are thus also added to the list of components to search.

See [here for more details on writing MacSyLib's models](#).

2. HMMER is run on the corresponding set of components' HMM profiles, and the hits are filtered according to the criteria defined by the user or by default (see [Hmmer options](#) and for more, the [API report](#) object page). This step, and the extraction of significant hits can be performed in parallel (*worker* config option). See the [Config object](#), and the [search_genes API](#) for more details.



B. Hits browsing

The following steps depend on whether the input dataset is **ordered** (complete or nearly complete genome(s)), or **unordered** (metagenomes, or unassembled genome(s)) (see the [Input dataset](#) section).

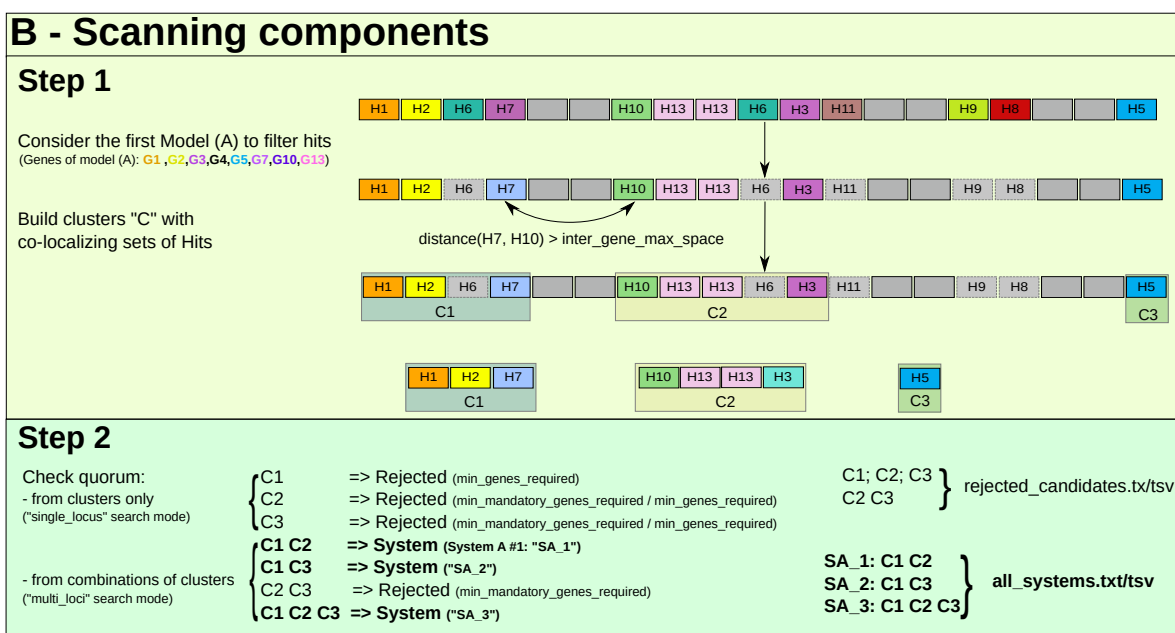
In the case of **ordered datasets** (*ordered_replicon* or *gembase* search mode), the hits are filtered to keep only hits related to the system's model we are looking for. These hits are used to build **clusters of co-localized genes** as defined in the *macsy-model files*. These clusters are then screened to check for the model specifications such as the minimal quorum of "Mandatory" or "Accessory" genes, or the absence of "Forbidden" components.

When the **gene order is unknown** (*unordered* search mode) the power of the analysis is more **limited**. In this case, the presence of systems can only be suggested on the basis of the **quorum** of components - and not based on genomic context information.

For *ordered* datasets: building clusters of components

The following two steps are reiterated for each model being searched.

1. The search starts with the filtering of hits to only keep the **hits that are listed in the model** (mandatory, accessory, neutral, forbidden, exchangeable).
2. MacSyLib searches for sets of contiguous hits to build **clusters**, following the (**co-localization criterion**) for each replicon, as defined in the MacSyLib's model. Two hits are deemed contiguous if their genomic location is separated by less than d protein-encoding genes, d being the maximum of the two *inter_gene_max_space* parameters from the two genes with hits (system-wise, or gene-specific parameter). The *loner* components may form a cluster on their own.



Once performed for each model searched, the *next step* is performed.

Note

The clusters that do not fulfill the quorum requirements are stored in the *rejected_candidates.txt/tsv* file.

Note

If several hits which co-locate have the same gene in the model. MSL does not consider them as a cluster.

Note

If a group of gene which co-locate is composed solely of Neutral genes, It has not considered by MSL as a cluster.

For unordered datasets:

For each model being searched:

1. The Hits are filtered by model.
2. They are used to check if they reach the quorum (i.e., the clustering step is skipped as there is no notion of genetic distance in this search mode).
3. For each system, if the quorum is reached, hits are reported in the *all_systems.tsv* output file. It has to be noted that forbidden components are listed too, as they can also be informative for the user.

Note

The “unordered” mode of detection is less powerful, as a single occurrence of a given model is filled for an entire dataset with hits that origin is unknown. Please consider the assessment of systems with caution in this mode.

For unordered datasets, the **search so ends**, and you can generate the final *output files* with `macsylib.io` module (`macsylib.io.likely_systems_to_tsv()` and `macsylib.io.unlikely_systems_to_txt()`).

C. Computing candidate Systems’ scores (ordered mode)

This step only applies to the most powerful search mode, i.e., on **ordered datasets**.

The **search engine** explores the space of possible Solutions regarding the presence of Systems in replicons analysed. It creates clusters of hits for Systems’ components separately for each System searched, and therefore might find **candidate occurrences of Systems that overlap** in terms of components. Moreover, if a System is possibly encoded at several locations on the replicon analysed (option *multi_loci* set to “True” in the model), this calls for a **combinatorial screening** of the different clusters to assemble them into coherent systems regarding the macsy-models.

- For a given model, clusters are used to “fill up” Systems’ occurrence(s) according to the **quorum criteria** defined in the System’s model (see methods `macsylib.system.OrderedMatchMaker.match()`, `macsylib.system.UnorderedMatchMaker.match()`):

The *min_genes_required* and *min_mandatory_genes_required* thresholds must be reached.

- In the case of the *single-locus system* search mode (default), each cluster in addition to potential loners are evaluated for System’s assessment separately.
- In the case of the *multi-loci system* search mode (`multi_loci=True`), each possible combination of clusters is confronted to the quorum of the System being examined.

The sets of clusters that fulfill the quorum are reported as candidate Systems in the *all_systems.txt* and *all_systems.tsv* output files (see *Output format*), and they obtain a **System’s score** (see below).

The clusters that do not allow to form a candidate System are reported in the *rejected_candidates.txt* and *rejected_candidates.tsv* output files.

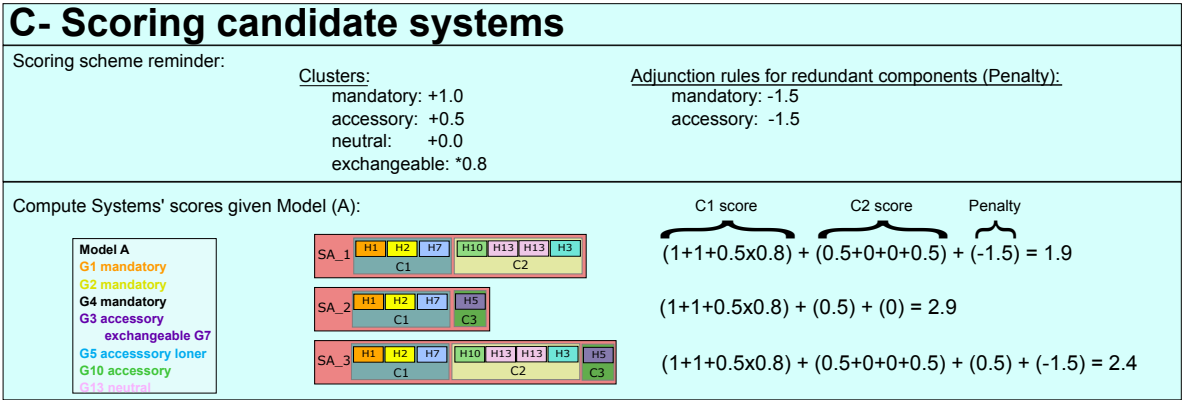
- We introduce a **scoring scheme for candidate Systems**, to easily separate combinations of clusters that are readily more similar to a system’s model than others.

The assumptions behind this scoring scheme are the following:

- We set a score for the different types of genes/components when defining a **cluster’s score**. Here are the default values, but these *can be changed*:
 - * +1.0 is added when a *mandatory* gene is present
 - * +0.5 is added when an *accessory* gene is present
 - * +0.0 is added when a *neutral* gene is present

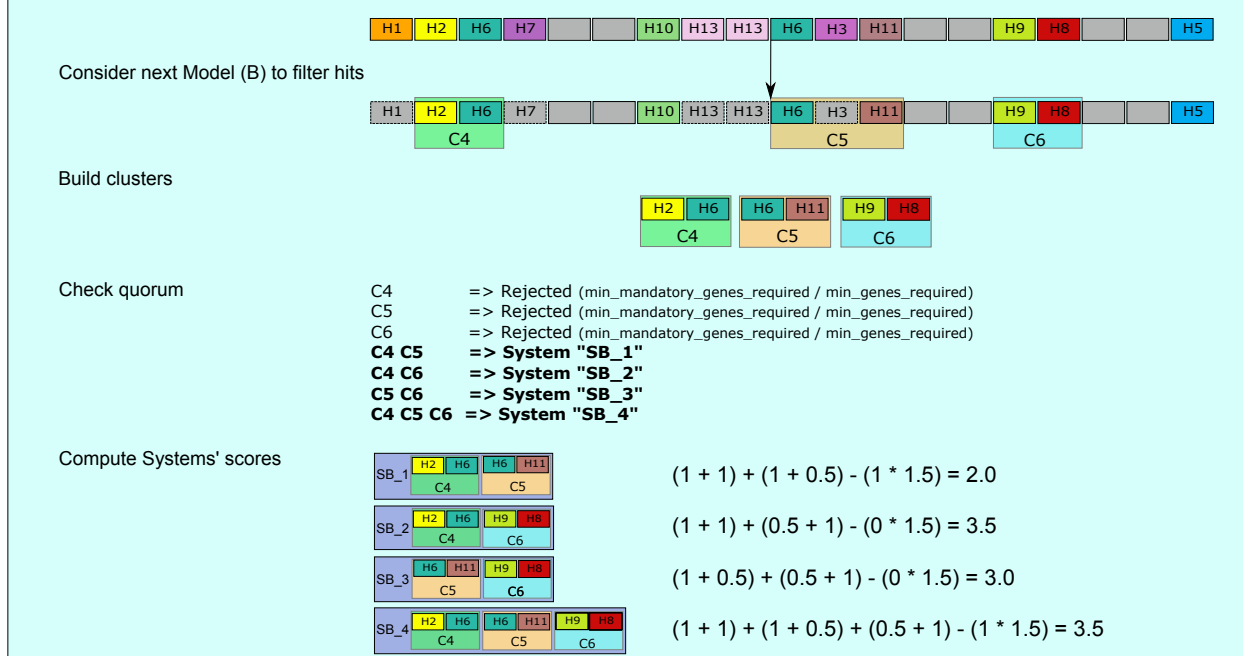
- * *0.8 (a factor of 0.8) is applied to the above-scores when the function is fulfilled by an *exchangeable* gene
- * *0.7 (a factor of 0.7) is applied to the above-scores if the gene is a *loner* and *multi system* component.
- When combinations of clusters are explored in order to fulfill macsy-models’ requirements and build candidate systems (“multi_loci” mode, several clusters can make a complete *System*), we sum the score of clusters to assign a *System*’s score.
- In addition, we want to **favor concise sets of clusters** to fulfill a *System*’s model. We thus **penalize the adjunction of a cluster** to a candidate *System* when this cluster does not bring any new components to the *System*’s quorum, or when it brings **redundant components**. Thus:
 - * -1.5 is added when a **redundant** mandatory gene is added when adjuncting the cluster to a candidate *System*
 - * -1.5 is added when a **redundant** accessory gene is added when adjuncting the cluster to a candidate *System*
 - * for the components that are *loner* and *multi system*, the score of the loner component is added only if the function is not fulfilled in the other clusters. In this case, even if there are several occurrences of the component, it is counted only once (and no penalty is applied).
- Only candidate sets of clusters that fulfill a macsy-model and that are thus designated candidate *Systems*, obtain a **System’s score**

In summary, a Systems’s score is made of two parts: the **sum of the scores** of the Clusters it is made of, plus a **penalty part** to avoid too much component’s redundancy in Cluster’s combinations. The systems’ scoring step is exemplified in this figure:



D. Repeat operations B and C for the other models being searched

D- Iterating all steps over systems



This search for candidate *Solutions* from different models results in a number of possible *Solutions* representing combinations of putative sets of *Systems* in the analysed dataset.

E. Computing possible Solutions, defining the best one (ordered mode)

At the end of the previous step MacSyLib has computed all potential *Systems* present in the replicon, made of combinations of Clusters and *loner* components that fulfill the model's requirements, which are themselves made of a subset of Hits (remember, Hits are at 1st filtered and treated separately for each model of System to be detected). Candidate *Systems* may thus overlap by being partly made of the same components, or even partly being made of the same Clusters.

We define a *Solution* as being a **set of compatible Systems**, i.e. that do not have any overlaps between their components. All possible *Solutions* are combinatorially explored and consist in all possible sets of compatible *Systems*.

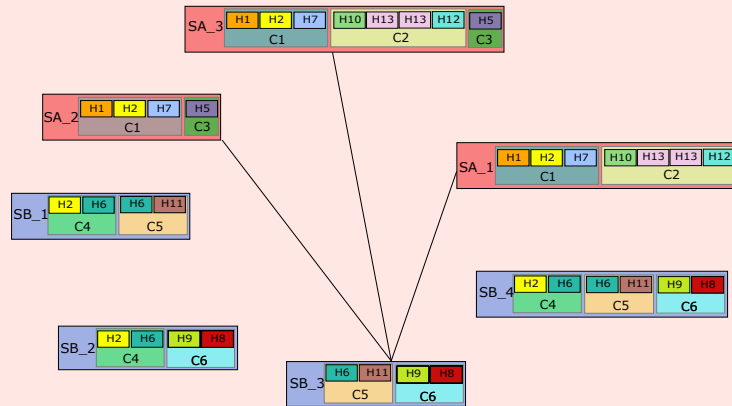
A scoring scheme enables to separate between sets of *Solutions*. A **Solution's score** is basically the **sum of its Systems' scores**. The overall procedure of exploring the space of all possible *Solutions* while finding the optimal one, i.e. that with the maximal score, is performed at once using a graph solution to this problem, implemented in the `networkx` package.

We create a graph where each potential *System* is a vertex, and we create an edge between pairs of vertices if they do not share any components (compatible *Systems*). Once the graph is created we look for the **maximum clique** which maximizes the score. This allows to provide the user with one, or multiple *Solutions* that have the **best score possible** among all combinations of compatible *Systems*. All this step is performed by the `macsylib.solution.find_best_solution()`

E- Computing solutions

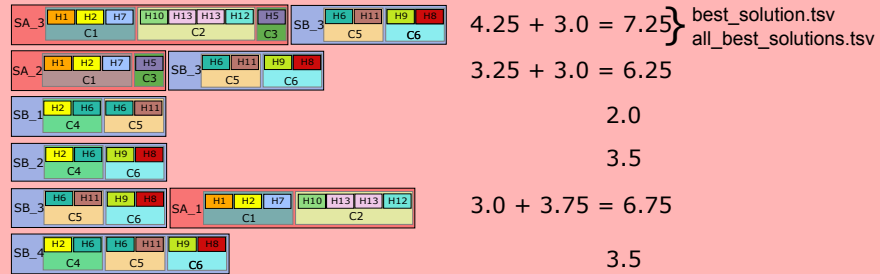
Step 1

Build a graph of Systems
(edges between compatible Systems)



Step 2

Compute the scores of maximal cliques



1.1.3 Frequently Asked Questions

Frequently Asked Questions

How to report an issue?

If you encounter a problem while running MacSyLib, please submit an issue on the dedicated page of the [GitHub project](#)

To ensure we have all elements to help, please provide:

- a concise description of the issue
- the expected behavior VS observed one
- the exact command-line used
- the version of MacSyLib used
- the exact error message, and if applicable, the `<macsylib>.log` and `<macsylib>.conf` files
- if applicable, an archive (or link to it) with the output files obtained
- if possible, the smallest dataset there is to reproduce the issue
- if applicable, this would also include the macy-models (XML models plus HMM profiles) used (or precise version of the models if there are publicly available). Same as above, if possible, please provide the smallest set possible of models and HMM profiles.

All these will definitely help us to help you! ;-)

Note

If you use macsylib in higher level script you can change <macsylib> by the name of your tool by setting it in `macsylib.config.MacsyDefault prog_name` parameter.

How to cite MacSyLib and published macy-models?

- Néron, Bertrand et al 2023, *Peer Community Journal* MacSyFinder v2: Improved modelling and search engine to identify molecular systems in genomes.
- Abby and Rocha 2012, *PLoS Genetics*, for the study of the evolutionary relationship between the T3SS and the bacterial flagellum, and how were designed the corresponding HMM protein profiles.
- Abby et al. 2016, *Scientific Reports*, for the description of bacterial protein secretion systems' models (TXSScan: T1SS, T2SS, T5SS, T6SS, T9SS, Tad, T4P).
- Denise et al. 2019, *PLoS Biology*, for the description of type IV-filament super-family models (TFF-SF: T2SS, T4aP, T4bP, Com, Tad, archaeal T4P).
- Rendueles et al. 2017, *PLoS Pathogens*, for the CapsuleFinder set of models.
- Couvin, Bernheim et al. 2018, *Nucleic Acids Research*, for the updated version of the set of Cas systems' models, CasFinder.

How to use MacSyLib ?

Here are an example of analyse with a python script using macsylib:

```
import os
import logging
from argparse import Namespace

import macsylib.config
import macsylib.registries
import macsylib.utils
import macsylib.search_systems
import macsylib.io
from macsylib.system import HitSystemTracker

defaults = macsylib.config.MacsyDefaults()
settings = Namespace(
    db_type='ordered_replicon',
    sequence_db='test.fasta',
    models=['TXSScan', 'all'], # this model must be installed with msl_data scripts
    worker=4,
    out_dir='my_results'
)
config = macsylib.config.Config(defaults, settings)

os.makedirs(config.working_dir()) # working_dir = out_dir; you have to create this_
↪directory

macsylib.init_logger(log_file=os.path.join(config.working_dir(), config.log_file()))
macsylib.logger_set_level(level=logging.INFO)
logger = logging.getLogger('macsylib')
```

(continues on next page)

(continued from previous page)

```

model_registry = macsylib.registries.ModelRegistry()

for model_dir in config.models_dir():
    models_loc_available = macsylib.registries.scan_models_dir(model_dir)
    for model_loc in models_loc_available:
        model_registry.add(model_loc)
models_def_to_detect, models_fam_name, models_version = macsylib.utils.get_def_to_
↳detect(config.models(),

↳model_registry)

all_systems, rejected_candidates = macsylib.search_systems.search_systems(config, model_
↳registry, models_def_to_detect,

logger)

track_multi_systems_hit = HitSystemTracker(all_systems)

with open(os.path.join(config.working_dir(), 'all_systems.txt'), "w", encoding='utf8') as
↳tsv_file:
    macsylib.io.systems_to_tsv('TXSScan', '1.1.3',
                              all_systems,
                              track_multi_systems_hit,
                              tsv_file,
                              header=lambda model_name, model_v, skipped_replicons: f'#{
↳created by {__file__} script with models {model_name}-{model_v}'}

```

The script above will produce the file below with a summary of all systems found.

```

# created by /home/bneron/Projects/GEM/MacSyFinder/src/macsylib/Sandbox/msl_example.py
↳script with models TXSScan-1.1.3
# Systems found:

```

replicon	hit_id	gene_name	hit_pos	model_fqn	sys_	
↳id	sys_loci	locus_num	sys_wholeness	sys_score	sys_	
↳occ	hit_gene_ref	hit_status	hit_seq_len	hit_i_		
↳eval	hit_score	hit_profile_cov	hit_seq_cov	hit_begin_		
↳match	hit_end_match	counterpart	used_in			
test	VICH001.B.00001.C001_01397	T1SS_abc	11	TXSScan/bacteria/		
↳diderm/T1SS	test_T1SS_1	1	1	1.000	2.	
↳700	2	T1SS_abc	mandatory	721	2.4e-157	516.
↳300	0.996	0.698	175	677		
test	VICH001.B.00001.C001_01398	T1SS_mfp	12	TXSScan/bacteria/		
↳diderm/T1SS	test_T1SS_1	1	1	1.000	2.	
↳700	2	T1SS_mfp	mandatory	467	2.3e-72	235.
↳600	1.000	0.797	85	456		
test	VICH001.B.00001.C001_01399	T1SS_abc	13	TXSScan/bacteria/		
↳diderm/T1SS	test_T1SS_1	1	1	1.000	2.	
↳700	2	T1SS_abc	mandatory	720	2.1e-158	519.
↳700	1.000	0.699	167	669		
test	VICH001.B.00001.C001_01506	T1SS_omf	23	TXSScan/bacteria/		
↳diderm/T1SS	test_T1SS_1	1	-1	1.000	2.	
↳700	2	T1SS_omf	mandatory	419	9.3e-35	111.
↳500	0.998	0.912	25	406		

(continues on next page)

(continued from previous page)

test	VICH001.B.00001.C001_01397	T1SS_abc	11	TXSScan/bacteria/		
↪diderm/T1SS	test_T1SS_2	1	1	1.000	2.	
↪700	2	T1SS_abc	mandatory	721	2.4e-157	516.
↪300	0.996	0.698	175	677		
test	VICH001.B.00001.C001_01398	T1SS_mfp	12	TXSScan/bacteria/		
↪diderm/T1SS	test_T1SS_2	1	1	1.000	2.	
↪700	2	T1SS_mfp	mandatory	467	2.3e-72	235.
↪600	1.000	0.797	85	456		
test	VICH001.B.00001.C001_01399	T1SS_abc	13	TXSScan/bacteria/		
↪diderm/T1SS	test_T1SS_2	1	1	1.000	2.	
↪700	2	T1SS_abc	mandatory	720	2.1e-158	519.
↪700	1.000	0.699	167	669		
test	VICH001.B.00001.C001_01506	T1SS_omf	23	TXSScan/bacteria/		
↪diderm/T1SS	test_T1SS_2	1	-1	1.000	2.	
↪700	2	T1SS_omf	mandatory	419	9.3e-35	111.
↪500	0.998	0.912	25	406		

The code and the data are available

`msl_example.py`.

`test.fasta`.

For more details check the developer guide [Developer Guide](#) and api documentation [MacSyLib API documentation](#)

For more example check [mascyfinder source code](#)

What search mode to be used?

Depending on the type of dataset you have, you will have to adapt MacSyLib's search mode.

- If you have a fasta file from a complete genome where **proteins are ordered** according to the corresponding genes' order along the replicon, your dataset is entitled to the most powerful search mode (see below): *ordered_replicon* and use the following option `-db-type ordered_replicon`.
- If you have a fasta file of proteins with **no sense of the order** of the corresponding genes along the chromosome(s) or replicon(s), you will have to use the *unordered* search mode with the following option: `-db-type unordered`
- If you have **multiple ordered replicons** to analyse at once, you can follow the *Gembase* convention to name the proteins in the fasta file, so that the original replicons can be assessed from their name: [see here for a description](#).

Note

- When the **gene order is known** (*ordered_replicon* search mode) the power of the analysis is **maximal**, since both the genomic content and context are taken into account for the search.
- When the **gene order is unknown** (*unordered* search mode) the power of the analysis is more **limited** since the presence of systems can only be suggested on the basis of the quorum of components - and not based on genomic context information.

More on MacSyLib's functioning [here](#).

How to deal with fragmented genomes (MAGs, SAGs, draft genomes)?

There are more and more genomes available which are not completely assembled, or are fragmented and incomplete. In this case, several options can be considered.

1. If your genome is at least partially assembled and contigs are not too short, you might “feel lucky” and first consider to run MacSyLib with the *ordered_replicon* mode. It could be particularly efficient if you are investigating systems encoded by compact loci (Cas systems, some secretion systems...), as they might be encoded by a single contig.
2. On top of the *ordered_replicon* mode, you might add the option “multi-loci” to the systems to annotate (if not already the case), in order to maximize the chance to annotate an entire system, even if encoded across several contigs.
3. The *unordered* mode can be used in complement of the two above options, e.g. to retrieve some of the missing components. It will enable to assess the genetic potential and possible presence of a system, independently of the quality of assembly of the genome. It might also be the only reasonable option if the genome is too fragmented and/or too incomplete.

Note

- The results obtained with the *ordered_replicon* mode on a fragmented genome have to be considered carefully, especially with respect to the contigs’ borders, as some proteins from different contigs might be artificially considered as closely encoded.
- To retrieve “fragments” of a system not found to reach the quorum in the *ordered_replicon* mode, it is possible to retrieve clusters of genes from the *rejected_candidates.tsv* file.

How to interpret the results from an *unordered* search?

As mentioned above, in the *unordered* search mode, the inference of a system’s presence is only based on the list of components found in the protein dataset. Thus, the kind of search specificity provided when using the genomic context (components next to each other are more likely to be part of a same system’s occurrence) is not within reach.

In the *unordered* search mode, the number of proteins selected as system’s components (based on the filtering of HMM profiles’ similarity search) is reported. We decided to report all kinds of system’s components, including the *forbidden* ones in order to raise awareness of the user -> even if all constraints are met for the system’s inference (here, the quorum: minimal number of components), it cannot be excluded that a *forbidden* component would lie next to the *bona fide* components (*mandatory* and *accessory* ones) in the genome...

In the end, the *unordered* search mode provides an idea as to whether the **genetic potential** for a given system is found in the set of proteins analysed, with no attempt to assign proteins to particular systems’ occurrences, nor guarantee as to whether *forbidden* components should be considered for the potential occurrences.

Where to find MacSyLib models?

Since version 2, there is a tool to enable the download and installation of published models from a repository: the *mssl_data* tool.

See [here for details](#) on how to use it.

What are the rules for options precedence?

MacSyLib offers many ways to parametrize the systems’ search: through the command-line, through various configuration files (for the models, for the run, etc...). It offers a large control over the search engine. But it also means you can get lost in configuration. ;-)

Here is a recap of the rules for options precedence. In a general manner, the command line always wins.

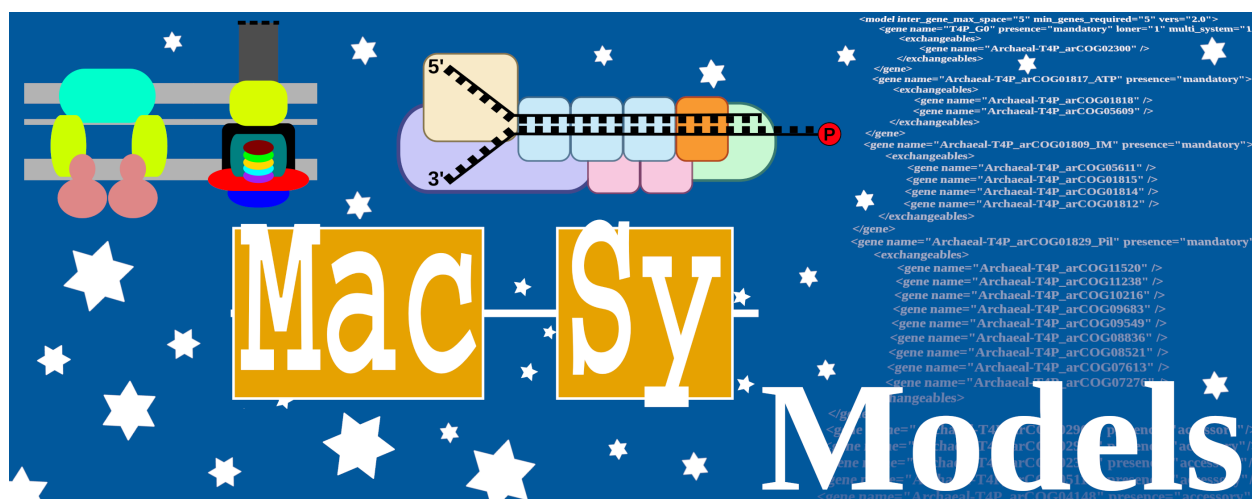
The precedence rules between the different levels of configuration are:

`system < home < model < project < --cfg-file | --previous-run < command line options`

- **system:** the `<macsylib>.conf` file either in `${VIRTUAL_ENV}/etc/<macsylib>/` in case of a *virtualenv* this configuration affects only the MacSyLib version installed in this *virtualenv*
- **home:** the `~/.<macsylib>/<macsylib>.conf` file
- **model:** the `model_conf.xml` file at the root of the model package
- **project:** the `<macsylib>.conf` file found in the directory where the *macsylib* command was run
- **cfgfile:** any configuration file specified by the user on the command line
- **previous-run:** the `<macsylib>.conf` file found in the results directory of the previous run

MODELLER GUIDE

2.1 Modeller Guide



2.1.1 Modelling Systems with MacSyLib

Installation

MacSyLib works with models for macromolecular systems that are not shipped with it, you have to install them separately. See the *msl_data* section below.

MacSyLib Installation procedure

To develop new models and share them, MacSyLib requires *git* and the python library *GitPython*

Below the procedure to install *MacSyLib* in *modeler* mode in a *virtualenv*

```
python3 -m venv macsylib
cd macsylib
source bin/activate
python3 -m pip install macsylib[model]
```

GitPython dependency will be automatically retrieved and installed when using *pip* for installation (see below).

Warning

But you have to install *git* by yourself (using your preferred package manager)

From Conda/Mamba

MacSyLib is packaged for Conda/Mamba. The Conda/Mamba package include modeler dependencies

From container

From version 2.0 and above, a docker image is available. This image allow you to develop models.

Models installation with *msl_data*

Once MacSyLib is installed you have access to an utility program to manage the models: *msl_data*

This script allows to search, download, install and get information from MacSyLib models stored on github (<https://github.com/macsy-models>) or locally installed. The general syntax for *msl_data* is:

```
msl_data <general options> <subcommand> <sub command options> <arguments>
```

To list all models available on *macsy-models*:

```
msl_data available
```

To search for models on *macsy-models*:

```
msl_data search TXSS
```

you can also search in models description:

```
msl_data search -S secretion
```

To install a model package:

```
msl_data install <model name>
```

To install a model when you have not the right to install it system-wide

To install it in your home (*./<macsylib>/data*):

```
msl_data install --user <model name>
```

To install it in any directory:

```
msl_data install --target <model dir> <model_name>
```

To know how to cite a model package:

```
msl_data cite <model name>
```

To show the name of the models and the structure of installed model package:

```
msl_data show <model package name>
```

for instance `msl_data show TXSScan`

```

TXSScan
├--archaea
│   └--Archaeal-T4P
├--bacteria
│   └--diderm
│       ├──Flagellum
│       ├──MSH
│       ├──T1SS
│       ├──T2SS
│       ├──T3SS
│       ├──T4aP
│       ├──T4bP
│       ├──T5aSS
│       ├──T5bSS
│       ├──T5cSS
│       ├──T6SSi
│       ├──T6SSii
│       ├──T6SSiii
│       ├──T9SS
│       ├──Tad
│       ├──pT4SSi
│       └--pT4SSt
└--monoderm
    └--ComM

```

TXSScan (1.1.3) : 19 models

To show the model definition:

```
mssl_data definition <package or subpackage> model1 [model2, ...]
```

for instance to show model definitions T6SSii and T6SSiii in TXSS+/bacterial subpackage:

```
mssl_data definition TXSS+/bacterial T6SSii T6SSiii
```

To show all models definitions in TXSS+/bacterial subpackage:

```
mssl_data definition TXSS+/bacterial
```

To create a git repository with a skeleton for your own model package:

```
mssl_data init --pack-name <MY_PACK_NAME> --maintainer <"mantainer name"> --email
↳<maintainer email> --authors <"author1, author2, ..">
```

above mssl_data with required options. Below I add optional but recommended options.

```
mssl_data init --pack-name <MY_PACK_NAME> --maintainer <"mantainer name"> --email
↳<maintainer email> --authors <"author1, author2, .."> \
--license cc-by-nc-sa --holders <"the copyright holders"> --desc <"one line package_
↳description">
```

To list all *mssl_data* subcommands:

```
msl_data --help
```

To list all available options for a subcommand:

```
msl_data <subcommand> --help
```

For models not stored in *macsy-models* the commands *available*, *search*, *installation* from remote or *upgrade* from remote are **NOT** available.

For models **NOT** stored in *macsy-models*, you have to manage them semi-manually. Download the archive (do not unarchive it), then use *msl_data* to install the archive.

Models Package

MacSyLib relies on the definition of models of macromolecular systems as a **set of models' components** to be searched by similarity search, and a **set of rules** regarding their genomic organization and their requirement level to make a complete system (mandatory, accessory components, number of components required).

See the section *The XML hierarchy* for more details on MacSyLib's modelling scheme and the section on *Functioning* for the principles of the MacSyLib's search engine.

A **MacSyLib model** (*macsy-model* for short) is the association of several elements:

- a **definition** which describes the system to detect with a specific **XML grammar** that is *described here*.
- a set of *HMM profiles* (one per component/gene in the model) to enable the similarity search of the systems' components with the HMMER program.

The models are grouped by *family* possibly gathering *sub-families* (multiple levels allowed), for instance *Secretion*, *Cas-proteins*... A set of models from a same family (coherent set) of systems to detect is called hereafter a **macsy-model package** NEW in V2.

Structure of a macsy-model package

A macsy-model package follows the following structure:

```
family_name
|_____ metadata.yml
|_____ LICENSE
|_____ README.md
|_____ model_conf.xml
|_____ definitions
|           |_____ model_1.xml
|           |_____ model_2.xml
|           :
|
|_____ profiles
|           |_____ geneA.hmm
|           |_____ geneB.hmm
|           |_____ geneC.hmm.gz
```

If the package contains sub-families:

```
family_name
|_____ metadata.yml
|_____ LICENSE
```

(continues on next page)

(continued from previous page)

```

|_____ README.md
|_____ model_conf.xml
|_____ definitions
|           |_____ subfamilyA
|           |           |_____ model_1.xml
|           |           |_____ model_2.xml
|           |_____ subfamilyB
|           |           |_____ model_3.xml
|           |           |_____ model_4.xml
|           :
|_____ profiles
|           |_____ geneA.hmm
|           |_____ geneB.hmm
|           |_____ geneC.hmm.gz

```

For examples of macsy-model packages, please visit <https://github.com/macsy-models>

You can create a template for your package by using *mssl_data init* (formerly *macsydata*). It will create for you:

- the data package directory with the right structure.
- a template of *metadata.yaml* .
- a template of *README.md* file.
- a generic *model_conf.xml* file.
- a LICENSE file if *-license* option is set.
- a COPYRIGHT file if *-holders* option is set.
- a directory *definitions* with an example of model definition (*model_example.xml* to remove before publishing).
- a directory *profiles* where to put the hmm profiles corresponding to the models genes.

Note

MSF can also read *.gz* compressed files; it will uncompress them on the fly. The compressed files must end with the *.gz* extension. For the *hmmsearch* step You need to have *gunzip* installed on your system for this to work.

README.md

A description of the package: what kind of systems the package models, how to use it etc... in *markdown* format. The Readme is displayed to the user on the macsy-models repository on Github. It is also displayed when the user runs *mssl_data help*.

LICENSE

The license is used to protect your work when sharing it. If you don't know which license to choose, have a look at [CreativeCommons](#) This file is optional, but highly recommended.

Metadata file

The *metadata.yml* file contains some meta information about the package itself.

It is in **YAML** format and must have the following structure:

```
---
maintainer:
  name: The name of the person who maintains/to contact for further information.
  ↪(required)
  email: The email of the maintainer (required)
short_desc: A one line description of the package (can e.g. be used for *msl_data*
  ↪searches) (required)
vers: The package version (DEPRECATED)
cite: The publication(s) to cite by the user when the package is used (optional, used by
  ↪`msl_data cite`)
doc: Where to find extended documentation (optional)
license: The license under the package is released (optional but highly recommended)
copyright: The copyright of the package (optional)
```

For example:

```
---
maintainer:
  name: first name last name
  email: login@my_domain.com
short_desc: Models for 15 types of secretion systems or bacterial appendages (T1SS, T2SS,
  ↪ T3SS, T4P, pT4SSi, pT4SSii, T5aSS, T5bSS, T5cSS, T6SSi, T6SSii, T6SSiii, Flagellum,
  ↪ Tad, T9SS).
cite:
  - |
    Abby Sophie S., Cury Jean, Guglielmini Julien, Néron Bertrand, Touchon Marie, Rocha
    ↪ Eduardo P. C. (2016).
    Identification of protein secretion systems in bacterial genomes.
    In Scientific Reports, 6, pp. 23080.
    http://dx.doi.org/10.1038/srep23080
doc: https://github.com/macsy-models/TXSS
license: CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0/)
copyright: 2014-2022, Institut Pasteur, CNRS
```

Note

- - specify an item of yaml list
- | is used to specify a single item but over multiple lines.

Error

This *metadata.yml* file is **mandatory**. Without this file your archive/repository will not be considered as a *macsy-model* package.

Warning

The field *vers* (the package version) is deprecated. *msl_data install* rely only on the git tag.

Model configuration

The modeler has the possibility to specify some options that are specific to their package, different than the MacSyLib defaults in the *model_conf.xml* file. **NEW in v2**

These options can be grouped in two families: the scoring weights and filtering options.

Scoring weights:

- mandatory (*float* default = 1.0)
- accessory (*float* default = 0.5)
- exchangeable (*float* default = 0.8)
- loner_multi_systems (*float* default = 0.7)
- redundancy_penalty (*float* default = 1.5)

Filtering options:

- e_value_search (*float* default = 0.1)
- i_evalue_sel (*float* default = 0.001)
- profile_coverage (*float* default = 0.5)
- cut_ga (*bool* default = True)

All these options are optional and can be omitted in the configuration file, **the file itself is optional**. The precedence rules between the different levels of configuration are:

```
system < home < model < project < --cfg-file | --previous-run < command line options
```

- **system**: the *<macsylib>.conf* file either in */etc/<macsylib>/* or in *\${VIRTUAL_ENV}/etc/<macsylib>/* in case of a *virtualenv* this configuration affects only the MacSyLib version installed in this *virtualenv*
- **home**: the *~/.<macsylib>/<macsylib>.conf* file
- **model**: the *model_conf.xml* file at the root of the model package
- **project**: the *<macsylib>.conf* file found in the directory where the *macsylib* command was run
- **cfgfile**: any configuration file specified by the user on the command line (conflicts with the *-previous-run* option)
- **previous-run**: the *<macsylib>.conf* file found in the results directory of the previous run (conflicts with the *-cfg-file* option)
- **command line**: any option specified directly in the command line

The *model_conf.xml* configuration file is in xml format and must have the following structure:

```
<model_config>
  <weights>
    <mandatory>1</mandatory>
    <accessory>0.5</accessory>
    <exchangeable>0.8</exchangeable>
    <redundancy_penalty>1.5</redundancy_penalty>
```

(continues on next page)

(continued from previous page)

```

    <out_of_cluster>0.7</out_of_cluster>
  </weights>
  <filtering>
    <e_value_search>0.1</e_value_search>
    <i_evalue_sel>0.01</i_evalue_sel>
    <coverage_profile>0.5</coverage_profile>
    <cut_ga>True</cut_ga>
  </filtering>
</model_config>

```

Details about the scoring method can be obtained [here](#).

Macromolecular models

MacSyLib relies on the definition of models of macromolecular systems as a **set of models' components** to be searched by similarity search, and a **set of rules** regarding their genomic organization and their requirement level to make a complete system (mandatory, accessory components, number of components required).

See [below](#) for more details on MacSyLib's modelling scheme and the section on [Functioning](#) for the principles of the MacSyLib's search engine.

A **MacSyLib model** (macy-model for short) is the association of several elements:

- a **definition** which describes the system to detect with a specific **XML grammar** that is described [below](#).
- a set of *HMM profiles* (one per component/gene in the model) to enable the similarity search of the systems' components with the HMMER program.

The models are grouped by *family* possibly gathering *sub-families* (multiple levels allowed), for instance *Secretion*, *Cas-proteins*... A set of models from a same family (coherent set) of systems to detect is called hereafter a **macy-model package** NEW in V2.

Principles, and how to write macy-models definitions

Macy-models are written as XML files, and should be named with the name of the system to detect as a prefix, and the XML file extension as a suffix. For example, 'T1SS.xml' for T1SS (Type I Secretion System).

A macy-model defines a macromolecular System as:

- A set of **components** (*i.e.* proteins, or protein-coding genes given the context) with different attributes that are used for system's **content description**.
- Features regarding the **genomic architecture** of the systems' components for system detection.
- Rules for **quorum** specifying how many components are required to infer the presence of a complete system.

Macy-model Components

Four distinct **types of components** can be used to model the System's content. Components correspond to Gene objects in MacSyLib's implementation, and point to corresponding HMM protein profiles.

- **mandatory** components represent components that are essential to be found to infer the system's presence.
- **accessory** components correspond to components that can be found in some systems' occurrence (or quickly evolving components that are hard to detect with a single HMM profile and thus can be missed along similarity search).
- **neutral** components are used to build/extend clusters of proximal genes/components on the replicon analysed, but are not part of the quorum (*i.e.*, not taken into account to assess the system's presence). NEW in V2

- **forbidden** components are components which presence is eliminatory for the system's presence assessment.

Specifying a genomic organization

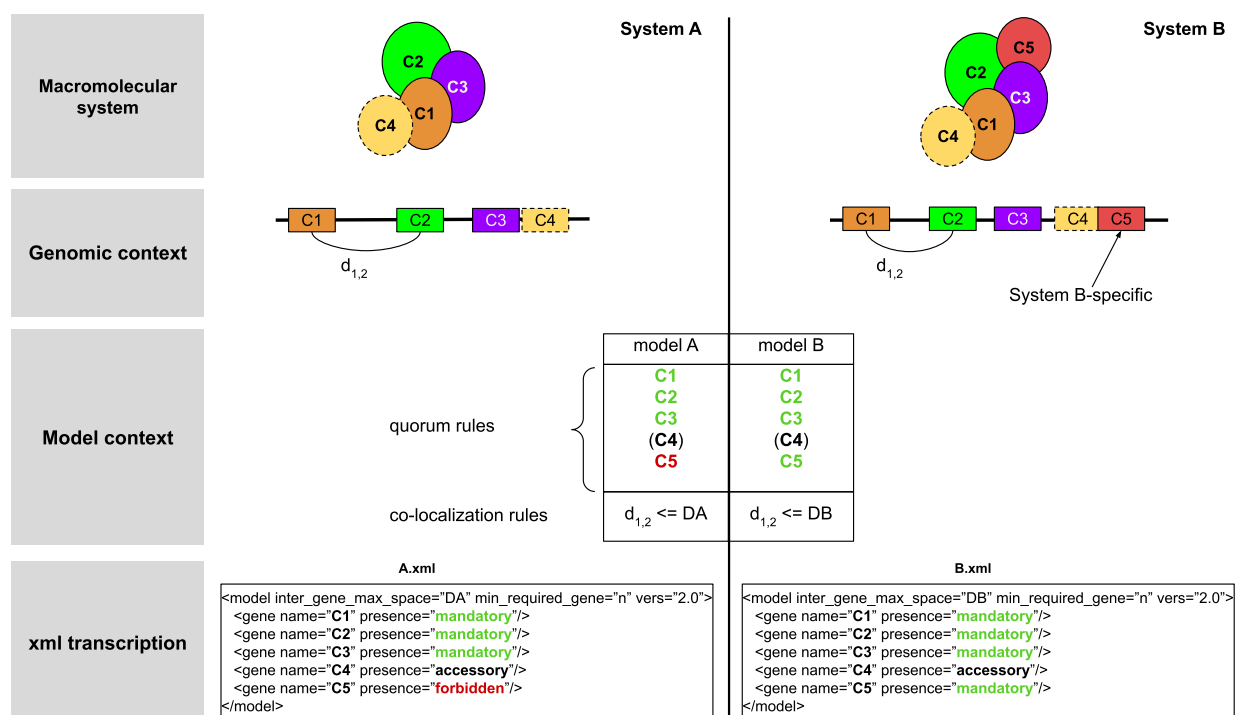
Beyond its list of Components, a MacSyLib's model of a System is defined by the genomic organization of its components. This genomic organization can be defined in several ways:

- the general System's architecture, whether it is *single-locus* or *multi-loci* (encoded at one or several loci)
- the co-localization criteria defined either at the System level or at the Gene (component) level:
 - the *inter-gene-max-space* parameter (system- or gene- wise)
 - the *loner* parameter (gene- wise)

See [below](#) for more details on how to specify these parameters in a macy-model.

The XML hierarchy

A System's model is defined using a specific XML grammar that is hereby described. It consists in a hierarchic view of a Model that has specific features described through parameters, and is made of a set of Genes that have specific features themselves. All these elements and corresponding parameters will parametrize the search of Systems matching the search by MacSyLib, in terms of Gene content and genomic architecture criteria.

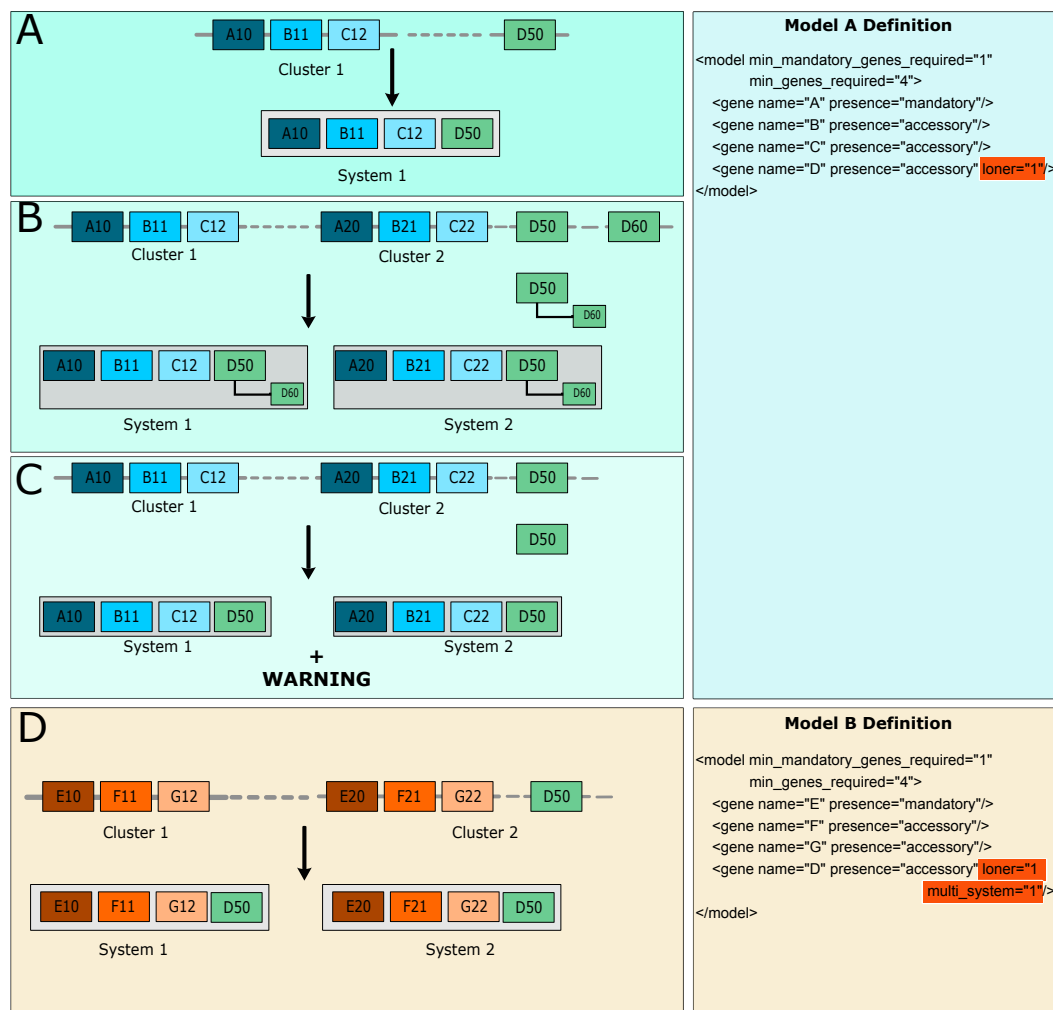


- The element root of a System's model is "model".
 - It has a mandatory attribute: "inter_gene_max_space", an integer representing the maximal number of components without a match between two components with a match for a component profile in order to consider them contiguous (part of a same *Cluster*).
 - The version of the XML grammar (the actual version is "2.0")
 - The element "model" may have attributes:

- * **min_mandatory_genes_required**: an *integer* representing the minimal number of mandatory genes required to infer the system's presence.
 - * **min_genes_required**: an *integer* representing the minimal number of mandatory or accessory genes (whose corresponding proteins match a profile of the model) required to infer the system's presence.
 - * **multi_loci**: a *boolean* set to True ("1", "true" or "True") to allow the definition of "scattered" systems (i.e., systems encoded at different genomic loci or by different gene *clusters*). If not specified, *default value is false*.
 - * **max_nb_genes** define how many genes is necessary to consider a system as full. By default it is the sum of mandatory and accessory genes. But sometimes in special cases, there is 2 profiles, so 2 *msf* genes in model for one real gene. So in system only one gene can be detected and the wholeness is false.
- The model contains one or more element(s) "gene" that correspond(s) to the genetic components of the macromolecular system.
- The element "gene" has several mandatory attributes:
 - **name**: a *string* representing the name of the component/gene which must match that of a profile enclosed in the profile directory of the macsy-model package (see *below*).
 - **presence**: a *string* representing the status of the gene's presence in the system. It can take four values among "mandatory", "accessory", "neutral", "forbidden" (see above).

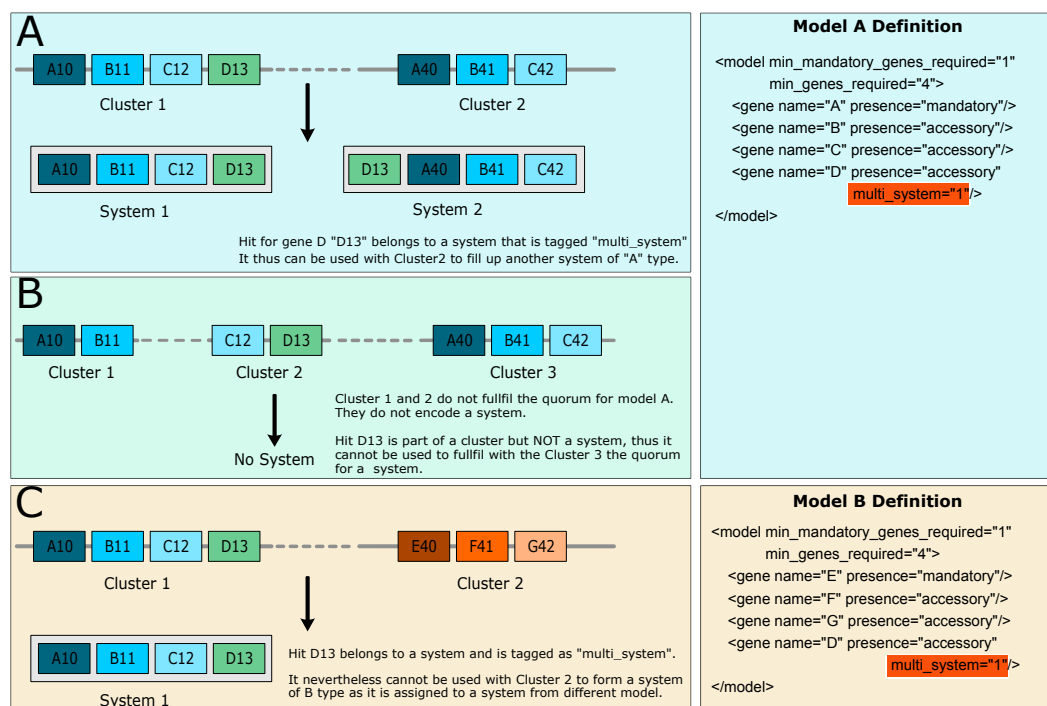
The element "gene" may have other attributes:

- **loner**: a *boolean*. A *loner* gene can be isolated on the genome and does not have to be part of a cluster of genes to be considered for system's assessment (*default false*).
- **multi_system**: a *boolean*. If a gene has the feature "multi_system" (value set to "1", "true" or "True"), it means that it can be used to fill multiple system occurrences (from a same model) - and thus be considered as part of several systems (*default false*).

Fig. 1: How *loner* works.

A) The *cluster 1* can be filled up with the loner *D50* to reach the quorum defined in *model A* and form a system occurrence. **B)** There are 2 clusters and 2 loners (*D50* and *D60*) and *msf* cannot assign which loner goes to which cluster. So *msl* picks the best loner (based on score) and sets the others as “counterpart”. 2 system occurrences are created with the best loner. The user has to choose which loner hit can be assigned to which cluster. All loners found in the best solution are reported in *best_solution_loners.tsv* file. **C)** There are 2 clusters but only 1 loner. *msf* cannot decide to which cluster assign the loner. So the 2 system occurrences are proposed to the user in the output and a warning is raised to indicate the user should pick one. **D)** There are 2 clusters with one loner, but this loner is also *multi_system*. So the 2 clusters can be filled up with the loner.

- **multi_model**: a *boolean*. If a gene has the feature “multi_model” (value set to “1”, “true” or “True”), it means that two systems from different models can coexist in the best solution (they are said “compatible”) even if they share a component. The gene must be tagged as *multi_model* in both model definitions.

Fig. 2: How *multi_system* works.

A) The hit encoding for gene D in position 13 belongs to the system 1 (encoding model A). So it is used to fill up some other cluster, for instance cluster 2, which lacks this functionality. The cluster 2 then also fulfil the requirement of a system. **B)** The hit encoding for gene D in position 13 does not belong to a system. It cannot be used to fill up other clusters. In this example there is no system that satisfies the rules of model A. **C)** The gene D is present in the definition of model A and B. The hit encoding for gene D in position 13 belongs to the system 1 (encoding model A). It cannot be used to fill up the cluster 2 which codes for model B.

- **inter_gene_max_space:** an *integer* that defines gene-wise value of system's "inter_gene_max_space" parameter (see above). It supersedes the system-wise parameter to give the gene a specific co-localization parameter.

The element "gene" may have one "exchangeables" child element:

- The element "exchangeables" can contain one or more elements "gene".

For a Gene to have "exchangeables" Genes listed, means that this Gene can be replaced *in the quorum* by the listed child Genes.

Note

If the attributes *inter_gene_max_space*, *loner*, *multi_model*, *multi_system* are not specified for the exchangeable genes, then they inherit the values from the reference gene. Below some examples of attributes inheritance.

```
<gene name="A" presence="mandatory" multi_model="True">
  <exchangeables>
    <gene name="B" />
    <gene name="C" />
  </exchangeables>
</gene>
```

In the snippet code above, the genes A/B/C are *multi_model* but not *loner* or *multi_system*.

```
<gene name="A" presence="mandatory">
  <exchangeables>
    <gene name="B" multi_model="True"/>
    <gene name="C" />
  </exchangeables>
</gene>
```

In the snippet code above, The gene B is *multi_model* but not A and C.

```
<gene name="A" presence="mandatory" loner="True" multi_system="True">
  <exchangeables>
    <gene name="B" />
    <gene name="C" multi_system="False"/>
  </exchangeables>
</gene>
```

In the snippet code above,

- The genes A/B/C are *loner*
- The genes A and B are *multi_system*, but **not** C.

```
<gene name="A" presence="mandatory" inter_gene_max_space="10">
  <exchangeables>
    <gene name="B" inter_gene_max_space="5"/>
    <gene name="C" />
  </exchangeables>
</gene>
```

In the snippet code above, The genes A and C have an *inter_gene_max_space* = 10 whereas its value is 5 for the gene B .

Warning

The *presence* attribute is inevitably the same for the exchangeable genes than the reference gene.

Note

If not specified by the user, several features will have their values assigned **by default**:

- the **genomic architecture** of the System being searched will consist in a **single locus**. If a System may be made of Genes from multiple loci, consider setting the *multi_loci* parameter to *True*.
- the **quorum parameters** *min_mandatory_genes_required* and *min_genes_required* will be set to the number of mandatory Genes listed - the *accessory* Genes being deemed not required to infer a complete System.

Example of a macy-model definition in XML (more examples in our *gallery of examples*):

```
<model inter_gene_max_space="5" vers="2.0">
  <gene name="gspD" presence="mandatory">
    <exchangeables>
```

(continues on next page)

(continued from previous page)

```

    <gene name="sctC"/>
  </exchangeables>
</gene>
<gene name="sctN_FLG" presence="mandatory" loner="1">
  <exchangeables>
    <gene name="gspE"/>
    <gene name="pilT"/>
  </exchangeables>
</gene>
<gene name="sctV_FLG" presence="mandatory"/>
<gene name="flp" presence="accessory"/>
</model>

```

In this example, the described System consists of three mandatory and one accessory components:

- Two components, the Gene “GspD” and the Gene “sctN_FLG” can respectively be replaced by sctC, and gspE and pilT genes in the quorum.
- To be considered as part of such System, the components should be co-localized in loci (Clusters of Genes), which in this case would amount to being located from each other at a distance of 5-Genes maximum, except for the Gene “sctN_FLG” that is allowed to be located “alone” in the genome being investigated, by a *loner* parameter being set to True. As the *multi_loci* parameter is not set, by default the System should be made of a single locus (Cluster of co-localized Genes - except for the ones listed as *loners*).
- To be considered a complete System, the quorum of Genes should be reached. In this case, the *min_genes_required* and *min_mandatory_genes_required* are not specified and therefore assigned to their default values: *min_mandatory_genes_required* is set to the number of mandatory Genes listed as well as the *min_genes_required* parameter (see above).

Warning

- a gene is identified by its name.
- this name is case sensitive.
- this name must be unique inside a family of models.
- a HMM profile with a gene-based name must exist in the *profiles* directory of the macy-model package (see *below*).

Providing HMM profiles

For each gene mentioned in each model you have to provide a **HMM profile** to enable the similarity search of this gene. The HMM profile must have been created by the user from a curated multiple sequence alignment with the *hmmbuild* program from the [HMMER package](#), or can have been obtained from HMM profiles’ databases such as [TIGRFAM](#) or [PFAM](#).

This profile *MUST* have the same name as the name of the gene mentioned in the definition. For instance, a component named “GeneA” in the macy-model would correspond by default to a HMM profile “GeneA.hmm” enclosed in the macy-model package. The names are **case-sensitive**. All HMM profiles must be placed in the *profiles* directory of the macy-model package.

Warning

For a detailed tutorial on how to define your macsy-model's features, parameters and HMM profiles, you can have a look at our cookbook in [this book chapter](#) .

msl profile

It is ran over a previous <macsylib> analysis:

- it extracts from raw HMMER output files the hits and computes the profile coverage for each of them.
- it enables to filter the hits in a user-defined manner, to test other values of filtering parameters than those used with the MacSyLib run.
- it writes down the results in a file in *tsv* format *hmm_coverage.tsv*.

a)

b)

positional arguments:

optional arguments:

```
-h, --help            show this help message and exit
--coverage-profile COVERAGE_PROFILE
                        Minimal profile coverage required for the hit
                        alignment with the profile to allow the hit selection
```

2.1. Modeller Guide

(continued from previous page)

```

                                for systems detection. (default no threshold)
--i-evalue-sel I_EVALUE_SEL
                                Maximal independent e-value for Hmmer hits to be
                                selected for systems detection. (default: no selection
                                based on i-evalue)
--best-hits {score,i_eval,profile_coverage}
                                If several hits match the same replicon, same gene.
                                Select only the best one (based on best 'score' or
                                'i_evalue' or 'profile_coverage')
-p PATTERN, --pattern PATTERN
                                pattern to filter the hmm files to analyse.
-o OUT, --out OUT
                                the path to a file to write results.
--index-dir INDEX_DIR
                                Specifies the path to a directory to store/read the sequence.
↪index
                                when the sequence-db dir is not writable.
-f, --force
                                force to write output even the file already exists
                                (overwrite it).
-V, --version
                                show program's version number and exit
-v, --verbosity
                                Increases the verbosity level. There are 4 levels:
                                Error messages (default), Warning (-v), Info (-vv) and
                                Debug. (-vvv)
--mute
                                Mute the log on stdout. (continue to log on
                                <macsylib>.log) (default: False)

```

For more details, visit the MacSyLib website and see the MacSyLib documentation.

For instance:

```
>msl_profile <macsylib>-2021XXXX_XX-XX-XX
```

will analyse the HMMER raw outputs stored in <macsylib>-2021XXXX_XX-XX-XX/hmmer_results directory and the results will be stored in <macsylib>-2021XXXX_XX-XX-XX/hmm_coverage.tsv file

Setting filtering parameters

This helper tool is designed to help the user test the relevance of the HMM profiles used, what filtering parameters for HMMER to be used, and understand why some components might be unexpectedly missing from the MacSyLib results. This can thus help to improve the models - for instance for the genomic location parameters (is a component not found cause it should be listed as a *loner*?).

Therefore by default, the filtering parameters are very loose so that most hits found with HMMER will be reported, even the weakest ones.

However, it is possible to filter hits to be extracted based on the profile coverage with *-coverage-profile* or the i-evalue (*-i-evalue-sel*) to be a bit more stringent.

Also, it is possible to use the *-best-hits* in order to report only the best hit for a given protein sequence when several profiles were matching hit.

Using patterns with “-pattern”

If in *previous_run/hmmer_results* you have the following files:

```
previous_run/hmmer_results/Archaeal-T4P_arCOG11238.search_hmm.out
previous_run/hmmer_results/Archaeal-T4P_arCOG11520.search_hmm.out
previous_run/hmmer_results/Archaeal-T4P_arCOG11777.search_hmm.out
previous_run/hmmer_results/Archaeal-T4P_arCOG11778.search_hmm.out
previous_run/hmmer_results/Archaeal-T4P_arCOG11936.search_hmm.out
previous_run/hmmer_results/Archaeal-T4P_arCOG14515.search_hmm.out
previous_run/hmmer_results/ComM_comC.search_hmm.out
previous_run/hmmer_results/ComM_comEB.search_hmm.out
previous_run/hmmer_results/ComM_comEC.search_hmm.out
previous_run/hmmer_results/ComM_comGA.search_hmm.out
previous_run/hmmer_results/ComM_comGB.search_hmm.out
previous_run/hmmer_results/ComM_comGC.search_hmm.out
previous_run/hmmer_results/ComM_comGD.search_hmm.out
previous_run/hmmer_results/ComM_comGE.search_hmm.out
previous_run/hmmer_results/MSH_mshA.search_hmm.out
previous_run/hmmer_results/MSH_mshB.search_hmm.out
previous_run/hmmer_results/MSH_mshC.search_hmm.out
```

But you are interested only in ComM family genes, you can specify the option `--pattern 'ComM*'` For instance:

```
>msl_profile --pattern 'ComM*' <macsylib>-2021XXXX_XX-XX-XX
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comB.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comC.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comEA.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comEB.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comEC.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGA.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGB.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGC.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGD.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGE.search_hmm.out
found 79 hits
result is in 'macsylib-2021XXXX_XX-XX-XX/hmm_coverage.tsv'
```

Note

The patterns available are the *glob* patterns (the jokers usable with unix *ls* command)

```
>msl_profile --pattern 'ComM_com?C' -f <macsylib>-2021XXXX_XX-XX-XX
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comEC.search_hmm.out
parsing <macsylib>-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGC.search_hmm.out
found 16 hits
result is in '<macsylib>-2021XXXX_XX-XX-XX/hmm_coverage.tsv'
```

Note

<macsylib> can be replace by the name of a program if macsylib is used in higher tool (as *macsyfinder*). Just specify the *prog_name* in *macsylib.config.MacsyDefault* object.

A useful example for modellers?

```
>msl_profile --best-hits i_eval --i-evalue-sel 0.001 --coverage-profile 0.5 -o msf_GCF_
↪003149495.1_ASM314949v1_tff-sf/hmm_coverage_best-hits_ieval_default_filter_MSF.tsv msf_
↪GCF_003149495.1_ASM314949v1_tff-sf
found 221 hits
result is in 'msf_GCF_003149495.1_ASM314949v1_tff-sf/hmm_coverage_best-hits_ieval_
↪default_filter_MSF.tsv'
```

This command line might be useful to macsy-models modellers, as it consists in extracting all relevant hits that are used by the MacSyLib engine to search systems, when using the default parameters:

- the proteins are assigned with their best hits (i-evalue based) when they match several profiles (*--best-hits i_eval* option)
- the default filtering parameters (i-evalue and profile coverage) are used (*--i-evalue-sel* and *--coverage-profile* options)

By using this command line that lists all hits available for MacSyLib to search for systems, one could be interested in comparing this list to the list of hits that end in being assigned to systems (listed e.g. in *best_solution.tsv*). This can help to determine why a component is missing from a system: is it because there are no good hits for it, or is it because it does not comply to the co-localization rules defined in the systems' model?

Parsing msl_profile outputs

The *msl_profile* output is a tabulated separated values (.tsv) files. The first lines which are comments (starting with '#') display the tool version and the complete command line used. Then follow the results. The first line of results is a header line.

```
# msl_profile 2.1.5
# msl_profile --pattern ComM* --coverage-profile 0.5 <macsylib>-20201202_15-17-46/
hit_id replicon_name position_hit hit_sequence_length gene_name i_eval
↪score profile_coverage sequence_coverage begin end
GCF_000006745_021980 GCF_000006745 2198 291 ComM_comC 2.500e-40
↪136.400 0.942 0.708 62 267
GCF_000006745_007650 GCF_000006745 765 253 ComM_comC 9.600e-31
↪105.100 0.937 0.798 43 244
...
```

Note

This file can be easily parsed using the Python [pandas](#) library.

```
import pandas as pd
```

```
systems = pd.read_csv("path/to/hmm_coverage.tsv", sep='\t', comment='#')
```

Warning

The *msl_profile* tool is not compliant with results produced with *macsyfinder v1*. If you get *Cannot find models in conf file XXX*. May be these results have been generated with an old version of *macsyfinder*. Check the configuration file, if *[models]* section contains *models_1 = XXX* *YYY* remove the *_1* from models *models = XXX YYY*

Publishing/sharing models

Writing your own macsy-model package

The whole package structure and the corresponding files are described in the section *Structure of a macsy-model package*. It requires five different types of files to be complete:

- a *metadata.yml* file (mandatory)
- a *README.md* file (mandatory)
- a *LICENSE* file (optional but **HIGHLY** recommended)
- a *model_conf.xml* file (optional)
- macsy-models definition(s) within a *definitions* folder (mandatory)
- HMM profiles within a *profiles* folder (mandatory)

You can create a template for your package by using *msl_data init*. It will create for you:

- the git repository with the data package with the right structure.
- a template of *metadata.yml*.
- a template of *README.md* file.
- a generic *model_conf.xml* file.
- a *LICENSE* file if *-license* option is set.
- a *COPYRIGHT* file if *-holders* option is set.
- a directory *definitions* with an example of model definition (*model_example.xml* to remove before publishing).
- a directory *profiles* where to put the hmm profiles corresponding to the models genes.

Sharing your models

If you want to share your models you can create a *macsy-model package* in your github repository. Several steps are needed to publish your model:

1. Check the **validity** of your package with the *msl_data check* command. You have to run it from within the folder containing your package files. It will report:
 - everything is clear: *msl_data* displays the next step to take to publish the package
 - warning: it means that the package could be improved.

It is better to fix it if you can, but you can also proceed to *Step 2*

 - error: the package is not ready to be published as is. You have to fix the errors before you go to *Step 2*.
2. Create a **tag**, and submit a **pull request** to the <https://github.com/macsy-models> organization. This step is **very important**: without a tag, there is no package. *msl_data check* only tagged packages. It is **Mandatory** to follow a versioning scheme described here:
 - <https://www.python.org/dev/peps/pep-0440/#public-version-identifiers>
 - <https://the-hitchhikers-guide-to-packaging.readthedocs.io/en/latest/specification.html#standard-versioning-schemes>

Important

If your package is in version *2.0.1* the tag must be *2.0.1*. The version or tag must **NOT** start with letter as *v2.0.1* or *my_package-2.0.1*.

Warning

To avoid making an inconsistent model visible by `msl_data install/search` (by pushing a tag), a pre-push hook has been setup in the git repository by `msl_data init` command. If you do not used `msl_data init` to create the model, It is a good idea to set up the hook by yourself.

Check that the hook is well named pre-push and it is executable (`chmod 755 .git/hooks/pre-push`) This script run `msl_data check` if you push a tag and it prevent the push if some error are found.

```
#!/bin/sh

# An example hook script to verify what is about to be pushed. Called by "git
# push" after it has checked the remote status, but before anything has been
# pushed. If this script exits with a non-zero status nothing will be pushed.
#
# This hook is called with the following parameters:
#
# $1 -- Name of the remote to which the push is being done
# $2 -- URL to which the push is being done
#
# If pushing without using a named remote those arguments will be equal.
#
# Information about the commits which are being pushed is supplied as lines to
# the standard input in the form:
#
# <local ref> <local oid> <remote ref> <remote oid>
#
# This script check if you push a tag
# if yes check if the tag match to the version decalred in metadata.yml
# if yes it prevents the push until the tag and the version match
#
# This script is widely inspired from https://gist.github.com/farseerfc/
# →0729c08cd7c82b07000f20105f733b17

remote="$1"
url="$2"

tagref=$(grep -Po 'refs/tags/([^\ ]*)' </dev/stdin | head -n1 | cut -c11- | tr -
→d '[:space:]')

if [[ "$tagref" == "" ]]; then
    ## NOT pushing tag , exit normally
    exit 0
fi

macsydata check
returncode=$?
```

```

if [ $returncode -ne 0 ]; then
    Red='\e[1;31m'
    Green='\e[1;32m'
    Yello='\e[1;33m'
    Clear='\e[0m'
    echo "${Green}To fix errors:${Clear}"
    echo "${Red} 1. remove tag:${Clear} git tag -d ${tagref}"
    echo "${Yello} 2. fix errors above ${Clear}"
    echo "${Yello} 3. run 'macsydata check' until everything is fixed ${Clear}"
    echo "${Green} 4. commit your fix:${clear} git add / git commit "
    echo "${Green} 5. tag again:${Clear} git tag -a ${tagref}"
    echo "${Green} 6. and push:${Clear} git push ${remote} ${tagref}"
fi

exit $returncode

pre-push .

```

3. When your pull request (PR) is accepted, the model package becomes automatically available to the community through the *msl_data* tool.

If you don't want to submit a PR you can provide the tag release tarball (tar.gz) as is to your collaborators. This archive will also be usable with the *msl_data* tool.

Note

The creation of a git repository with the right hooks, skeleton of license, copyrights, metadata, profiles and definitions can be done by *msl_data init* command.

Note

msl_data check checks the syntax of the package, but it does not publish anything. It just warns you if something is wrong with the package. Every model provider should check its own package before publishing it. The package publication is done by the *git push* and the *pull request*.

Examples of *msl_data check* outputs:

Your package is syntactically correct:

```

msl_data check tests/data/models/test_model_package/
Checking 'test_model_package' package structure
Checking 'test_model_package' metadata_path
Checking 'test_model_package' Model definitions
Models Parsing
Definitions are consistent
Checking 'test_model_package' model configuration
There is no model configuration for package test_model_package.
If everyone were like you, I'd be out of business
To push the models in organization:
    cd tests/data/models/test_model_package
Transform the models into a git repository

```

(continues on next page)

(continued from previous page)

```

git init .
git add .
git commit -m 'initial commit'
add a remote repository to host the models
for instance if you want to add the models to 'macsy-models'
git remote add origin https://github.com/macsy-models/
git tag 1.0b2
git push --tags

```

You received some warnings:

```

msl_data check tests/data/models/Model_w_conf/
Checking 'Model_w_conf' package structure
Checking 'Model_w_conf' metadata_path
Checking 'Model_w_conf' Model definitions
Models Parsing
Definitions are consistent
Checking 'Model_w_conf' model configuration
The package 'Model_w_conf' have not any LICENSE file. May be you have not right to use_
↪ it.
The package 'Model_w_conf' have not any README file.
msl_data says: You're only giving me a partial QA payment?
I'll take it this time, but I'm not happy.
I'll be really happy, if you fix warnings above, before to publish these models.

```

You received some errors:

```

msl_data check tests/data/models/TFF-SF/
Checking 'TFF-SF' package structure
The package 'TFF-SF' have no 'metadata.yml'.
Please fix issues above, before publishing these models.
ValueError

```

Gallery of examples of MacSyLib's models

Table of contents of the gallery

- *Getting started with a one-component system: the autotransporter T5SS*
- *A (not-so-)simple example: modelling the T1SS*
- *The case of T3SS and bacterial flagella, or how to distinguish homologous cellular machineries*

Here follows a “gallery” of MacSyLib models we have developed over the years, attempting to describe the reasoning behind the modeling process.

These examples are extracted from published work, see the following references (they include more examples):

- [Abby et al. 2016, *Scientific Reports*](#), for the description of the T1SS, T3SS and T5aSS models (and way more models not discussed here).
- [Abby and Rocha 2012, *PLoS Genetics*](#), for the evolutionary study of the T3SS and the bacterial flagellum, and how were designed the corresponding profiles.

- Denise et al. 2019, *PLoS Biology*, for the description of the T2SS and type IV-filament super-family models.

Getting started with a one-component system: the autotransporter T5aSS

This case is rather straight-forward, as the detection of the autotransporter type V secretion system (T5aSS) relies solely on the detection of a single component. This system indeed encodes both a translocator (outer membrane, pore-forming domain) and a passenger domain (toxin or enzyme) on the same gene.

The translocator domain is the **evolutionarily conserved** part across T5aSS. This family of homologous proteins is gathered in the PFAM protein family [PF03797](#) of “Autotransporter” domains.

We thus downloaded the corresponding pre-computed HMM profile that we named “T5aSS_PF03797.hmm” to enable its search using sequence similarity.

We then wrote the corresponding MacSyLib model in a file **T5aSS.xml**:

```
<model inter_gene_max_space="1" vers="2.0">
  <gene name="T5aSS_PF03797" presence="mandatory"/>
</model>
```

It can be noted that several features do not have to be defined if default values are relevant. In particular, in this example it is not needed to specify the quorum parameters: the default value for the minimal number of genes required to infer the presence of the T5aSS is by default the number of components listed in the definition of the system (1).

A (not-so-)simple example: modelling the T1SS

1. Identifying genetic components

The type I secretion system (T1SS) consists in three conserved components:

- an ABC transporter (ABC)
- a membrane-fusion protein (MFP)
- an outer membrane protein (OMF)

For their detection, we therefore need to provide HMM profiles for each component, for example: “abc.hmm”, “mfp.hmm” and “omf.hmm”. These can be specifically designed, or taken from HMM profiles databanks such as [PFAM](#), [TIGRFAM](#) or [SUPERFAMILY](#)..

Note

For suggestions on how to design specific HMM protein profiles, read our dedicated book chapter:

Identification of Protein Secretion Systems in Bacterial Genomes Using MacSyFinder by Sophie Abby and Eduardo Rocha, in *Methods in Molecular Biology* (2017).

2. Determining the role of the components

From literature, the three components listed above *must* be present to have a viable T1SS. Therefore, these are all deemed *mandatory* in the model of the T1SS.

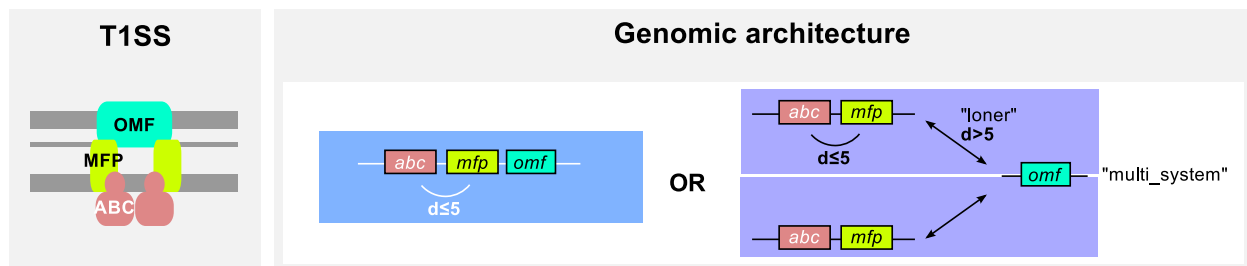
3. Describing their genetic architecture

According to the literature, the genes encoding the three components listed above are generally found lying next to each other in genomes. Therefore, these are considered as “single-locus” system. In addition, there is the particular case of the OMF component. It can either be found:

- next to the two other components, as explained just below
- in some other cases, it can be involved in other cellular machineries functioning, and thus be encoded some place else that at the main T1SS’ locus (in this case, made of ABC+MFP).

Therefore, we can attribute the *loner* feature to the OMF component.

In addition to the latter exception described, it means that this OMF component can also be involved in the functioning of not a single, but several machineries at the same time. In practice, this would mean that two full sets of T1SS components can be inferred with a single OMF component found in the genome. This corresponds to the *multi-system* feature.



4. Writing down the model

Now that all elements of the model are listed, the model for the T1SS can be written using the dedicated MacSyLib XML grammar:

```
<model inter_gene_max_space="5" min_mandatory_genes_required="3" min_genes_required="3"
  vers="2.0">
  <gene name="T1SS_abc" presence="mandatory"/>
  <gene name="T1SS_mfp" presence="mandatory"/>
  <gene name="T1SS_omf" presence="mandatory" loner="1" multi_system="1"/>
</model>
```

The case of T3SS and bacterial flagella, or how to distinguish homologous cellular machineries

The type III secretion system (T3SS), involved in proteic effectors secretion into eukaryotic cells) and the bacterial flagellum (involved in motility) are evolutionarily related (Abby and Rocha 2012). This can make their annotation in genomes tricky, if only based on core components that can have homologs in both systems.

However, these machineries also have **specific core components**. With MacSyLib and the *forbidden* feature for components, it is possible to model this, and create models for efficient discrimination between homologous machineries.

For a toy example on how to model similar yet distinct machineries, you can also have a look [here](#).

1. Identifying genetic components and determining their role

The T3SS is partly homologous to the bacterial flagellum: 8 of its 9 core components are homologous to core components of the flagellum. This is explained by the fact that the T3SS is evolutionarily derived from the flagellum (Abby and Rocha 2012). Yet, the T3SS is made of two dozens of components, and the flagellum, more than twice this number of components:

- The flagellum presents specific core components that have no counterpart in the T3SS.
- It is also the case of the T3SS, which has one specific core component: the secretin.

Solely based on the specificity of core components, it is possible to distinguish T3SS from flagella. This can be done by listing the **specific core components** of a given system as *mandatory* in the system, and as *forbidden* in the homologous system.

Then, HMM protein profiles can be specifically designed for these components, or can be retrieved from databases such as [PFAM](#) , [TIGRFAM](#) or [SUPERFAMILY](#).

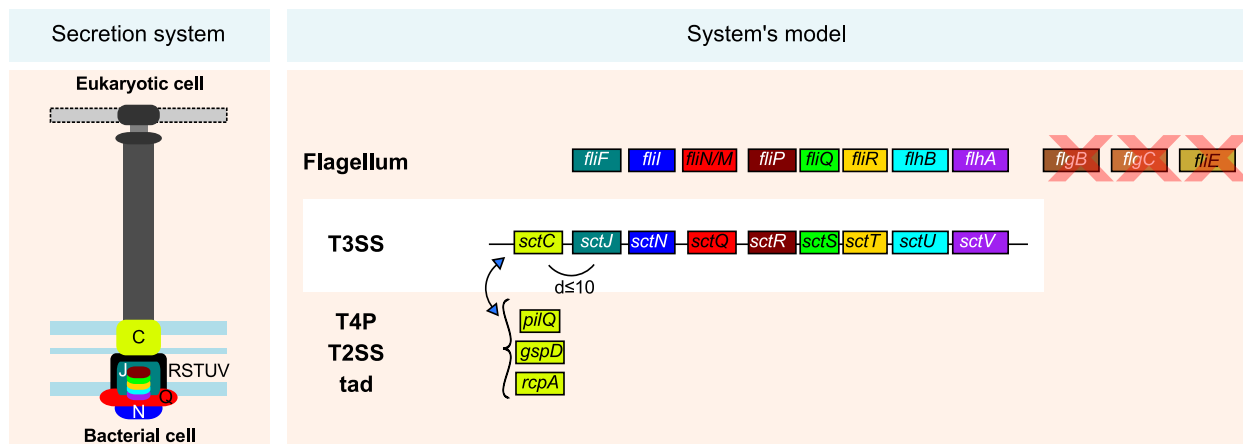
2. Dealing with components with varied evolutionary origins

Another peculiarity of T3SS' evolutionary history consists in that of the secretin, which has been co-opted (acquired) at least three times independently along T3SS diversification: once from the T2SS, once from the Tad pilus, and once from the Type IVa pilus ([Abby and Rocha 2012](#) , [Denise et al. 2019](#)).

This means that sometimes, the T3SS secretin will have more sequence similarity for the secretins from these other machineries - and thus that the profile for the T3SS secretin might “miss” these components, whereas profiles for secretins from the T2SS, T4P or Tad might be more efficient to retrieve them.

Using the *exchangeables* feature, MacSyLib enables to use different HMM protein profile to search for components that may fill a same function. Therefore, it is possible to list profiles of secretins from other machineries among the set of profiles to use to retrieve all T3SS potential secretins.

In the following drawing, a scheme of a T3SS is shown on the left, and the features listed above are shown on a scheme of the T3SS model, including forbidden components from the flagellum (red crosses), and exchangeable components for the secretin “sctC”, depicted with yellow boxes (with the name of the secretin gene from the T4aP, T2SS and Tad pilus respectively). The *inter-gene-max-space* parameter - i.e., maximal number of components allowed between two systems' components to consider them consecutive - is expressed with the “d” letter.



3. Describing the quorum, and genetic architecture of the systems

- T3SS and bacterial flagella are generally encoded on the form of multi-components loci in genomes. Given the fact that we designed HMM protein profiles only for the most conserved, core components of these machineries, and that it means that several systems' components can intersperse between the core ones (remember, T3SS has around 25 components, and the flagellum >40), we set the *inter-gene-max-space* parameter (maximal number of components allowed between two systems' components to consider them consecutive) to 10 in the case of the T3SS, and to 20 in the case of the flagellum.
- T3SS and bacterial flagella can be encoded by one, or multiple loci. We therefore use the *multi-loci* feature to describe their genetic architecture (set to “1”, meaning “True” in the models).

Note

For suggestions on how to set the quorum and genetic architecture parameters, read our dedicated book chapter:

Identification of Protein Secretion Systems in Bacterial Genomes Using MacSyFinder by Sophie Abby and Eduardo Rocha, in *Methods in Molecular Biology* (2017).

4. Writing down the models

Given all the features described above, here is the model of the T3SS:

T3SS.xml

```
<model inter_gene_max_space="10" min_mandatory_genes_required="7" min_genes_required="7"
↳ multi_loci="1" vers="2.0">
  <gene name="T3SS_sctC" presence="mandatory">
    <exchangeables>
      <gene name="T2SS_gspD"/>
      <gene name="T4P_pilQ"/>
      <gene name="Tad_rcpA"/>
    </exchangeables>
  </gene>
  <gene name="T3SS_sctJ" presence="mandatory"/>
  <gene name="T3SS_sctN" presence="mandatory"/>
  <gene name="T3SS_sctQ" presence="mandatory"/>
  <gene name="T3SS_sctR" presence="mandatory"/>
  <gene name="T3SS_sctS" presence="mandatory"/>
  <gene name="T3SS_sctT" presence="mandatory"/>
  <gene name="T3SS_sctU" presence="mandatory"/>
  <gene name="T3SS_sctV" presence="mandatory"/>
  <gene name="Flg_fliE" presence="forbidden"/>
  <gene name="Flg_flgB" presence="forbidden"/>
  <gene name="Flg_flgC" presence="forbidden"/>
</model>
```

And the model of the Flagellum:

Flagellum.xml

```
<model inter_gene_max_space="20" min_mandatory_genes_required="9" min_genes_required="10"
↳ multi_loci="1" vers="2.0">
  <gene name="Flg_sctJ_FLG" presence="mandatory"/>
  <gene name="Flg_sctN_FLG" presence="mandatory"/>
  <gene name="Flg_sctQ_FLG" presence="mandatory"/>
  <gene name="Flg_sctR_FLG" presence="mandatory"/>
  <gene name="Flg_sctS_FLG" presence="mandatory"/>
  <gene name="Flg_sctT_FLG" presence="mandatory"/>
  <gene name="Flg_sctU_FLG" presence="mandatory"/>
  <gene name="Flg_sctV_FLG" presence="mandatory"/>
  <gene name="Flg_flgB" presence="mandatory"/>
  <gene name="Flg_flgC" presence="mandatory"/>
  <gene name="Flg_fliE" presence="mandatory"/>
  <gene name="T3SS_sctC" presence="forbidden"/>
</model>
```

2.1.2 Frequently Asked Questions

Frequently Asked Questions

How to report an issue?

If you encounter a problem while running MacSyLibr, please submit an issue on the dedicated page of the [GitHub project](#)

To ensure we have all elements to help, please provide:

- a concise description of the issue
- the expected behavior VS observed one
- the exact command-line used
- the version of MacSyLib used
- the exact error message, and if applicable, the *macsylib.log* and *macsylib.conf* files
- if applicable, an archive (or link to it) with the output files obtained
- if possible, the smallest dataset there is to reproduce the issue
- if applicable, this would also include the macsy-models (XML models plus HMM profiles) used (or precise version of the models if there are publicly available). Same as above, if possible, please provide the smallest set possible of models and HMM profiles.

All these will definitely help us to help you! ;-)

How to list several components or HMM profiles for a given function in the model?

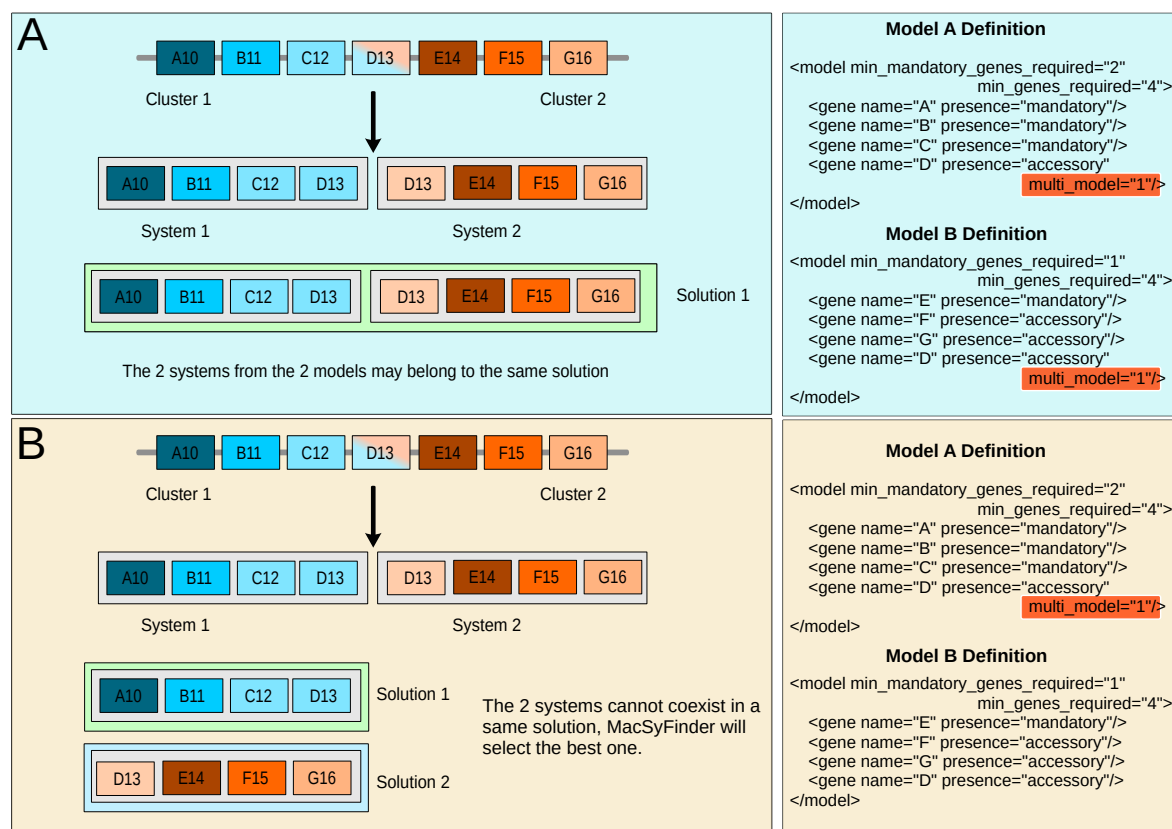
MacSyLib provides a framework to associate a component/function in the model of a system with the mean to search for it - a HMM profile.

In some cases, it is needed to list several possible components (i.e. HMM profiles) to assume a given function for the system to model. There can be several reasons for that:

- a biological reason (e.g., two components from two different gene families can assume a same role in the system)
- a methodological reason (it is not possible or difficult to provide a single HMM profile that covers the diversity of the components' sequences to be retrieved).

It is possible to list several possible components for a same role within the system's model using the *exchangeables* keyword.

See [here](#) for more details and examples.

Fig. 3: How *multi_model* works.

The hit encoding for gene D in position 13 is part of 2 systems: one for Model A, one for Model B. **A)** In both model definitions the gene D is tagged as *multi_model*. So the 2 systems can coexist in a same solution (they are “compatible”). **B)** The gene D is tagged as *multi_model* **only** in model A definition. The 2 systems are not compatible. So *msf* build 2 solutions and choose the best one. It has to be noted that this behaviour would actually be the same if gene D was not declared *multi_model* in either definitions.

DEVELOPER GUIDE

3.1 Developer Guide

3.1.1 Installation

MacSyLib works with models for macromolecular systems that are not shipped with it, you have to install them separately. See the *msl_data* section below. (msl_data is packaged alongside macsylib)

MacSyLib dependencies

Python version ≥ 3.10 is required to run MacSyLib: <https://docs.python.org/3.10/index.html>

MacSyLib has one program dependency:

- the *Hmmer* program, version 3.1 or greater (<http://hmmer.org/>).

The *hmmsearch* program should be installed (*e.g.*, in the PATH) in order to use MacSyLib. Otherwise, the paths to this executable must be specified in the command-line: see the *command-line options*.

MacSyLib also relies on some Python library dependencies:

- colorlog
- colorama
- pyyaml
- packaging
- networkx
- pandas

For modeler (models creator)

- GitPython

For developer

- sphinx
- sphinx_rtd_theme
- sphinx-autodoc-typehints
- sphinxcontrib-svg2pdfconverter
- coverage
- build
- ruff

- pre-commit

These dependencies will be automatically retrieved and installed when using *pip* for installation (see below).

MacSyLib Installation and testing procedures

Installation steps:

Make sure every required dependency/software is present.

By default MacSyLib will try to use *hmmsearch* in your PATH. If *hmmsearch* is not in the PATH, you have to set the absolute path to *hmmsearch* in a *configuration file* or in the *command-line* upon execution. If the tools are not in the path, some test will be skipped and a warning will be raised.

installation in a virtualenv

```
# create a new virtualenv
python3 -m venv MacSyLib
# activate it
cd MacSyLib
source bin/activate
# clone/install the project in editable mode
git clone
cd MacSyLib
python3 -m pip install -e .[dev]
# install tools to ensure coding style
pre-commit install
```

To exit the virtualenv just execute the *deactivate* command.

```
source MacSyLib/bin/activate
```

Then use *MacSyLib* *How to use MacSyLib ?* or installed models with *msl_data* tool.

Note

MacSyLib has adopted *ruff* as linter and *pre-commit* to ensure the coding style. Please read [CONTRIBUTING.md](#) guide lines.

Testing

MacSyLib project use *unittest* framework (included in the standard library) to test the code.

All tests stuff is in *tests* directory.

- The data directory contains data needed by the tests
- in the *__init__.py* file a *MacsyTest* class is defined and should be the base of all testcase use in the project
- each *test_.py** represent a file containing unit tests.

To run all the tests (in the virtualenv)

```
python -m unittest discover
```

To increase verbosity of output

```
python -m unittest discover -vv
```

```
...
test_average_wholeness (tests.test_solution.SolutionTest.test_average_wholeness) ... ok
test_gt (tests.test_solution.SolutionTest.test_gt) ... ok
test_hits_number (tests.test_solution.SolutionTest.test_hits_number) ... ok
test_hits_positions (tests.test_solution.SolutionTest.test_hits_positions) ... ok
test_iteration (tests.test_solution.SolutionTest.test_iteration) ... ok
test_lt (tests.test_solution.SolutionTest.test_lt) ... ok
test_score (tests.test_solution.SolutionTest.test_score) ... ok
test_sorting (tests.test_solution.SolutionTest.test_sorting) ... ok
test_systems (tests.test_solution.SolutionTest.test_systems) ... ok
test_get_def_to_detect (tests.test_utils.TestUtils.test_get_def_to_detect) ... ok
test_get_replicon_names_bad_type (tests.test_utils.TestUtils.test_get_replicon_names_bad_
↳ type) ... ok
test_get_replicon_names_gembase (tests.test_utils.TestUtils.test_get_replicon_names_
↳ gembase) ... ok
test_get_replicon_names_ordered (tests.test_utils.TestUtils.test_get_replicon_names_
↳ ordered) ... ok
test_get_replicon_names_unordered (tests.test_utils.TestUtils.test_get_replicon_names_
↳ unordered) ... ok
test_parse_time (tests.test_utils.TestUtils.test_parse_time) ... ok
test_threads_available (tests.test_utils.TestUtils.test_threads_available) ... ok

-----
Ran 548 tests in 34.265s

OK
```

The tests must be in python file (.py) starting with `test_`. It's possible to specify one or several test files, one module, or one class in a module or a method in a Test class.

Test the `test_package` module

```
python -m unittest -vv tests.test_package
```

```
test_init (tests.test_package.TestLocalModelIndex.test_init) ... ok
test_repos_url (tests.test_package.TestLocalModelIndex.test_repos_url) ... ok
test_check (tests.test_package.TestPackage.test_check) ... ok
test_check_bad_metadata (tests.test_package.TestPackage.test_check_bad_metadata) ... ok
test_check_dir_in_profile (tests.test_package.TestPackage.test_check_dir_in_profile) ...
↳ ok

...

test_list_package_vers (tests.test_package.TestRemoteModelIndex.test_list_package_vers) .
↳ ... ok
test_list_packages (tests.test_package.TestRemoteModelIndex.test_list_packages) ... ok
test_remote_exists (tests.test_package.TestRemoteModelIndex.test_remote_exists) ... ok
test_repos_url (tests.test_package.TestRemoteModelIndex.test_repos_url) ... ok
test_unarchive (tests.test_package.TestRemoteModelIndex.test_unarchive) ... ok
test_url_json (tests.test_package.TestRemoteModelIndex.test_url_json) ... ok
test_url_json_reach_limit (tests.test_package.TestRemoteModelIndex.test_url_json_reach_
```

(continues on next page)

(continued from previous page)

```
↪limit) ... ok
```

```
-----  
Ran 56 tests in 0.242s
```

```
OK
```

Test only the class *TestPackage* (this module contains 3 classes)

```
python -m unittest -vv tests.test_package.TestPackage
```

```
test_check (tests.test_package.TestPackage.test_check) ... ok  
test_check_bad_metadata (tests.test_package.TestPackage.test_check_bad_metadata) ... ok  
test_check_dir_in_profile (tests.test_package.TestPackage.test_check_dir_in_profile) ...  
↪ok  
test_check_empty_profile (tests.test_package.TestPackage.test_check_empty_profile) ... ok  
test_check_metadata (tests.test_package.TestPackage.test_check_metadata) ... ok  
test_check_metadata_no_cite (tests.test_package.TestPackage.test_check_metadata_no_cite)..  
↪... ok
```

```
...
```

```
test_metadata (tests.test_package.TestPackage.test_metadata) ... ok  
test_profile_with_bad_ext (tests.test_package.TestPackage.test_profile_with_bad_ext) ...  
↪ok
```

```
-----  
Ran 42 tests in 0.151s
```

```
OK
```

Test only the method *test_metadata* from the test Class *TestPackage* in module *test_package*

```
python -m unittest -vv tests.test_package.TestPackage.test_metadata
```

```
test_metadata (tests.test_package.TestPackage.test_metadata) ... ok
```

```
-----  
Ran 1 test in 0.005s
```

```
OK
```

Coverage

To compute the tests coverage, we use the [coverage](#) package. The package is automatically installed if you have installed *MacSyLib* with the *dev* target see [installation](#) The coverage package is setup in the *pyproject.toml* configuration file

To compute the coverage

```
coverage run
```



```

...
test_lt (tests.test_solution.SolutionTest.test_lt) ... ok
test_score (tests.test_solution.SolutionTest.test_score) ... ok
test_sorting (tests.test_solution.SolutionTest.test_sorting) ... ok
test_systems (tests.test_solution.SolutionTest.test_systems) ... ok
test_get_def_to_detect (tests.test_utils.TestUtils.test_get_def_to_detect) ... ok
test_get_replicon_names_bad_type (tests.test_utils.TestUtils.test_get_replicon_names_bad_
↳type) ... ok
test_get_replicon_names_gembase (tests.test_utils.TestUtils.test_get_replicon_names_
↳gembase) ... ok
test_get_replicon_names_ordered (tests.test_utils.TestUtils.test_get_replicon_names_
↳ordered) ... ok
test_get_replicon_names_unordered (tests.test_utils.TestUtils.test_get_replicon_names_
↳unordered) ... ok
test_parse_time (tests.test_utils.TestUtils.test_parse_time) ... ok
test_threads_available (tests.test_utils.TestUtils.test_threads_available) ... ok

-----
Ran 548 tests in 34.485s

OK

```

Then display a report

```
coverage report
```

Name	Stmts	Miss	Branch	BrPart	Cover
src/macsylib/__init__.py	56	2	12	1	96%
src/macsylib/cluster.py	278	7	114	2	97%
src/macsylib/config.py	391	12	140	7	96%
src/macsylib/data/__init__.py	0	0	0	0	100%
src/macsylib/database.py	203	3	52	1	98%
src/macsylib/definition_parser.py	219	3	70	2	98%
src/macsylib/error.py	8	0	0	0	100%
src/macsylib/gene.py	144	2	18	1	98%
src/macsylib/hit.py	198	1	54	2	99%
src/macsylib/io.py	173	1	76	1	99%
src/macsylib/licenses.py	14	0	2	0	100%
src/macsylib/metadata.py	126	0	36	2	99%
src/macsylib/model.py	127	0	34	0	100%
src/macsylib/model_conf_parser.py	62	0	12	0	100%
src/macsylib/package.py	326	9	110	4	96%
src/macsylib/profile.py	115	7	28	1	94%
src/macsylib/registries.py	189	5	62	6	96%
src/macsylib/report.py	121	0	28	2	99%
src/macsylib/scripts/__init__.py	0	0	0	0	100%
src/macsylib/scripts/macsydata.py	682	57	182	15	91%
src/macsylib/scripts/macsyprofile.py	247	5	64	6	96%
src/macsylib/search_genes.py	79	7	20	3	90%
src/macsylib/search_systems.py	150	7	50	5	94%
src/macsylib/serialization.py	137	3	48	2	97%

(continues on next page)

(continued from previous page)

src/macsylib/solution.py	97	0	34	0	100%
src/macsylib/system.py	397	3	96	0	99%
src/macsylib/utils.py	84	0	24	1	99%

TOTAL	4623	134	1366	64	96%

or generate a html report

```
coverage html
```

```
Wrote HTML report to htmlcov/index.html
```

The results are in the *htmlcov* directory. With you favourite web browser, open the *index.html* file. for more options please refer to the [coverage documentation](#).

3.1.2 MacSyLib implementation overview

MacSyLib is implemented with an object-oriented architecture. Below a short glossary to fix the vocabulary used in MacSyLib.

Cluster

Is a “contiguous” set of hits. two hits are considered contiguous if the number of genes between the 2 genes matching the 2 hits in the replicon is lesser than inter-genes-max-space.

Model

Is a formal description of a macromolecular system. Is composed of a definition and a list of profiles. at each gene of the Model must correspond a profile

Model family

A set of models, on the same topic. It is composed of several definitions which can be sorted in hierachical structure and profiles. A profile is a hmm profile file.

ModelDefinition

Is a definition of model, it's serialize as a xml file

Solution

It's a systems combination for one replicon. The best solution for a replicon, is the combination of all systems found in this replicon which maximize the score.

System

It's an occurrence of a specific Model on a replicon. Basically, it's a cluster or set of clusters which satisfy the Model quorum.

MacSyLib project structure

A brief overview of the files and directory constituting the MacSyLib project.

doc

The project is documented using sphinx. All sources files needed to generate this documentation is in the directory *doc*

macsylib

This the MacSyLib python library Inside macsypy there is a subdirectory *scripts* which are the entry points for *macsyprofile* (tool to help modeler to analyse *hmmsearch* output) and *macsydata* (tool to install models)

tests

The code is tested using *unittests*. In *tests* the directory *data* contains all data needed to perform the tests.

CONTRIBUTORS

A file containing the list of code contributors.

CONTRIBUTING

A guide on how to contribute to the project.

COPYRIGHT

The MacSyLib copyrights.

COPYING

The licencing. MacSyLib is released under GPLv3.

README.md

Brief information about the project.

setup.py

The installation recipe.

pyproject.toml

The project information and installation recipe.

codemeta.json

metadata on this project. (<https://codemeta.github.io/>)

MacSyLib architecture overview

An overview of the main classes.

Note

use *view image* of your browser to zoom in the diagram

search_system functioning overview

In this section I'll give you an idea of the `macsylib.search_systems.search_systems()` functioning at very high grain coarse.

The higher level function is `macsylib.search_systems.search_systems()`. But to call this function you have to create a `macsylib.config.Config` object (*configuration*) and also initialize the logger.

The first *search_systems* task is to create models asked by the user. So a `macsylib.definition_parser.DefinitionParser` is instantiated and the `macsylib.model.ModelBank` and `macsylib.gene.GeneBank` are populated.

Once all models are created, we gather all genes and search them in the replicons. This step is done in parallel. The search is done by profile object associated to each gene and rely on the external software *hmmsearch*. The parallelization is ensure by *search_genes* function The results of this step is a list of hits.

This list is sorted by position and score. this list is filtered to keep only one hit for each position, the one with the best score (position is a gene product in a replicon)

For each model asked by the user, we filter the hits list to keep only those related to the model. Those which are link to *mandatory*, *accessory*, *neutral* or *forbidden* genes included the exchangeables.

This hits are clustered based on distance constraints describe in the models:

- **inter_gene_max_space** : the maximum genes allowed between to genes of a system.
- **loner** : allow a gene to participate to system even if it does not clusterize with some other genes.

Then we check if each cluster satisfy the quorum described in the model.

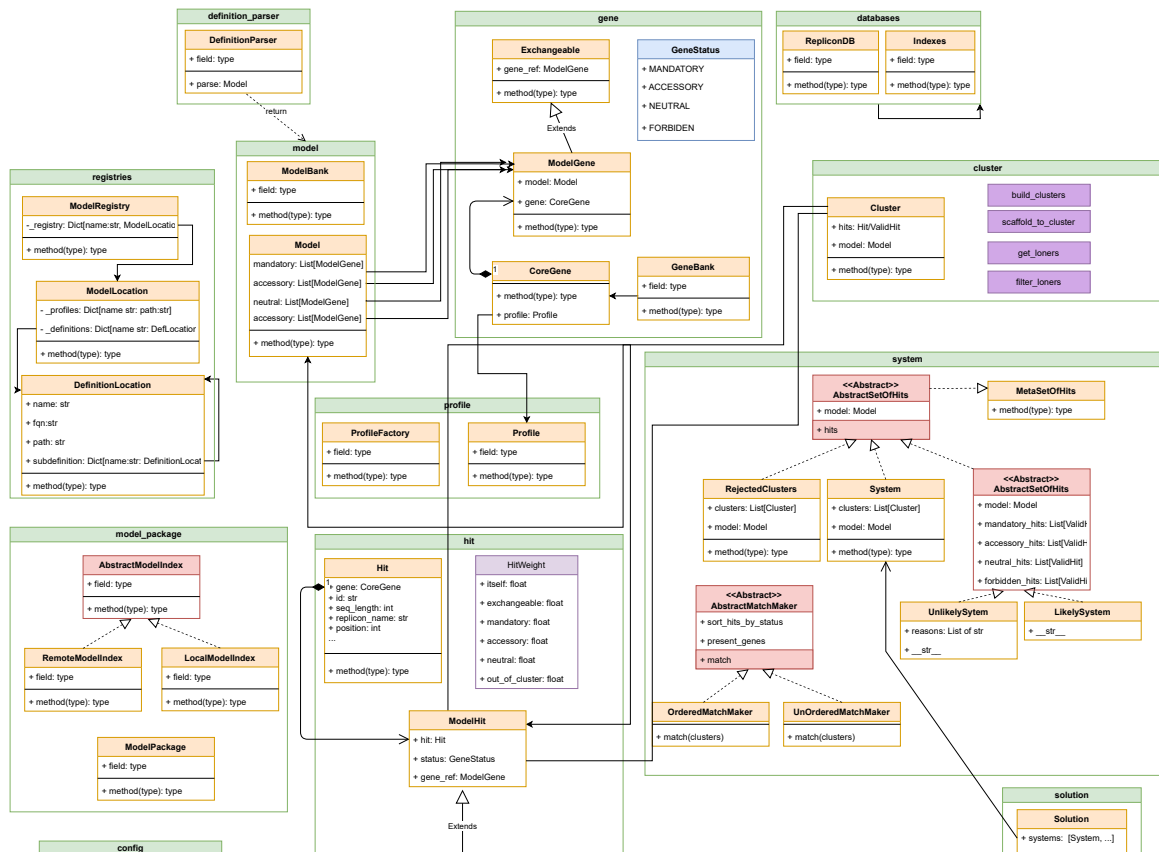


Fig. 1: The MacSyLib classes diagram. The classes are not details. only the main attributes allowing us to understand the interaction are mentioned.

- in green the modules
- in orange, the concrete class
- in red the abstract classes
- in blue the enumeration
- in purple the dataclass
- in purple/pink functions

- **min_mandatory_genes** : the minimum of mandatory genes requisite to have a system.
- **min_genes_required** : the minimum of genes (mandatory + accessory) requisite to have a system.
- **forbidden_genes** : no forbidden genes may appear in the cluster.

If the model is *multi_loci* we generate a combination of the clusters and check the quorum for each combination. If the cluster or combination satisfy the quorum a `macsypy.systems.System` is created otherwise a `macsypy.cluster.RejectedCandidate`.

The Systems from the same replicon are sort against their position, score.

Note

The *neutral* genes are used to build clusters. But not to fulfill the quorum.

Among all this potential systems, MSL (MacSyLib) compute the best combination. `macsylib.solution.find_best_solutions()`. The best combination is the set of compatible systems (do not share common hits) which maximize the score. It's possible to have several equivalent "best solutions". The results of this step is reported in the *best_systems.tsv* file.

The Model object

The *Model object* represents a macromolecular model to detect. It is defined *via* a definition file in XML stored in a dedicated location that can be specified *via* the `macsylib.config.Config` object. See *The XML hierarchy* for more details on the XML grammar.

An object *ModelDefinitionParser* is used to build a model object from its XML definition file.

A model is named after the file tree name of its XML definition. A model has an attribute *inter_gene_max_space* which is an integer, and four kind of components are listed in function of their presence in the system:

- The genes that must be present in the genome to define this model ("mandatory").
- The genes that can be present, but do not have to be found in every case ("accessory").
- The genes that are used to build clusters, but not take in account to check the quorum (*min-genes-required* and *min-mandatory-genes-required*) are described as "neutral".
- The genes that must not be present in the system ("forbidden").

Note

A complete description of macromolecular models modelling is available in the section *Macromolecular models*

The Gene object

The *Gene object* represents genes encoding the protein components of a Model. There is 2 kind of gene The *CoreGene* (`macsylib.gene.CoreGene`) which must be unique given a name. A *CoreGene* must have a corresponding HMM protein profile. These profiles are represented by Profile objects (`macsylib.profile.Profile`), and must be named after the gene name. For instance, the gene *gspD* will correspond to the "gspD.hmm" profile file. See *The Profile object*). After *hmmsearch* step the hits are link the them. The `macsylib.gene.CoreGene` objects must be created by using the `macsylib.gene.GeneBank` factory.

A *ModelGene* (`macsylib.gene.ModelGene`) which encapsulate a *CoreGene* and is linked to a Model. Instead *CoreGene*, several *ModelGene* with the same name may coexists in one macsylib run, in different Models and hold dif-

ferent values for attributes as *inter_gene_max_space*, ... Each ModelGene points out its Model of origin (*macsylib.model.Model*). A Gene has several properties described in the *Gene API*.

A ModelGene may be functionally replaced by an other (usually Homologs or Analogs). In this case these genes are described as exchangeables. Exchangeable object encapsulates a ModelGene and has a reference to the ModelGene it is exchangeable to. See the *Exchangeable API* for more details.

Warning

To optimize computation and to avoid concurrency problems when we search several Models, each CoreGene must be instantiated only once, and stored in a “gene_bank”. *gene_bank* is a *macsylib.gene.GeneBank* object. The *gene_bank* and *model_bank* are filled by the *system_parser* (*macsylib.definition_parser.ModelDefinitionParser*)

The Profile object

Each “CoreGene” component corresponds to a “Profile”. The *macsylib.profile.Profile* object is used for the search of the gene with *Hmmer*. Thus, a “Profile” must match a HMM file, which name is based on the profile name. For instance, the *gspG* gene has the corresponding “gspG.hmm” profile file provided at a dedicated location.

Reporting Hmmer search results

A “HMMReport” (*macsylib.report.HMMReport*) object represents the results of a Hmmer program search on the input dataset with a hidden Markov model protein profile. This object has methods to extract and build “Hits” that are then analyzed for systems assessment.

It analyses Hmmer raw outputs, and applies filters on the matches (according to *Hmmer options*). See *Hmmer results’ output files* for details on the resulting output files. For profile matches selected with the filtering parameters, “Hit” objects are built (see *the Hit API*).

3.1.3 MacSyLib API documentation

configuration

Options to run MacSyLib can be specified in a *Configuration file*. The API described below handles all configuration options for MacSyLib.

The *macsypy.config.MacsyDefaults* hold the default values for *macsylib* whereas the *macsypy.config.Config* hold the values for a *macsylib* run.

configuration API reference

MacsyDefaults

Hold the default values for *macsylib*

```
class macsylib.config.MacsyDefaults(tool_name: str | None = None, pack_name: str = 'macsylib',
                                    **kwargs)
```

Handle all default values for *macsylib*. the default values must be defined here, **NOT** in argument parser nor in config the argument parser or config must use a *MacsyDefaults* object

```
__init__(tool_name: str | None = None, pack_name: str = 'macsylib', **kwargs) → None
```

Parameters

kwargs – allow to overwrite a default value. It mainly used in unit tests

To define a new default value just add an attribute with the default value

__weakref__

list of weak references to the object

Config

Hold the values for this *macsylib* run

class `macsylib.config.Config`(defaults: `MacsyDefaults`, parsed_args: `Namespace`)

Handle configuration values for macsylib. This values come from default and are superseded by the configuration files, then the command line settings.

__init__(defaults: `MacsyDefaults`, parsed_args: `Namespace`) → None

Store macsylib configuration options and propose an interface to access to them.

The config object is populated in several steps, the rules of precedence are

system-wide conf < user home conf < model conf < (project conf | previous run) < command line

system-wide conf = etc/<program name>/<program name>.conf user home conf = ~/.<program name>/<program name>.conf model conf = model_conf.xml at the root of the model package project conf = <program name>.conf where the analysis is run previous run = <program name>.conf in previous run results dir command line = the options set on the command line

Parameters

- **defaults**
- **parsed_args** – the command line arguments parsed

__weakref__

list of weak references to the object

_config_file_2_dict(file: `str`) → dict[slice(<class 'str'>, typing.Any, None)]

Parse a configuration file in ini format in dictionary

Parameters

file – path to the configuration file

Returns

the parsed options

_set_command_line_config(parsed_args: `Namespace`) → None

Parameters

parsed_args – the argument set on the command line

_set_db_type(value: `Literal['gembase', 'ordered_replicon', 'unordered']`) → None

set value for 'db_type' option

Parameters

value – the value for db_type, allowed values are : 'ordered_replicon', 'gembase', 'unordered'

Raises

ValueError – if value is not allowed

_set_default_config() → None

set the value coming from MacsyDefaults

_set_inter_gene_max_space(*value: str*) → None

set value for 'inter_gene_max_space' option

Parameters

value – the string parse representing the model fully qualified name and it's associated value and so on the model_fqn is a string, the associated value must be cast in int

Raises

ValueError – if value is not well-formed

_set_log_level(*value: str*) → None

Parameters

value

_set_max_nb_genes(*value: str | Iterable[tuple[str, int]]*) → None

set value for 'max_nb_genes' option

Parameters

value (str) – the string parse representing the model fully qualified name and it's associated value and so on the model_fqn is a string, the associated value must be cast in int

Raises

ValueError – if value is not well-formed

_set_min_genes_required(*value: str*) → None

set value for 'min_genes_required' option

Parameters

value – the string parse representing the model fully qualified name and it's associated value and so on the model_fqn is a string, the associated value must be cast in int

Raises

ValueError – if value is not well-formed

_set_min_mandatory_genes_required(*value: str | Iterable[tuple[str, int]]*) → None

set value for 'min_mandatory_genes_required' option

Parameters

value – the string parse representing the model fully qualified name and it's associated value and so on the model_fqn is a string, the associated value must be cast in int

Raises

ValueError – if value is not well-formed

_set_model_config(*model_conf_path: str*) → None

Set the options from the model package model_conf.xml file

Parameters

model_conf_path – The path to the model_conf.xml file

_set_models(*value: str | list[str, list[str]]*) → None

Parameters

value – The models to search as return by the command line parsing or the configuration files

if value come from command_line

['model1', 'def1', 'def2', 'def3']

if value come from config file

['set_1', 'T9SS T3SS T4SS_typeI'] [(model_family, [def_name1, ...]), ...]

_set_models_dir(*path: str*) → None

Parameters

path (*str*) – the path to the models (definitions + profiles) are stored.

_set_multi_loci(*value: str*) → None

Parameters

value (*str*) – the models fqcn list separated by comma of multi loc models

_set_no_cut_ga(*value*) → None

Parameters

value

Returns

Return type

_set_options(*options: dict[slice(<class 'str'>, typing.Any, None)]*) → None

set key, value in the general config

Parameters

options (a dictionary with option name as keys and values as values) – the options to specify in general config

_set_previous_run_config(*prev_config_path: str*) → None

Set the options specified by the user on the command line via `–previous-run`

Parameters

prev_config_path

_set_project_config_file(*config_path: str*) → None

Set the options from the `<program_name>.conf` present in the current directory

Parameters

config_path – the path to the configuration file

_set_replicon_topology(*value: Literal['linear', 'circular']*) → None

set the default replicon topology

Parameters

value – ‘circular’ or ‘linear’

_set_sequence_db(*path: str*) → None

Parameters

path – set the path to the sequence file (in fasta format) to analysed

_set_system_models_dir(*value*)

Parameters

value (*list of string or a single string*) – the path of the models dir set by the system (vs set by the user)

Returns

_set_system_wide_config(*config_path: str*) → None

set the options from the system-wide configuration file

Parameters

config_path

_set_topology_file(*path: str*) → None

test if the path exists and set it in config

Parameters

path – the path to the topology file

_set_user_config_file(*config_path: str*) → None

Set the options specified by the user on the command line via the `-cfg-file` option

Parameters

config_path – The path to the configuration path

_set_user_wide_config(*config_path: str*) → None

Set the options from the `~/<program name>/<program name>.conf` file

Parameters

config_path – The path to the `~/<program name>/<program name>.conf`

_str_2_tuple(*value: str*) → list[tuple[str, str]]

transform a string with syntax `{model_fqn int}` in list of tuple

Parameters

value (*str*) – the string to parse

Returns

[(*model_fqn*, *value as str*), ...]

hit_weights() → dict[slice(<class 'str'>, <class 'float'>, None)]

Returns

the options used in scoring systems (*mandatory_weight*, *accessory_weight*, *itself_weight*, *exchangeable_weight*, *out_of_cluster_weight*)

Return type

dict

hammer_dir() → str

Returns

The name of the directory containing the `hmmsearch` results (*output*, *error*, *parsing*)

inter_gene_max_space(*model_fqn: str*) → int | None

Parameters

model_fqn – the model fully qualified name

Returns

the *gene_max_space* for the *model_fqn* or None if it's does not specify

log_level() → int

Returns

the verbosity output level

Return type

int

max_nb_genes(*model_fqn: str*) → int | None

Parameters

model_fqn – the model fully qualified name

Returns

the max_nb_genes for the model_fqn or None if it's does not specify

min_genes_required(model_fqn: str) → int | None

Parameters

model_fqn – the model fully qualified name

Returns

the min_genes_required for the model_fqn or None if it's does not specify

min_mandatory_genes_required(model_fqn: str) → int | None

Parameters

model_fqn – the model fully qualified name

Returns

the min_mandatory_genes_required for the model_fqn or None if it's does not specify

models_dir() → str | None

Returns

list of models dir path

Return type

list of str

multi_loci(model_fqn: str) → bool

Parameters

model_fqn (str) – the model fully qualified name

Returns

True if the model is multi loci, False otherwise

Return type

bool

out_dir() → str

Returns

the path to the directory where the results are stored

save(path_or_buf: str | TextIO | None = None) → None

save itself in a file in ini format.

Note

the undefined options (set to None) are omitted

Parameters

path_or_buf (str or file like object) – where to serialize itself.

working_dir() → str

alias to config.Config.out_dir()

NoneConfig

Minimalist Config object just use in some special case where config is required by api but not used for instance in `macsylib.package.Package`

class `macsylib.config.NoneConfig`

Minimalist Config object just use in some special case where config is required by api but not used for instance in `macsylib.package.ModelPackage`

__weakref__

list of weak references to the object

model_conf_parser

The parser of xml file `model_conf.xml` located at the root of the model package. This file is optional in package

model_conf_parser API reference

ModelConfParser

class `macsylib.model_conf_parser.ModelConfParser`(*path: str*)

Handle `model_conf.xml` configuration file.

__init__(*path: str*) → None

Parameters

path (*str*) – The path to the configuration file

__weakref__

list of weak references to the object

_get_model_conf_node() → ElementTree

Find the root of the document

Returns

the document root of `model_conf`

_parse_section(*section_node: ~xml.etree.ElementTree.ElementTree, allowed_elements: dict[slice(<class 'str'>, typing.Callable, None)]*) → dict[slice(<class 'str'>, typing.Any, None)]

Parse a node containing configurations options and value

Parameters

- **section_node**
- **allowed_elements** (*a dict with options name as keys and function to parse the element*) – The elements allowed in this section Only these elements are parsed and in the final dictionary

Returns

dict

parse() → dict[slice(<class 'str'>, typing.Any, None)]

Parse the xml 'model_conf' file set at the root of a data package

Returns

The specific configuration for a model family

Return type

dict with the name of variables as keys and value as values

parse_filtering(*filtering_node: ElementTree*) → dict[slice(<class 'str'>, typing.Any, None)]

Parse the node 'filtering' containing the filtering options configuration

Parameters

filtering_node – the node 'filtering'

Returns

the configuration option/value about the filtering

parse_weights(*weights_node: ElementTree*) → dict[slice(<class 'str'>, <class 'float'>, None)]

Parse the node 'weights' containing the scoring weight configuration

Parameters

weights_node – the node 'weights'

Returns

the configuration option/value about the scores

registries

The registry manage the different location where *macsylib* can find models definitions and their associated profiles.

registries API reference

ModelRegistry

class macsylib.registries.**ModelRegistry**

scan canonical directories to register the different models available in global <program name> share data location (depending on installation /usr/share/data/models) or can be overloaded with the location specify in the <program name | 'macsylib'> configuration (either in config file or command line)

__getitem__(*name: str*) → *ModelLocation*

Parameters

name

Returns

the model corresponding to name.

Raises

KeyError – if name does not match any ModelLocation registered.

__init__() → None

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

add(*model_loc: ModelLocation*) → None

Parameters

model_loc – the model location to add to the registry

models() → list[*ModelLocation*]

Returns

the list of models

ModelLocation

```
class macsylib.registries.ModelLocation(path: str = None, profile_suffix: str = '.hmm', relative_path:
                                         bool = False)
```

Handle where are store Models. Models are organized in families and subfamilies. each family match to a ModelLocation. a ModelLocation contains the path toward the definitions and the paths to corresponding to the profiles.

```
__eq__(other: ModelLocation) → bool
```

Return self==value.

```
__gt__(other: ModelLocation) → bool
```

Return self>value.

```
__init__(path: str = None, profile_suffix: str = '.hmm', relative_path: bool = False) → None
```

Parameters

- **path** – if it's an installed model, path is the absolute path to a model family.
- **profile_suffix** – the suffix of hmm files
- **relative_path** – True if you want to work with relative path, False to work with absolute path.

```
__lt__(other: ModelLocation) → bool
```

Return self<value.

```
__repr__()
```

Return repr(self).

```
__str__() → str
```

Return str(self).

```
__weakref__
```

list of weak references to the object

```
_scan_definitions(parent_def: DefinitionLocation = None, def_path: str = None) → DefinitionLocation
```

Scan recursively the definitions tree on the file model and store them.

Parameters

- **parent_def** – the current model definition to add new submodel location
- **def_path** – the absolute path to analyse

Returns

a definition location

```
_scan_profiles(path: str, profile_suffix: str = '.hmm', relative_path: bool = False) → dict[slice(<class
'str'>, <class 'str'>, None)]
```

Store all hmm profiles associated to the model

Parameters

- **path** – the path to a directory containing hmm profiles
- **profile_suffix** – the extension of hmm profile file
- **relative_path** – True if the path is relative, False otherwise.

Returns

all profiles found in the path

get_all_definitions(*root_def_name: str = None*) → list[*DefinitionLocation*]

Name root_def_name

The name of the root definition to get sub definitions. If root_def is None, return all definitions for this set of models

Returns

the list of definitions or subdefinitions if root_def is specified for this model.

Raises

ValueError – if root_def_name does not match with any definitions

get_definition(*fqn: str*) → *DefinitionLocation*

Parameters

fqn – the fully qualified name of the definition to retrieve. it's complete path without extension. for instance for a file with path like this: models/TXSS/definitions/T3SS.xml the name is: TXSS/T3SS for models/CRISPR-Cas/definitions/typing/CAS.xml: the name is CRISPR-Cas/typing/CAS

Returns

the definition corresponding to the given name.

Raise

valueError if fqn does not match with any model definition.

get_definitions() → list[*DefinitionLocation*]

Returns

the list of the definitions of this modelLocation. It returns the 1st level only (not recursive). For recursive explorations see [macsylib.registries.ModelLocation.get_all_definitions\(\)](#)

get_profile(*name: str*) → str

Parameters

name – the name of the profile to retrieve (without extension).

Returns

the absolute path of the hmm profile.

Raise

KeyError if name does not match with any profiles.

get_profiles_names() → list[str]

Returns

The list of profiles name (without extension) for this model location

property version: str

Returns

The version of the models

MetaDefLoc

class macsylib.registries.**MetaDefLoc**

DefinitionLocation

```
class macsylib.registries.DefinitionLocation(name: str | None = None, fq_n: str | None = None,
                                             subdefinitions: DefinitionLocation | None = None, path:
                                             str | None = None)
```

Manage where definitions are stored. a Model is a xml definition and associated profiles. It has 3 attributes

name: the fully qualified definitions name like TXSS/T3SS or CRISPR-cas/Typing/Cas path: the absolute path to the definitions or set of definitions subdefinitions: the subdefinitions if it exists

```
__eq__(other: DefinitionLocation) → bool
```

Return self==value.

```
__gt__(other: DefinitionLocation) → bool
```

Return self>value.

```
__hash__() → int
```

Return hash(self).

```
__init__(name: str | None = None, fq_n: str | None = None, subdefinitions: DefinitionLocation | None =
          None, path: str | None = None) → None
```

```
__lt__(other: DefinitionLocation) → bool
```

Return self<value.

```
__str__() → str
```

Return str(self).

```
__weakref__
```

list of weak references to the object

```
add_subdefinition(subdefinition: DefinitionLocation) → None
```

add new sub category of definitions to this definition

Parameters

subdefinition – the new definition to add as subdefinition.

```
all() → list[DefinitionLocation]
```

Returns

the definition and all recursively all subdefinitions

```
property family_name: str
```

Returns

the models family name which is the name of the package

```
classmethod root_name(fq_n: str) → str
```

Parameters

fq_n (str) – the fully qualified name of a definition

Returns

the root name of this definition (family name)

```
classmethod split_fqn(fq_n: str) → list[str]
```

Parameters

fq_n – the fully qualified name of a definition

Returns

each member of the fully qn in list.

split_def_name

macsylib.registries.**split_def_name**(fqn: str) → list[str]

Parameters

fqn – the fully qualified de name of a DefinitionLocation object the follow the schema model_name/<def_name>*/def_name for instance CRISPR-Cas/typing/cas

Returns

the list of components of the def path ['CRISPR-Cas', 'typing', 'cas']

join_def_path

macsylib.registries.**join_def_path**(*args: str) → str

join different elements of the definition path :param str args: the elements of the definition path, each element must be a string :return: The return value is the concatenation of different elements of args with one separator

scan_models_dir

macsylib.registries.**scan_models_dir**(models_dir: str, profile_suffix: str = '.hmm', relative_path: bool = False) → list[ModelLocation]

Parameters

- **models_dir** (str) – The path to the directory where are stored the models
- **profile_suffix** – the suffix of the hmm profiles
- **relative_path** – True if models_dir is relative false otherwise

Returns

the list of models in models_dir

Return type

[macsylib.registries.ModelLocation, ...]

definition_parser

The model definition parser object “DefinitionParser” instantiates Models and Genes objects from XML model definitions (see *Macromolecular models*). The parsing consists in three phases.

Phase 1.

- For each model to parse
 - create the Model
 - add this Model to the model_bank
 - findall genes defined in this model what are the level in the model definition.
 - create the CoreGene (a Gene which is not bind to a model). For each gene name there is only one instance of CoreGene
 - add these CoreGene in the gene_bank

Phase 2.

- For each model to search
 - For each Gene defined in this System:

- * link the gene to the model. Create a ModelGene by encapsulating CoreGene from the gene_bank It can exists at each run several ModelGene for one CoreGene
- * If a gene has exchangeables create them (an Exchangeable inherits from ModeleGene) and add them to the current ModelGene

For instance:

```
Syst_1
<system inter_gene_max_space="10">
  <gene name="A" mandatory="1" loner="1">
    <exchangeables>
      <gene name="B">
    </exchangeables>
  </gene>
</system>

Syst_2
<system inter_gene_max_space="15">
  <gene name="B" mandatory="1">
    <exchangeables>
      <gene name="C">
    </exchangeables>
  </gene>
</system>

Syst_3
<system inter_gene_max_space="20">
  <gene name="c" mandatory="1" />
</system>
```

With the example above:

- the CoreGene A, B, C will be created
- the ModelGene (Syst_1, A) (Syst_1, B), (Syst_2, B), (Syst_2, C), (Syst_3, C)
- The ModeleGene (Syst_1, A), (Syst_2, B) and (Syst_3, C) are directly link to their respective Models
- and where (Syst_1, B) (Syst_2, C) are exchangeables and link respectively to (Syst_1, A) and (Syst_2, B)
- the ModelGene has attributes defined in the model where they appear (Syst_1, B) inter_gene_max_space="10" (Syst_2, B) inter_gene_max_space="15"

Note

The only “full” Systems (*i.e.*, with all corresponding Genes created) are those to detect.

defintion_parser API reference

DefinitionParser

Module use to parse XML model defintion and create a python Model and Genes, ...

```
class macsylib.definition_parser.DefinitionParser(cfg: Config | NoneConfig, model_bank:
    ModelBank, gene_bank: GeneBank,
    model_registry: ModelRegistry, profile_factory:
    ProfileFactory)
```

Build a Model instance from the corresponding model definition described in the XML file.

__init__(*cfg*: [Config](#) | [NoneConfig](#), *model_bank*: [ModelBank](#), *gene_bank*: [GeneBank](#), *model_registry*: [ModelRegistry](#), *profile_factory*: [ProfileFactory](#)) → None

Parameters

- **cfg** – the configuration object of this run
- **model_bank** – the model factory
- **gene_bank** – the gene factory
- **model_registry** – The registry with all model location
- **profile_factory** – The profile factory

__weakref__

list of weak references to the object

_check_syntax(*model_node*: [ElementTree](#), *path*: *str*) → None

Check if the definition does not contain logical error which is allowed by syntax and absence of explicit grammar.

Parameters

- **model_node** – the node corresponding to the model
- **path** – the path of the definition.

Raises

[ModelInconsistencyError](#) – if an error is encountered in the document.

_create_model(*def_loc*: [DefinitionLocation](#), *model_node*: [ElementTree](#)) → [Model](#)

Parameters

- **def_loc** – the definition location to parse.
- **model_node** – the node corresponding to the model.

Returns

the model corresponding to the definition location.

_fill_gene_bank(*model_node*: [ElementTree](#), *model_location*: [ModelLocation](#), *def_loc*: [DefinitionLocation](#)) → None

find all gene node and add them to the gene_bank

Parameters

- **model_node** – the node corresponding to the model.
- **model_location**
- **def_loc** – a definition location corresponding to the 'model' to parse.

_get_model_node(*def_loc*: [DefinitionLocation](#)) → [ElementTree](#)

Parameters

def_loc (return the node corresponding to the 'model' tag) – a definition location to parse.

_parse_exchangeable(*gene_node*: *ElementTree*, *gene_ref*: [ModelGene](#), *curr_model*: [Model](#)) → [Exchangeable](#)

Parse a xml element gene child of exchangeable and build the corresponding object

Parameters

- **gene_node** – a “node” corresponding to the gene element in the XML hierarchy
- **gene_ref** – the gene which this gene is homolog to
- **curr_model** – the model being parsed .

Returns

the gene object corresponding to the node

_parse_genes(*model*: [Model](#), *model_node*: *ElementTree*) → None

Create genes belonging to the models. Each gene is directly added to the model in its right category (‘mandatory’, ‘accessory’, ...)

Parameters

- **model** – the Model currently parsing
- **model_node** – the element ‘model’

check_consistency(*models*: *list*[[Model](#)]) → None

Check the consistency of the co-localization features between the different values given as an input: between XML definitions, configuration file, and command-line options.

Parameters

models – the list of models to check

Raise

[macsylib.error.ModelInconsistencyError](#) if one test fails

(see [feature](#))

In the different possible situations, different requirements need to be fulfilled (“mandatory_genes” and “accessory_genes” consist of lists of genes defined as such in the model definition):

- **If:** min_mandatory_genes_required = None ; min_genes_required = None
- **Then:** min_mandatory_genes_required = min_genes_required = len(mandatory_genes)

always True by Models design

- **If:** min_mandatory_genes_required = value ; min_genes_required = None
- **Then:** min_mandatory_genes_required <= len(mandatory_genes)
- AND min_genes_required = min_mandatory_genes_required

always True by design

- **If:** min_mandatory_genes_required = None ; min_genes_required = Value
- **Then:** min_mandatory_genes_required = len(mandatory_genes)
- AND min_genes_required >= min_mandatory_genes_required
- AND min_genes_required <= len(mandatory_genes+accessory_genes)

to be checked

- **If:** min_mandatory_genes_required = Value ; min_genes_required = Value
- **Then:** min_genes_required <= len(accessory_genes+mandatory_genes)

- AND min_genes_required >= min_mandatory_genes_required
- AND min_mandatory_genes_required <= len(mandatory_genes)

to be checked

parse(models_2_detect: list[DefinitionLocation]) → None

Parse models definition in XML format to build the corresponding Model objects, and add them to the model factory after checking its consistency. To get the model ask it to model_bank

Parameters

models_2_detect – a list of model definition to parse.

model

The model is a formal representation of system. The model is describe in terms of components. There are 4 component classes:

- genes which are mandatory
- genes which are accessory
- genes which are neutral
- genes which are forbidden

Each genes can have Exchangeable. An exchangeable is another gene which can paly the same role in the system. Usually an analog or homolog. The models describe also distance constraints between genes:

- inter_gene_max_space
- loner
- multi_loci

and quorum constraints

- min_mandatory_genes_required
- min_genes_required

and if a gene can be shared by several systems (several occurrences of the same model)

- multisystem

model API reference

ModelBank

class macsylib.model.ModelBank

Store all Models objects.

__contains__(model: Model) → bool

Implement the membership test operator

Parameters

model – the model to test

Returns

True if the model is in the Model factory, False otherwise

__getitem__(fqn: str) → Model

Parameters

fqn – the fully qualified name of the model

Returns

the model corresponding to the fq. n.

Raises

KeyError – if the model corresponding to the name does not exist

__init__() → None

__iter__() → Iterator

Returns

an iterator object on the models contained in the bank

__len__() → int

Returns

the number of models stored in the bank

__weakref__

list of weak references to the object

add_model(*model*: [Model](#)) → None

Parameters

model – the model to add

Raise

KeyError if a model with the same name is already registered.

Model

```
class macsylib.model.Model(*args, **kwargs)
```

Handles a macromolecular model.

Contains all its pre-defined characteristics expected to be fulfilled to predict a complete model:

- component list (genes that are mandatory, accessory, neutral, forbidden)
- quorum (number of genes)
- genetic architecture

__eq__(*other*: [Model](#)) → bool

Parameters

other – the other model to compare

Returns

True if this fully qualified name is equal to other fully qualified name. False otherwise.

__gt__(*other*: [Model](#)) → bool

Parameters

other – the other model to compare

Returns

True if this fully qualified name is greater than to other fully qualified name. False otherwise.

__hash__() → int

Returns

__init__(*fq*: str, *inter_gene_max_space*: int, *min_mandatory_genes_required*: int = None, *min_genes_required*: int = None, *max_nb_genes*: int = None, *multi_loci*: bool = False) → None

Parameters

- **fqn** – the fully qualified name of the model CRISPR-Cas/sub-typing/CAS-TypeIE
- **inter_gene_max_space** – the maximum distance between two genes (**co-localization** parameter)
- **min_mandatory_genes_required** – the quorum of mandatory genes to define this model
- **min_genes_required** – the quorum of genes to define this model
- **max_nb_genes** – The number of gene to be considered as full system Used to compute the wholeness. If None the mx_nb_genes = mandatory + accessory
- **multi_loci** – if the systems can split in different loci on the genome

Raises

ModelInconsistencyError – if an error is found in model logic. For instance *genes_required > min_mandatory_genes_required*

__lt__(*other*: [Model](#)) → bool

Parameters

other – the other model to compare

Returns

True if this fully qualified name is lesser than to other fully qualified name. False otherwise.

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

property family_name: str

Returns

the family name of the model for instance ‘CRISPRCas’ or ‘TXSS’

filter(*hits*: list[[CoreHit](#)]) → list[[ModelHit](#)]

filter out the hits according to this model and cast them in [ModelHit](#). The filtering is based on the name of [CoreGene](#) associated to hit and the name of [ModelGene](#) of the model (the name of the [ModelGene](#) is the name of the [CoreGene](#) embedded in the [ModelGene](#)) only the hits related to genes implied in the model are kept.

Parameters

hits (list of `macsylib.report.CoreHit` object) – list of hits to filter

Returns

list of hits

Return type

list of `macsylib.report.Model` object

genes(*exchangeable*: bool = False) → set[[ModelGene](#)]

Parameters

exchangeable – include exchangeable if True

Returns

all the genes described in the model. with exchangeables if exchangeable is True. otherwise only “first level” genes.

get_gene(*gene_name: str*) → *ModelGene*

Parameters

gene_name – the name of the gene to get

Returns

the gene corresponding to gene_name.

Raise

KeyError the model does not contain any gene with name gene_name.

property inter_gene_max_space: int

Returns

set the maximum distance allowed between 2 genes for this model

property max_nb_genes: int

Returns

the maximum number of genes to assess the model presence.

property min_genes_required: int

Returns

get the minimum number of genes to assess for the model presence.

Return type

integer

property min_mandatory_genes_required: int

Returns

get the quorum of mandatory genes required for this model

property multi_loci: bool

Returns

True if the model is authorized to be inferred from multiple loci, False otherwise

property name: str

Returns

the short name of this model

gene

The *Gene object* represents genes encoding the protein components of a Model. There is 2 kind of gene The CoreGene (*macsylib.gene.CoreGene*) which must be unique given a name. A CoreGene must have a corresponding HMM protein profile. A ModelGene encapsulate a CoreGene and is linked to a Model.

Warning

To optimize computation and to avoid concurrency problems when we search several models, each gene must be instantiated only once, and stored in gene_bank. gene_bank is a *macsylib.gene.GeneBank* object. The gene_bank and model_bank (*macsylib.model.ModelBank* object) are instantiated in *macsylib.search_systems.search_systems()* function and filled by a definition_parser (*macsylib.definition_parser.DefinitionParser*)

Example to get a CoreGene object:


```
# get a model object
model_a = model_bank("TXSS/model_a")
model_b = model_bank("TXSS/model_b")

# get of a <CoreGene> object
t2ss = gene_bank(["TXSS", "T2SS"])
pilo = gene_bank(["TXSS", "pilo"])
```

to create a ModelGene

```
modelA_t2ss(t2ss, model_A)
modelA_pilo(pilo, model_a, loner=True, inter_gene_max_space=12)
modelB_pilo(pilo, model_b, inter_gene_max_space=5)
```

There is only *one* instance of CoreGene with a given name (model family name, gene name) in one MSF run. But several instance of a ModelGene with the same name may exists. Above, there is 2 <ModelGene> representing *pilo* one in model_a the second in model_b with different properties.

Exchangeable inherits from ModelGene. Then a gene in some model is seen as a Gene, in some other models as an Exchangeable. But there only one instance of the corresponding CoreGene.:

```
core_sctn = gene_bank(("TXSS", "sctN"))
core_sctn_flg = gene_bank(("TXSS", "sctN_FLG"))
model_sctn = ModelGene(core_sctn, model_a)
ex_sctn_flg = Exchangeable(core_sctn_flg, model_sctn)
model_sctn.add_exchangeable(ex_sctn_flg)

model_sctn_flg = ModelGene(core_sctn_flg, model_b)
```

which means that in model_a the gene *sctn* can be functionally replaced by *sctn_flg*. In Model_a it appear as an alternative to *sctn* but in model_B it appear as *sctn_flg* itself. In one MacSyLib run several instances of ModelGene and/or Exchangeable with the same name may coexists . But in A whole macsylib run there is only one instance *core_sctn_flg* and *core_sctn*.

gene API reference

GeneBank

class macsylib.gene.GeneBank

Store all Gene objects. Ensure that genes are instanced only once.

__contains__(gene: CoreGene) → bool

Implement the membership test operator

Parameters

gene – the gene to test

Returns

True if the gene is in, False otherwise

Return type

boolean

__getitem__(key: tuple[str, str]) → CoreGene

Parameters

key – The key to retrieve a gene. The key is composed of the name of models family and the gene name. for instance CRISPR-Cas/cas9_TypeIIB (‘CRISPR-Cas’, ‘cas9_TypeIIB’) or TXSS/T6SS_tssH (‘TXSS’, ‘T6SS_tssH’)

Returns

return the Gene corresponding to the key.

Raises

KeyError – if the key does not exist in GeneBank.

__init__() → None

__iter__() → Iterator

Return an iterator object on the genes contained in the bank

__weakref__

list of weak references to the object

add_new_gene(*model_location*: [ModelLocation](#), *name*: str, *profile_factory*: [ProfileFactory](#)) → None

Create a gene and store it in the bank. If the same gene (same name) is added twice, it is created only the first time.

Parameters

- **model_location** – the location where the model family can be found.
- **name** – the name of the gene to add
- **profile_factory** – The Profile factory

genes_fqn() → list[str]

Returns

the fully qualified name for all genes in the bank

Gene

There is two classes to modelize a gene: [macsylib.gene.CoreGene](#) and [macsylib.gene.ModelGene](#). The CoreGene are created using the [macsylib.gene.GeneBank](#) factory and there is only one instance of a CoreGene with a given name. Whereas several ModelGene with the same name can appear in different model and can have differents properties, *loner* in one model and not in an other, have different *inter_gene_max_space* ... The ModelGene is attached to the model and is composed of a CoreGene.

Note

The [macsylib.hit.Hit](#) object are link to a CoreGene, whereas the [macsylib.hit.ValidHit](#) *ref_gene* attribute reference a [macsylib.gene.ModelGene](#)

CoreGene

class [macsylib.gene.CoreGene](#)(*model_location*: [ModelLocation](#), *name*: str, *profile_factory*: [ProfileFactory](#))

Modeling gene attached to a profile. It can be only one instance with the same name (family name, gene name)

__hash__() → int

Return hash(self).

__init__(*model_location*: [ModelLocation](#), *name*: str, *profile_factory*: [ProfileFactory](#)) → None

__weakref__

list of weak references to the object

property model_family_name: str

The name of the model family for instance 'CRISPRCas' or 'TXSS'

property name: `str`

The name of the gene a hmm profile with the same name must exist.

property profile: `Profile`

The HMM protein Profile corresponding to this gene

ModelGene

```
class macsylib.gene.ModelGene(gene: CoreGene, model: Model, loner: bool = False, multi_system: bool = False,
                                inter_gene_max_space: int = None, multi_model: bool = False)
```

Handle Gene described in a Model

`__hash__()` → `int`

Return hash(self).

```
__init__(gene: CoreGene, model: Model, loner: bool = False, multi_system: bool = False,
          inter_gene_max_space: int = None, multi_model: bool = False)
```

Handle gene described in a Model

Parameters

- **gene** – a gene link to a profile
- **model** – the model that owns this Gene
- **loner** – True if the Gene can be isolated on the genome (with no contiguous genes), False otherwise.
- **multi_system** – True if this Gene can belong to different occurrences of this System.
- **inter_gene_max_space** – the maximum space between this Gene and another gene of the System.
- **multi_model** – True if this Gene is allowing to appear in several system occurrence from different model.

`__str__()` → `str`

Print the name of the gene and of its exchangeable genes.

`__weakref__`

list of weak references to the object

add_exchangeable(exchangeable: `Exchangeable`)

Add an exchangeable gene to this Gene

Parameters

exchangeable – the exchangeable to add

alternate_of() → `ModelGene`

Returns

the gene to which this one is an exchangeable to (reference gene), or itself if it is a first level gene.

property core_gene: `CoreGene`

Returns

The CoreGene associated to this ModelGene

property exchangeables: list[*ModelGene*]

Returns

the list of genes which can replace this one without any effect on the model

property inter_gene_max_space: int | None

Returns

The maximum distance allowed between this gene and another gene for them to be considered co-localized. If the value is not set at the Gene level, return None.

is_accessory() → bool

Returns

True if the gene is within the *accessory* genes of the model, False otherwise.

Do not take in account the exchangeable For instance:

```
<gene name="A" presence="accessory">
  <exchangeables>
    <gene name="B />
  </exchangeables>
</gene>
```

gene_A.is_accessory -> True gene_B.is_accessory -> False To get the real status of a gene whatever it is a regular or exchangeable gen use instead status property gene_A.status -> <GeneStatus.ACCESSORY: 2> gene_B.status -> <GeneStatus.ACCESSORY: 2>

property is_exchangeable: bool

Returns

True if this gene is described in the model as an exchangeable. False if it is described as first level gene.

is_forbidden() → bool

Returns

True if the gene is within the *forbidden* genes of the model, False otherwise.

Do not take in account the exchangeable For instance:

```
<gene name="A" presence="forbidden">
  <exchangeables>
    <gene name="B />
  </exchangeables>
</gene>
```

gene_A.is_forbidden -> True gene_B.is_forbidden -> False To get the real status of a gene whatever it is a regular or exchangeable gen use instead status property gene_A.status -> <GeneStatus.FORBIDDEN: 3> gene_B.status -> <GeneStatus.FORBIDDEN: 3>

is_mandatory() → bool

Returns

True if the gene is within the *mandatory* genes of the model, False otherwise.

Do not take in account the exchangeable For instance:

```
<gene name="A" presence="mandatory">
  <exchangeables>
    <gene name="B" />
  </exchangeables>
</gene>
```

gene_A.is_mandatory -> True gene_B.is_mandatory -> False To get the real status of a gene whatever it is a regular or exchangeable gen use instead status property gene_A.status -> <GeneStatus.MANDATORY: 1> gene_B.status -> <GeneStatus.MANDATORY: 1>

property loner: **bool**

Returns

True if the gene can be isolated on the genome, False otherwise

property model: *Model*

Returns

the Model that owns this Gene

property multi_model: **bool**

Returns

True if this Gene can belong to different occurrences of systems from different model *macsylib.model.Model* (and can be used for multiple System assessments), False otherwise.

property multi_system: **bool**

Returns

True if this Gene can belong to different occurrences of **the model** (and can be used for multiple System assessments), False otherwise.

set_status(status: *GeneStatus*) → None

Set the status for this gene

Parameters

status – the status of this gene

property status: *GeneStatus*

Returns

The status of this gene

Exchangeable

```
class macsylib.gene.Exchangeable(c_gene: CoreGene, gene_ref: ModelGene, loner: bool | None = None,
                                  multi_system: bool | None = None, multi_model: bool | None = None,
                                  inter_gene_max_space: int | None = None)
```

Handle Exchangeables. Exchangeable are ModelGene which can replaced functionally another ModelGene. Biologically it can be Homolog or Analog

```
__init__(c_gene: CoreGene, gene_ref: ModelGene, loner: bool | None = None, multi_system: bool | None = None, multi_model: bool | None = None, inter_gene_max_space: int | None = None) → None
```

Parameters

- **c_gene** – the gene

- **gene_ref** – the gene to which the current can replace it.

add_exchangeable(exchangeable: [Exchangeable](#)) → None

This method should never be called, it's a security to avoid to add exchangeable to an exchangeable.

Parameters

exchangeable – the exchangeable gene to add

Raises

[MacsylibError](#) –

alternate_of() → [ModelGene](#)

Returns

the gene to which this one is an exchangeable to (reference gene)

property is_exchangeable: bool

Returns

True

property status: [GeneStatus](#)

Returns

The status of this gene. if the status is not define for this gene itself, return the status of the reference gene.

GeneStatus

class macsylib.gene.**GeneStatus**(*values)

Handle status of Gene GeneStatus can take 4 value:

- MANDATORY
- ACCESSORY
- FORBIDDEN
- NEUTRAL

__str__() → str

Return str(self).

profile

The *Profile object* is used for the search of the gene with Hmmer. A “*Profile*” must match a HMM protein profile file, which name is based on the profile name. For instance, the *gspG* gene has the corresponding “gspG.hmm” profile file provided at a dedicated location.

profile API reference

ProfileFactory

class macsylib.profile.**ProfileFactory**(cfg: [Config](#) | [NoneConfig](#))

Build and store all Profile objects. Profiles must not be instantiated directly. The profile_factory must be used. The profile_factory ensures there is only one instance of profile for a given name. To get a profile, use the method get_profile. If the profile is already cached, this instance is returned. Otherwise, a new profile is built, stored in the profile_factory and then returned.

__init__(*cfg*: [Config](#) | [NoneConfig](#)) → None

__weakref__

list of weak references to the object

get_profile(*gene*: [CoreGene](#), *model_location*: [ModelLocation](#)) → [Profile](#)

Parameters

- **gene** – the gene associated to this profile
- **model_location** – The where to get the profile

Returns

The profile corresponding to the name. If the profile already exists, return it. Otherwise, build it, store it and return it.

Profile

class `macsylib.profile.Profile`(*gene*: [CoreGene](#), *cfg*: [Config](#), *path*: *str*)

Handle a HMM protein profile

__init__(*gene*: [CoreGene](#), *cfg*: [Config](#), *path*: *str*) → None

Parameters

- **gene** – the gene corresponding to this profile
- **cfg** – the configuration
- **path** – the path to the hmm profile.

__len__() → int

Returns

the length of the HMM protein profile

Return type

int

__str__() → str

Print the name of the corresponding gene and the path to the HMM profile.

__weakref__

list of weak references to the object

_profile_features() → tuple[int, bool]

Parse the HMM profile to extract the length and the presence of GA bit threshold

Returns

the length, presence of ga bit threshold

execute(*cpu*: *int* = 1) → [HMMReport](#) | None

Launch the Hmmer search (hmmsearch executable) with this profile

Parameters

- **cpu** – the number of cpu to use for hmmsearch (must be >= 1)

Returns

an object storing information on the results of the HMM search ([HMMReport](#))

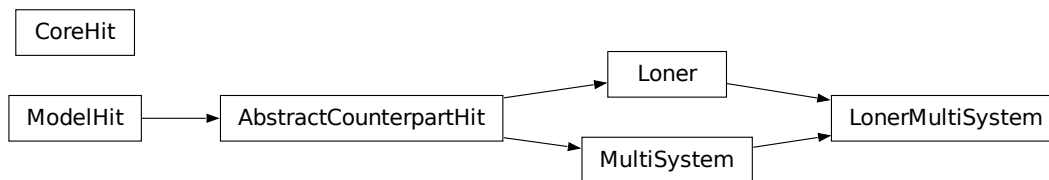
hit

This module implements class relative to hit and some functions to do some computation on hit objects.

<code>macsylib.hit.CoreHit</code>	Modelize a hmm hit on the replicon. There is only one Corehit for a CoreGene.
<code>macsylib.hit.ModelHit</code>	Modelize a hit and its relation to the Model.
<code>macsylib.hit.AbstractCounterpartHit</code>	Parent class of Loner, MultiSystem. It's inherits from ModelHit.
<code>macsylib.hit.Loner</code>	Modelize "true" Loner.
<code>macsylib.hit.MultiSystem</code>	Modelize hit which can be used in several Systems (same model)
<code>macsylib.hit.LonerMultiSystem</code>	Modelize a hit representing a gene Loner and MultiSystem at same time.
<code>macsylib.hit.HitWeight</code>	The weights apply to the hit to compute score
<code>macsylib.hit.get_best_hit_4_func()</code>	Return the best hit for a given function
<code>macsylib.hit.sort_model_hits()</code>	Sort hits
<code>macsylib.hit.compute_best_MSHit()</code>	Choose among several multisystem hits the best one
<code>macsylib.hit.get_best_hits()</code>	If several profile hit the same gene return the best hit

A Hit is created when *hmmsearch* find similarities between a profile and protein of the input dataset

Below the inheritance diagram of Hits



And a diagram showing the interaction between CoreGene, ModelGene, Model, Hit, Loner, ... interactions

hit API reference

CoreHit

```
class macsylib.hit.CoreHit(gene: CoreGene, hit_id: str, hit_seq_length: int, replicon_name: str,
                           position_hit: int, i_eval: float, score: float, profile_coverage: float,
                           sequence_coverage: float, begin_match: int, end_match: int)
```

Handle the hits filtered from the Hmmer search. The hits are instantiated by `HMMReport.extract()` method In one run of MacSyLib, there exists only one CoreHit per gene These hits are independent of any `macsylib.model.Model` instance.

```
__eq__(other: CoreHit) → bool
```

Return True if two hits are totally equivalent, False otherwise.

Parameters

other – the hit to compare to the current object

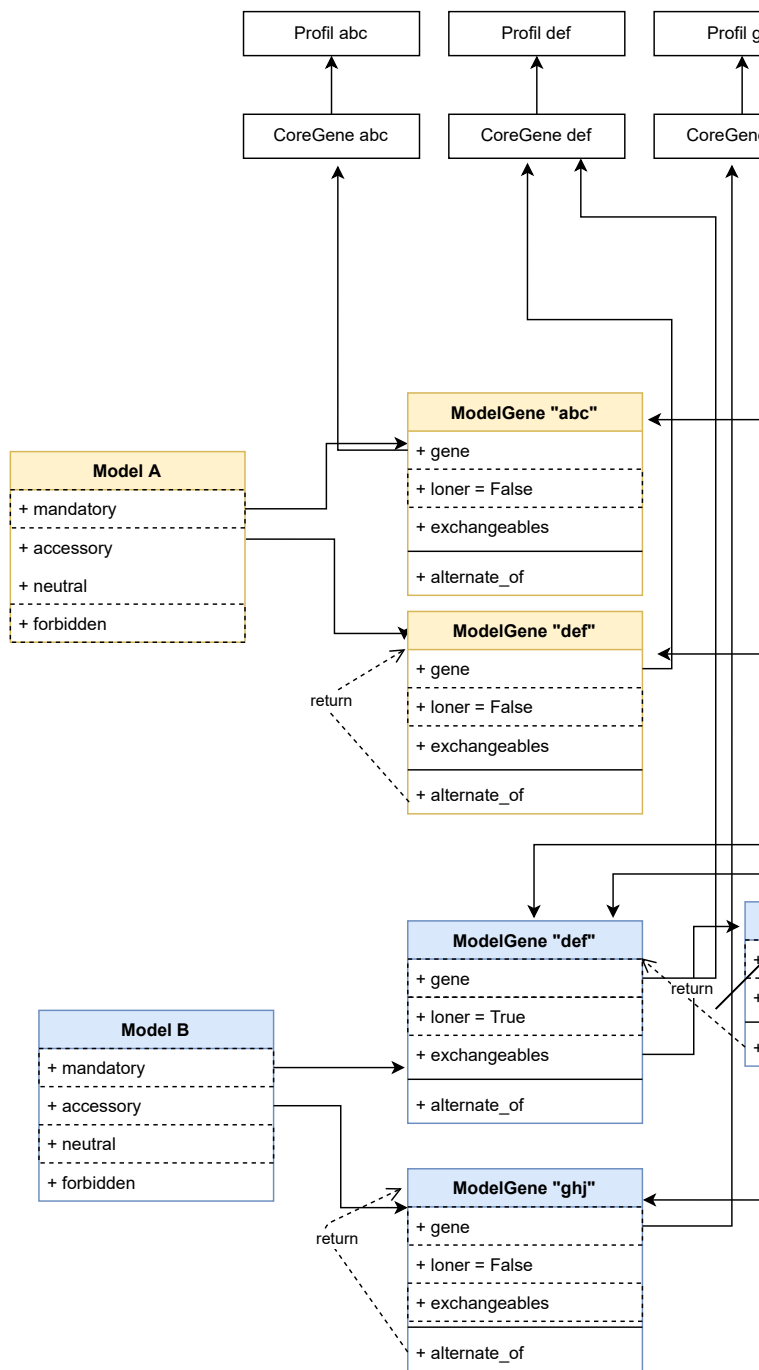


Fig. 2: The diagram above represents the models, genes and hit generated from the definitions below.

Returns

the result of the comparison

__gt__(*other*: [CoreHit](#)) → bool

compare two Hits. If the sequence identifier is the same, do the comparison on the score. Otherwise, do it on alphabetical comparison of the sequence identifier.

Parameters

other – the hit to compare to the current object

Returns

True if self is > other, False otherwise

__hash__() → int

To be hashable, it's needed to be put in a set or used as dict key

__init__(*gene*: [CoreGene](#), *hit_id*: str, *hit_seq_length*: int, *replicon_name*: str, *position_hit*: int, *i_eval*: float, *score*: float, *profile_coverage*: float, *sequence_coverage*: float, *begin_match*: int, *end_match*: int) → None

Parameters

- **gene** – the gene corresponding to this profile
- **hit_id** – the identifier of the hit
- **hit_seq_length** – the length of the hit sequence
- **replicon_name** – the name of the replicon
- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_eval** – the best-domain evaluate (i-evalue, “independent evaluate”)
- **score** – the score of the hit
- **profile_coverage** – percentage of the profile that matches the hit sequence
- **sequence_coverage** – percentage of the hit sequence that matches the profile
- **begin_match** – where the hit with the profile starts in the sequence
- **end_match** – where the hit with the profile ends in the sequence

__lt__(*other*: [CoreHit](#)) → bool

Compare two Hits. If the sequence identifier is the same, do the comparison on the score. Otherwise, do it on alphabetical comparison of the sequence identifier.

Parameters

other – the hit to compare to the current object

Returns

True if self is < other, False otherwise

__str__() → str

Returns

Useful information on the CoreHit: regarding Hmmer statistics, and sequence information

Return type

str

__weakref__

list of weak references to the object

ModelHit

class `macsylib.hit.ModelHit`(*hit*: `CoreHit`, *gene_ref*: `ModelGene`, *gene_status*: `GeneStatus`)

Encapsulates a `macsylib.report.CoreHit` This class stores a `CoreHit` that has been attributed to a putative system. Thus, it also stores:

- the system,
- the status of the gene in this system, ('mandatory', 'accessory', ...)
- the gene in the model for which it's an occurrence

for one gene it can exist several `ModelHit` instance one for each `Model` containing this gene

`__eq__`(*other*: `ModelHit`) → bool

Return self==value.

`__gt__`(*other*: `ModelHit`) → bool

Return self>value.

`__hash__`() → int

To be hashable, it's needed to be put in a set or used as dict key

`__init__`(*hit*: `CoreHit`, *gene_ref*: `ModelGene`, *gene_status*: `GeneStatus`) → None

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The `ModelGene` link to this hit The `ModelGene` have the same name as the `CoreGene` But one hit can be linked to several `ModelGene` (several `Model`) To know for what gene this hit play role use the `macsylib.gene.ModelGene.alternate_of()`

```
hit.gene_ref.alternate_of()
```

- **gene_status**

`__lt__`(*other*: `ModelHit`) → bool

Return self<value.

`__str__`() → str

Return str(self).

`__weakref__`

list of weak references to the object

property `hit`: `CoreHit`

Returns

The `CoreHit` below this `ModelHit`

property `loner`: bool

Returns

True if the hit represent a *loner* `macsylib.Gene.ModelGene`, False otherwise. A True *Loner* is a hit representing a gene with the attribute *loner* and which does not include in a cluster.

- a hit representing a *loner* gene but include in a cluster is not a true *loner*
- a hit which is not include with other gene in a cluster but does not represent a gene *loner* is not a True *loner* (This situation may append when `min_genes_required = 1`)

property multi_model: bool

Returns

True if the hit represent a *multi_model* `macsylib.Gene.ModelGene`, False otherwise.

property multi_system: bool

Returns

True if the hit represent a *multi_system* `macsylib.Gene.ModelGene`, False otherwise.

AbstractCounterpartHit

```
class macsylib.hit.AbstractCounterpartHit(hit: CoreHit | ModelHit, gene_ref: ModelGene = None,
                                          gene_status: GeneStatus = None, counterpart: set[ModelHit]
                                          = None)
```

Abstract Class to handle ModelHit wit equivalent for instance Loner or MultiSystem hit

```
__init__(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None,
         counterpart: set[ModelHit] = None) → None
```

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModeleGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the `macsylib.gene.ModelGene.alternate_of()`

`hit.gene_ref.alternate_of()`

- **gene_status**

```
__str__() → str
```

Return str(self).

property counterpart: set[ModelHit]

Returns

The set of hits that can play the same role

property loner: bool

Returns

True if the hit represent a *loner* `macsylib.Gene.ModelGene`, False otherwise. A True Loner is a hit representing a gene with the attribute loner and which does not include in a cluster.

- a hit representing a loner gene but include in a cluster is not a true loner
- a hit which is not include with other gene in a cluster but does not represent a gene loner is not a True loner (This situation may append when `min_genes_required = 1`)

property multi_system: bool

Returns

True if the hit represent a *multi_system* `macsylib.Gene.ModelGene`, False otherwise.

Loner

```
class macsylib.hit.Loner(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None, counterpart: Iterable[ModelHit] = None)
```

Handle hit which encode for a gene tagged as loner and which not clustering with other hit.

```
__init__(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None, counterpart: Iterable[ModelHit] = None) → None
```

hit that is outside a cluster, the gene_ref is a loner

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModelGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the `macsylib.gene.ModelGene.alternate_of()`

```
hit.gene_ref.alternate_of()
```

- **gene_status**
- **counterpart** – the other occurrence of the gene or exchangeable in the replicon

property loner

Returns

True if the hit represent a *loner* `macsylib.Gene.ModelGene`, False otherwise. A True Loner is a hit representing a gene with the attribute loner and which does not include in a cluster.

- a hit representing a loner gene but include in a cluster is not a true loner
- a hit which is not include with other gene in a cluster but does not represent a gene loner is not a True loner (This situation may append when `min_genes_required = 1`)

MultiSystem

```
class macsylib.hit.MultiSystem(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None, counterpart: Iterable[ModelHit] = None)
```

Handle hit which encode for a gene tagged as loner and which not clustering with other hit.

```
__init__(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None, counterpart: Iterable[ModelHit] = None)
```

hit that is outside a cluster, the gene_ref is a loner

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModelGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the `macsylib.gene.ModelGene.alternate_of()`

```
hit.gene_ref.alternate_of()
```

- **gene_status**
- **counterpart** – the other occurrence of the gene or exchangeable in the replicon

property multi_system: bool

Returns

True if the hit represent a *multi_system* `macsylib.Gene.ModelGene`, False otherwise.

LonerMultiSystem

```
class macsylib.hit.LonerMultiSystem(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status:
    GeneStatus = None, counterpart: Iterable[ModelHit] = None)
```

Handle hit which encode for a gene

- gene tagged as multi-system
- and gene tagged as loner also
- and the hit do not clustering with other hits.

```
__init__(hit: CoreHit | ModelHit, gene_ref: ModelGene = None, gene_status: GeneStatus = None,
    counterpart: Iterable[ModelHit] = None)
```

hit that is outside a cluster, the `gene_ref` is loner and `multi_system`

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** (`macsylib.gene.ModelGene` object) – The `ModelGene` link to this hit The `ModelGene` have the same name as the `CoreGene` But one hit can be linked to several `ModelGene` (several `Model`) To know for what gene this hit play role use the `macsylib.gene.ModelGene.alternate_of()`

`hit.gene_ref.alternate_of()`

- **gene_status** (`macsylib.gene.GeneStatus` object)
- **counterpart** (list of `macsylib.hit.CoreHit`) – the other occurrence of the gene or exchangeable in the replicon

HitWeight

```
class macsylib.hit.HitWeight(itself: float = 1, exchangeable: float = 0.8, mandatory: float = 1, accessory:
    float = 0.5, neutral: float = 0, out_of_cluster: float = 0.7)
```

The weight to compute the cluster and system score see user documentation MacSyLib functioning for further details by default

- `itself` = 1
- `exchangeable` = 0.8
- `mandatory` = 1
- `accessory` = 0.5
- `neutral` = 0
- `out_of_cluster` = 0.7

```
__delattr__(name)
```

Implement `delattr(self, name)`.

```

__eq__(other)
    Return self==value.

__hash__()
    Return hash(self).

__init__(itself: float = 1, exchangeable: float = 0.8, mandatory: float = 1, accessory: float = 0.5, neutral:
    float = 0, out_of_cluster: float = 0.7) → None

__repr__()
    Return repr(self).

__setattr__(name, value)
    Implement setattr(self, name, value).

__weakref__
    list of weak references to the object

```

get_best_hit_4_func

`macsylib.hit.get_best_hit_4_func(function: str, hits: Iterable[ModelHit], key: str = 'score') → ModelHit`
 select the best Loner among several ones encoding for same function

- score
- i_evalue
- profile_coverage

Parameters

- **function** – the name of the function fulfill by the hits (all hits must code for the same gene)
- **hits** – the hits to filter.
- **key** – The criterion used to select the best hit ‘score’, ‘i_evalue’, ‘profile_coverage’

Returns

the best hit

sort_model_hits

`macsylib.hit.sort_model_hits(model_hits: Iterable[ModelHit]) → dict[slice(<class 'str'>, list[macsylib.hit.ModelHit], None)]`

Sort *macsylib.hit.ModelHit* per function

Parameters

model_hits – a sequence of *macsylib.hit.ModelHit*

Returns

dict {str function name: [model_hit, ...] }

compute_best_MSHit

`macsylib.hit.compute_best_MSHit(ms_registry: dict[slice(<class 'str'>, list[macsylib.hit.MultiSystem | macsylib.hit.LonerMultiSystem], None)]) → list[MultiSystem | LonerMultiSystem]`

Parameters**ms_registry****Returns****get_best_hits**

`macsylib.hit.get_best_hits(hits: Iterable[CoreHit | ModelHit], key: Literal['score', 'i_eval', 'profile_coverage'] = 'score') → list[CoreHit | ModelHit]`

If several hits match the same protein, keep only the best match based either on

- score
- i_evalue
- profile_coverage

Parameters

- **hits** ([*macsylib.hit.CoreHit* object, ...]) – the hits to filter, all hits must match the same protein.
- **key** (str) – The criterion used to select the best hit ‘score’, ‘i_evalue’, ‘profile_coverage’

Returns

the list of the best hits

Return type

[*macsylib.hit.CoreHit* object, ...]

cluster

A cluster is an ordered set of hits related to a model which satisfy the model distance constraints.

cluster API reference**Class Cluster**

class `macsylib.cluster.Cluster(hits: list[CoreHit] | list[ModelHit], model, hit_weights)`

Handle hits relative to a model which collocates

__contains__(*m_hit*: ModelHit) → bool

Parameters

m_hit – The hit to test

Returns

True if the hit is in the cluster hits, False otherwise

__init__(*hits*: list[CoreHit] | list[ModelHit], *model*, *hit_weights*) → None

Parameters

- **hits** – the hits constituting this cluster
- **model** – the model associated to this cluster
- **hit_weights** – the weight of the hit to compute the score

`__str__()` → str

Returns

a string representation of this cluster

`__weakref__`

list of weak references to the object

`_check_replicon_consistency()` → None

Raise

MacsylibError if all hits of a cluster are NOT related to the same replicon

`fulfilled_function(*genes: ModelGene | str) → frozenset[str]`

Parameters

genes – The genes which must be tested.

Returns

the common functions between genes and this cluster.

property functions: frozenset[str]

Returns

The set of functions encoded by this cluster *function* mean gene name or reference gene name for exchangeables genes for instance

```
<model vers="2.0">
  <gene a presence="mandatory"/>
  <gene b presence="accessory"/>
    <exchangeable>
      <gene c />
    </exchangeable>
  <gene />
</model>
```

the functions for a cluster corresponding to this model will be {'a', 'b'}

property hit_weights: HitWeight

Returns

the different weight for the hits used to compute the score

property hits: list[CoreHit | ModelHit]

Returns

the hits sorted by the increasing position

property loner: bool

Returns

True if this cluster is made of only some hits representing the same gene and this gene is tag as loner False otherwise:

- contains several hits coding for different genes
- contains one hit but gene is not tag as loner (max_gene_required = 1)

merge(*cluster*: [Cluster](#), *before*: *bool* = *False*) → None

merge the cluster param in this one. (do it in place)

Parameters

- **cluster**
- **before** (*bool*) – If *False* the hits of the cluster will be added at the end of this one, Otherwise the cluster hits will be inserted before the hits of this one.

Raises

MacError – if the two clusters have not the same model

property multi_system: *bool*

Returns

True if this cluster is made of only one hit representing a multi_system gene False otherwise:

- contains several hits
- contains one hit but gene is not tag as loner (max_gene_required = 1)

replace(*old*: [ModelHit](#), *new*: [ModelHit](#)) → None

replace hit old in this cluster by new one. (do it in place) beware the hits in a cluster are sorted by their position so if old hit and new hit have not same position the order will be changed

Parameters

- **old** – the hit to replace
- **new** – the new hit

Returns

None

property replicon_name: *str*

Returns

The name of the replicon where this cluster is located

Return type

str

property score: *float*

Returns

The score for this cluster

cluster functions

Functions that help to build `macsylib.cluster.Cluster` object.

class `macsylib.cluster.Cluster`(*hits*: *list*[[CoreHit](#)] | *list*[[ModelHit](#)], *model*, *hit_weights*)

Handle hits relative to a model which collocates

fulfilled_function(**genes*: [ModelGene](#) | *str*) → *frozenset*[*str*]

Parameters

genes – The genes which must be tested.

Returns

the common functions between genes and this cluster.

property functions: `frozenset[str]`

Returns

The set of functions encoded by this cluster *function* mean gene name or reference gene name for exchangeables genes for instance

```
<model vers="2.0">
  <gene a presence="mandatory"/>
  <gene b presence="accessory"/>
    <exchangeable>
      <gene c />
    </exchangeable>
  <gene />
</model>
```

the functions for a cluster corresponding to this model will be {'a', 'b'}

property hit_weights: `HitWeight`

Returns

the different weight for the hits used to compute the score

property hits: `list[CoreHit | ModelHit]`

Returns

the hits sorted by the increasing position

property loner: `bool`

Returns

True if this cluster is made of only some hits representing the same gene and this gene is tag as loner False otherwise:

- contains several hits coding for different genes
- contains one hit but gene is not tag as loner (`max_gene_required = 1`)

merge(*cluster*: `Cluster`, *before*: `bool = False`) → `None`

merge the cluster param in this one. (do it in place)

Parameters

- **cluster**
- **before** (`bool`) – If False the hits of the cluster will be added at the end of this one, Otherwise the cluster hits will be inserted before the hits of this one.

Raises

MacError – if the two clusters have not the same model

property multi_system: `bool`

Returns

True if this cluster is made of only one hit representing a multi_system gene False otherwise:

- contains several hits
- contains one hit but gene is not tag as loner (`max_gene_required = 1`)

replace(*old*: [ModelHit](#), *new*: [ModelHit](#)) → None

replace hit old in this cluster by new one. (do it in place) beware the hits in a cluster are sorted by their position so if old hit and new hit have not same position the order will be changed

Parameters

- **old** – the hit to replace
- **new** – the new hit

Returns

None

property replicon_name: **str**

Returns

The name of the replicon where this cluster is located

Return type

str

property score: **float**

Returns

The score for this cluster

macsylib.cluster.build_clusters(*hits*: list[[ModelHit](#)], *rep_info*: [RepliconInfo](#), *model*: [Model](#), *hit_weights*: [HitWeight](#)) → tuple[list[~macsylib.cluster.Cluster], dict[slice(<class 'str'>, macsylib.hit.Loner | macsylib.hit.LonerMultiSystem, None)]]

From a list of filtered hits, and replicon information (topology, length), build all lists of hits that satisfied the constraints:

- **max_gene_inter_space**
- **loner**
- **multi_system**

If Yes create a cluster. A cluster contains at least two hits separated by less or equal than **max_gene_inter_space** Except for loner genes which are allowed to be alone in a cluster

Parameters

- **hits** – list of filtered hits
- **rep_info** – the replicon to analyse
- **model** – the model to study
- **hit_weights** – the hit weight needed to compute the cluster score

Returns

list of regular clusters, the special clusters (loners not in cluster and multi systems)

Return type

tuple with 2 elements

- **true_clusters** which is list of [Cluster](#) objects
- **true_loners**: a dict { str function: :class:macsylib.hit.Loner | :class:macsylib.hit.LonerMultiSystem object }

`macsylib.cluster.closest_hit(hit: ModelHit, ref_hits: list[ModelHit]) → ModelHit`

Parameters

- **hit** – the hit
- **ref_hits** – The reference hits. the distance between *hit* and each *ref_hit* will be computed. the closest *ref_hit* will be returned

Returns

The closest *ref_hit* to the hit. If two *ref_hits* are equidistant from the *hit* return those with the lowest position. for instance:

```
position      40  20  60
closest_hit( ref_hit, [H1, H2]
```

will return *H1*

`macsylib.cluster.clusterize_hits_around_key_genes(key_genes: set[str], hits: list[ModelHit], model: Model, hit_weights: HitWeight, rep_info: RepliconInfo) → list[Cluster]`

clusterize hit regarding the distance between them and around key_gene

Parameters

- **hits** (list of `macsylib.model.ModelHit` objects) – the hits to clusterize
- **model** (`macsylib.model.Model` object) – the model to consider
- **hit_weights** (`macsylib.hit.HitWeight` object) – the hit weight to compute the score

Returns

the clusters

Return type

list of `macsylib.cluster.Cluster` objects.

`macsylib.cluster.clusterize_hits_on_distance_only(hits: list[ModelHit], model: Model, hit_weights: HitWeight, rep_info: RepliconInfo) → list[Cluster]`

clusterize hit regarding the distance between them

Parameters

- **hits** – the hits to clusterize
- **model** – the model to consider
- **hit_weights** – the hit weight to compute the score
- **rep_info** – The information on the replicon

Returns

the clusters

`macsylib.cluster.is_a(hit: ModelHit | CoreHit, ref_hits: set[str]) → bool`

Parameters

- **hit** – The hit to check
- **ref_hits** – the gene name of the reference hit

Returns

True if the *hit* belong to the reference hits, False otherwise

`macsylib.cluster.scaffold_to_cluster(cluster_scaffold: list[ModelHit], model: Model, hit_weights: HitWeight) → Cluster`

transform a list of ModelHit in a cluster if the hit colocalize and they are not all neutral and they do not code for same gene add the new cluster to the clusters

Parameters

- **cluster_scaffold** – model hit to transform in cluster
- **model** – The model related to thus cluster
- **hit_weights** – the hit weight to compute scores

Returns

Cluster

`macsylib.cluster.split_cluster_on_key_genes(key_genes: set[str], cluster: Cluster) → list[Cluster]`

split a Cluster containing several key genes to have one cluster per key genes, with their closest hits

For instance if a set of gene clusterize as following (we considering that all gene are 10 genea between next one:

positions	10	20	30	40	50	60	70
genes	A	KG1	B	C	D	KG2	E

The resulting cluster after split around the 2 KG (key genes):

```
c1 = [A, KG1, B, C], c2 = [D, KG2, E]
```

The question is for gene C which is equidistant from KG1 KG2 C will be clustered with the most left cluster

Parameters

- **key_genes** – the gene names which be seed for cluster
- **cluster** – The cluster to split

Returns

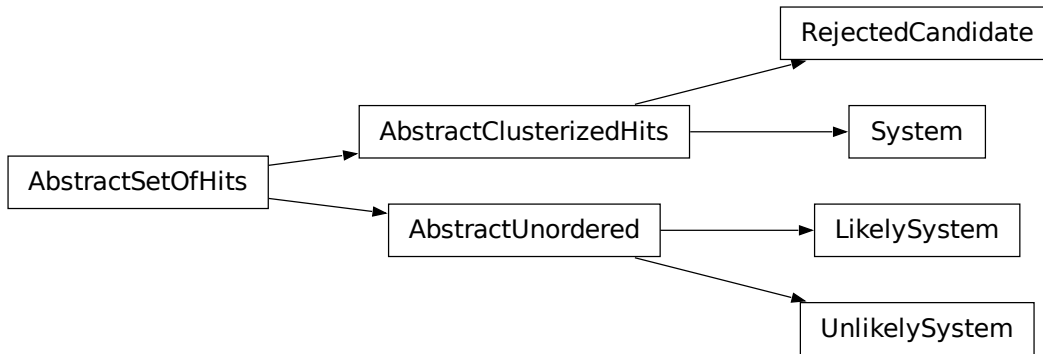
system

This module classes and functions which a given set of hits and a model compute if this set satisfy the model or not

The object which check the compliance of hits to a model is MatchMaker which have 2 sub-classes for ordered and unordered replicons

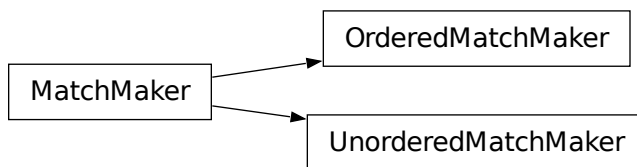
MatchMaker.match method link hit to a model (`macsylib.hit.ValidHit`) and then check if these valid hit satisfy the quorum constraints defined in the model. According this it instanciate a `macsylib.system.System` or `macsylib.system.RejectedCandidate` for ordered replicons or `macsylib.system.LikelySystem` or `macsylib.system.UnlikelySystem` for unordered replicons

below the inheritance diagram:



Warning

The abstract class `macsylib.system.AbstractSetOfHits` is controlled by the metaclass `macsylib.system.MetaSetOfHits` which inject on the fly several private attributes and public properties (see more in `macsylib.system.MetaSetOfHits` documentation)



system reference api

MatchMaker

class `macsylib.system.MatchMaker(model: Model)`

Is an abstract class for (Ordered|Unordered)MatchMaker the *match* class method must be implemented in concrete classes.

__init__(model: Model) → None

Parameters

model – The model used compute genetic constraints

__weakref__

list of weak references to the object

_create_exchangeable_map(genes: list[ModelGene]) → dict[slice(<class 'str'>, <class 'macsylib.gene.ModelGene'>, None)]

create a map between an exchangeable (formly homolog or analog) gene name and it's gene reference

Parameters

genes – The genes to get the exchangeable genes

Returns

a dict with keys are the exchangeable gene_name and the value the reference gene

present_genes() → tuple[list[str], list[str], list[str], list[str]]

Returns

the lists of genes name in model which are present in the replicon (included exchangeable)

tuple of 4 lists for mandatory, accessory, neutral and forbidden

([str gene_name, ...], [str gene_name], [str gene_name], [str gene_name])

sort_hits_by_status(hits: Iterable[ModelHit]) → tuple[list[ModelHit], list[ModelHit], list[ModelHit], list[ModelHit]]

sort *macsylib.hit.ModelHit* according the status of the gene the hit code for.

Parameters

hits – list of *macsylib.hit.ModelHit* object

Returns

the valid hits according their status ([mandatory,], [accessory,], [neutral,], [forbidden])

Raises

MacsylibError – when a gene is not found in the model

OrderedMatchMaker

class *macsylib.system.OrderedMatchMaker*(model: Model, redundancy_penalty: float)

check if a set of hits match the quorum for ordered replicons (ordered_replicon or gembase)

__init__(model: Model, redundancy_penalty: float) → None

Parameters

- **model** – The model used compute genetic constraints
- **redundancy_penalty** – The penalty applied to compute the score when several genes coding for same function are present in same cluster.

match(clusters: Iterable[Cluster]) → System | RejectedCandidate

Check a set of clusters fill model constraints. If yes create a *macsylib.system.System* otherwise create a *macsylib.cluster.RejectedCandidate*.

Parameters

clusters – The list of cluster to check if fit the model

Returns

either a System or a RejectedCandidates

UnorderedMatchMaker

class *macsylib.system.UnorderedMatchMaker*(model: Model)

match(hits: Iterable[ModelHit]) → LikelySystem | UnlikelySystem

Parameters

hits – the hits to check

HitSystemTracker

class macsylib.system.HitSystemTracker(systems: list[System | LikelySystem])

track in which system is implied each hit

__init__(systems: list[System | LikelySystem]) → None

__weakref__

list of weak references to the object

MetaSetOfHits

class macsylib.system.MetaSetOfHits(name, bases, namespace, / (Positional-only parameter separator (PEP 570)), **kwargs)

This metaclass control the AbstractSetOfHits class creation. In this metaclass we inject on the fly several attributes and properties two private attributes and one public property corresponding to each value of `_supported_status` class attribute defined in the concrete classes. for instance for System class

- **the attributes**

- self._mandatory
- self._mandatory_occ
- self._accessory
- self._accessory_occ
- self._neutral
- self._neutral_occ

- **and the properties**

- mandatory
- accessory
- neutral

are automatically injected

The value for attributes `_<status>_occ` are filled by the *count* method which is defined in AbstractSetOfHits

__call__(*args, **kwargs)

Call self as a function.

getter_maker() → Callable

Create a property which allow to access to the gene corresponding of the cat of the model

Parameters

status – the type of gene category to which we create the getter

Returns

unbound method

AbstractSetOfHits

class macsylib.system.**AbstractSetOfHits**(*args, **kwargs)

Is the mother class of System, RejectedCandidates, LikelySystems UnlikelySystem, ...

__init__(model: [Model](#)) → None

__weakref__

list of weak references to the object

count() → None

fill structures one for supported status mandatory, accessory, ... each structure count how many hit for each gene of the model mandatory_occ = { gene_name : [ModelHit, ...] :return: None

property position: tuple[int, int]

Returns

The position of the first and last hit (start: int, end:int), excluded the hit coding for loners. If the system is composed only by loners, used loners to compute position

property replicon_name: str

Returns

The name of the replicon

Return type

str

property wholeness: float

Returns

a score indicating the genes ratio of the model which have at least one hit by default full system is mandatory + accessory ('neutral' genes do not count) but for special corner case it can be specified in model definition (xml) or on the command line

AbstractClusterizedHits

class macsylib.system.**AbstractClusterizedHits**(*args, **kwargs)

Modelize SetOfHits that colocalize.

should be inherited

__init__(model: [Model](#), clusters: [Cluster](#) | list[[Cluster](#)])

fulfilled_function(*genes: [ModelGene](#) | str) → set[str]

Parameters

genes ([macsylib.gene.ModelGene](#) object or string representing the gene name) – The genes which must be tested.

Returns

the common functions between genes and this system.

Return type

set of string

System

class `macsylib.system.System(*args, **kwargs)`

Modeling as system. a system is an occurrence of a given model on a replicon.

__init__(*model*: [Model](#), *clusters*: *list*[[Cluster](#)], *redundancy_penalty*: *float* = 1.5) → None

Parameters

- **model** ([macsylib.model.Model](#) object) – The model which has been used to build this system
- **clusters** (list of [macsylib.cluster.Cluster](#) objects) – The list of cluster that form this system

__str__() → str

Return str(self).

get_hits_encoding_multisystem() → set[[MultiSystem](#)]

Returns

The hits coding for a gene tagged as multi system

get_loners() → set[[Loner](#) | [LonerMultiSystem](#)]

Returns

The True Loners (Loner which not colocalize with another hit) belonging to the systems

get_multisystems() → set[[MultiSystem](#) | [LonerMultiSystem](#)]

Returns

The MultiSystem hit (comming from out system (other cluster or loner) and tag as multisystem)

property hits: *list*[[ModelHit](#)]

Returns

The list of all hits that compose this system

is_compatible(*other*: [System](#)) → bool

Parameters

other – the other systems to test compatibility

Returns

True if other system is compatible with this one. False otherwise. Two systems are compatible if they do not share [macsylib.hit.CoreHit](#) except hit corresponding to a multi_system gene in the model.

Note

This method is used to compute the best combination of systems.

property loci_nb: *int*

Returns

The number of loci of this system (loners are not considered)

Return type

int ≥ 0

property loci_num: `list[int]`

Returns

the number of the corresponding locus for each cluster the cluster made of only one Loner are not considered as a loci so these clusters have a negative locus_num

property multi_loci: `bool`

Returns

True if the systems is encoded in multiple loci. False otherwise

occurrence() \rightarrow `int`

sometimes several systems collocates so they form only one cluster so macsylib build only one system the occurrence is an indicator of how many systems are it's based on the number of occurrence of each mandatory genes The multi_system genes are not take in account.

Returns

a predict number of biologic systems

property score: `float`

Returns

a score take in account * if a hit match for the gene or it is an exchangeable gene * if a hit is duplicated and already present in the system or the cluster * if a hit match for mandatory/accessory gene of the model

Return type

`float`

RejectedCandidate

class `macsylib.system.RejectedCandidate(*args, **kwargs)`

Handle a set of clusters which has been rejected during the `macsylib.system.match()` step This clusters (can be one) does not fill the requirements or contains forbidden genes.

__init__(*model*: `Model`, *clusters*: `list[Cluster]`, *reasons*: `list[str]`) \rightarrow `None`

Parameters

- **model**
- **clusters** – list of clusters. These Clusters should be created with `macsylib.cluster.Cluster` of `macsylib.hit.ModelHit` objects
- **reasons** – the reason why these clusters have been rejected

__str__() \rightarrow `str`

Returns

a string representation of this RejectedCandidates

property hits: `list[ModelHit]`

Returns

The list of all hits that compose this system

property reasons: `list[str]`

Returns

The reason why it has been rejected

AbstractUnordered

class macsylib.system.**AbstractUnordered**(*args, **kwargs)

Technical abstract class to factorize code share between LikelySystem and UnlikelySystem

__init__(model: [Model](#), mandatory_hits: list[[ModelHit](#)], accessory_hits: list[[ModelHit](#)], neutral_hits: list[[ModelHit](#)], forbidden_hits: list[[ModelHit](#)]) → None

Parameters

- **model** – The model which has been used to build this system
- **mandatory_hits** – The list of mandatory hits (encode for a gene tagged as mandatory)
- **accessory_hits** – The list of accessory hits (encode for a gene tagged as accessory)
- **neutral_hits** – The list of neutral hits (encode for a gene tagged as neutral)
- **forbidden_hits** – The list of hits that are forbidden

property accessory_hits: list[[ModelHit](#)]

Returns

The list of accessory hits

property allowed_hits: list[[ModelHit](#)]

Returns

The list of allowed (mandatory, accessory, neutral) hits

property forbidden_hits: list[[ModelHit](#)]

Returns

The list of forbidden hits

property hits: list[[ModelHit](#)]

Returns

The list of all hits sorted by their position

property mandatory_hits: list[[ModelHit](#)]

Returns

The list of mandatory hits

property neutral_hits: list[[ModelHit](#)]

Returns

The list of neutral hits

LikelySystem

class macsylib.system.**LikelySystem**(*args, **kwargs)

” Handle components that fill the quorum requirements defined in model. with no idea about genetic organization (gene cluster) so we cannot take in account forbidden genes

__str__() → str

Returns

a string representation of this LikelySystem

UnlikelySystem

class macsylib.system.UnlikelySystem(*args, **kwargs)

Handle components that not fill the quorum requirements defined in model.

__init__(model: Model, mandatory_hits: list[ModelHit], accessory_hits: list[ModelHit], neutral_hits: list[ModelHit], forbidden_hits: list[ModelHit], reasons: list[str]) → None

Parameters

- **model** – The model which has been used to build this system
- **mandatory_hits** – The list of mandatory hits (encode for a gene tagged as mandatory)
- **accessory_hits** – The list of accessory hits (encode for a gene tagged as accessory)
- **neutral_hits** – The list of neutral hits (encode for a gene tagged as neutral)
- **forbidden_hits** – The list of hits that are forbidden
- **reasons** – the reasons why this set of hits has been rejected

__str__() → str

Returns

a string representation of this UnlikelySystem

property reasons: list[str]

Returns

The reasons why it probably not a system

Return type

list of string

report

A “HMMReport” object represents the results of a Hmmer program search on a dataset with a hidden Markov model protein profile (see [this section](#)). This object has methods to extract and filter Hmmer raw outputs (see [generated output files](#)), and then build Hits relevant for system detection. For matches selected with the filtering parameters, “Hit” objects (macsylib.HMMReport.Hit) are built.

report API reference

HMMReport

class macsylib.report.HMMReport(gene: CoreGene, hmmer_output: str, cfg: Config)

Handle the results from the HMM search. Extract a synthetic report from the raw hmmer output, after having applied a hit filtering. This class is an **abstract class**. There are two implementations of this abstract class depending on whether the input sequence dataset is “ordered” (“gembase” or “ordered_replicon” db_type) or not (“unordered” db_type).

__init__(gene: CoreGene, hmmer_output: str, cfg: Config) → None

Parameters

- **gene** – the gene corresponding to the profile search reported here
- **hmmer_output** – The path to the raw Hmmer output file
- **cfg** – the configuration object

__str__() → str

Returns

string representation of this report

__weakref__

list of weak references to the object

_build_my_db(*hmm_output*: str) → dict[slice(<class 'str'>, None, None)]

Build the keys of a dictionary object to store sequence identifiers of hits.

Parameters

hmm_output (string) – the path to the hmmsearch output to parse.

Returns

a dictionary containing a key for each sequence id of the hits

_fill_my_db(*db*: dict[slice(<class 'str'>, tuple[int, int], None)]) → None

Fill the dictionary with information on the matched sequences

Parameters

db (dict) – the database containing all sequence id of the hits.

abstractmethod _get_replicon_name(*hit_id*: str) → str

This method is used by extract method and must be implemented by concrete class

Parameters

hit_id (str) – the id of the current hit extract from hmm output.

Returns

The name of the replicon

_hit_start(*line*: str) → bool

Parameters

line (string) – the line to parse

Returns

True if it's the beginning of a new hit in Hmmer raw output files. False otherwise

Return type

boolean.

_parse_hmm_body(*hit_id*: str, *gene_profile_lg*: int, *seq_lg*: int, *coverage_threshold*: float, *replicon_name*: str, *position_hit*: int, *i_value_sel*: float, *b_grp*: Iterator) → list[CoreHit]

Parse the raw Hmmer output to extract the hits, and filter them with threshold criteria selected (“coverage_profile” and “i_value_select” command-line parameters)

Parameters

- **hit_id** – the sequence identifier
- **gene_profile_lg** – the length of the profile matched
- **seq_lg** – the length of the sequence
- **coverage_threshold** – the minimal coverage of the profile to be reached in the Hmmer alignment for hit selection.
- **replicon_name** – the identifier of the replicon
- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_value_sel** – the maximal i-value (independent evalule) for hit selection

- **b_grp** (*list of list of strings*) – the Hmmer output lines to deal with (grouped by hit)

Returns

a sequence of hits

Return type

list of `macsylib.report.CoreHit` objects

_parse_hmm_header(*h_grp: Iterator*) → str

Parameters

h_grp (*sequence of string (<itertools._grouper object at 0x7ff9912e3b50>)*) – the sequence of string return by groupby function representing the header of a hit

Returns

the sequence identifier from a set of lines that corresponds to a single hit

Return type

string

best_hit() → *CoreHit* | None

Return the best hit among multiple hits

extract() → None | list[*CoreHit*]

Parse the output file of hmmer compute from an unordered genes base and produced a new synthetic report file.

save_extract() → None

Write the string representation of the extract report in a file. The name of this file is the concatenation of the gene name and of the “res_extract_suffix” from the config object

GeneralHMMReport

class `macsylib.report.GeneralHMMReport`(*gene: CoreGene, hmmer_output: str, cfg: Config*)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to any type of ‘unordered’ datasets.

_get_replicon_name(*hit_id: str*) → str

This method is used by extract method and must be implemented by concrete class

Parameters

hit_id (*str*) – the id of the current hit extract from hmm output.

Returns

The name of the replicon

OrderedHMMReport

class `macsylib.report.OrderedHMMReport`(*gene: CoreGene, hmmer_output: str, cfg: Config*)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to ‘ordered_replicon’ datasets.

_get_replicon_name(*hit_id: str*) → str

This method is used by extract method and must be implemented by concrete class

Parameters

hit_id (*str*) – the id of the current hit extract from hmm output.

Returns

The name of the replicon

GembaseHMMReport

class macsylib.report.**GembaseHMMReport**(*gene*: [CoreGene](#), *hmmer_output*: *str*, *cfg*: [Config](#))

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to ‘gembase’ format datasets.

_get_replicon_name(*hit_id*: *str*) → *str*

This method is used by extract method and must be implemented by concrete class

Parameters

hit_id (*str*) – the id of the current hit extract from hmm output.

Returns

The name of the replicon

search_genes

manage the paralelization of code which execute *in fine hmsearch* to find the genes constituting the models in the input dataset.

search_genes API reference**search_genes**

Manage the hmm step (hmsearch or recover results from previous run) in parallele

macsylib.search_genes.**search_genes**(*genes*: *list*[[ModelGene](#)], *cfg*: [Config](#)) → *list*[[HMMReport](#)]

For each gene of the list, use the corresponding profile to perform an Hmmer search, and parse the output to generate a HMMReport that is saved in a file after CoreHit filtering. These tasks are performed in parallel using threads. The number of workers can be limited by worker_nb directive in the config object or in the command-line with the “-w” option.

Parameters

- **genes** – the genes to search in the input sequence dataset
- **cfg** – the configuration object

macsylib.search_genes.**worker_cpu**(*genes_nb*: *int*, *cfg*: [Config](#)) → *tuple*[*int*, *int*]

Compute the optimum number of worker and cpu per worker The number of worker is set by the user (1 by default 0 means all worker available)

we use one worker per gene if number of workers is greater than number of genes then several cpu can be use by hmsearch to speed up the search step

Parameters

- **genes_nb** – the number of genes to search
- **cfg** – The macsylib configuration

Returns

the number of worker and cpu_per_worker to use

Return type

tuple (*int* worker_nb, *int* cpu_per_worker)

solution

MacSyLib find lot of potential systems for the same model, all these systems are saved in “all_systems.xxx” files. This module allow to explore among of all systems which combination seems to be more probable.

solution API reference

Solution

class `macsylib.solution.Solution(systems: list[System])`

Handle Solution, a solution is a set of compatible Systems

when compare solutions we check the following criteria

1. The number of hits
2. The number of systems
3. The average of wholeness
4. The hits position (is used ti give predictable output for unit tests)

`__eq__(other) → bool`

Return self==value.

`__gt__(other) → bool`

Return self>value.

`__init__(systems: list[System]) → None`

Parameters

systems – The list of system that composed this solution

`__iter__() → Generator`

Solution allow to iterate over the systems

Returns

generator

`__lt__(other) → bool`

Return self<value.

`__weakref__`

list of weak references to the object

`_sorted_systems(systems: list[System]) → list[System]`

sort the systems following the positions of th hits that composed the systems

Parameters

systems (list of `mcsypy.system.System` objects) – the systems to sort

Returns

a sorted copy of the *systems*

Return type

list of `mcsypy.system.System` objects

property `average_wholeness: float`

The average of the systems wholeness

property hits_number: int

The sum of the hits of each system in this solution

property hits_positions: list[int]

The list of position of all hits of the solution

property score: float

The score of this solution

property systems: list[System]

“a sorted list of the *systems* that composed the solution

combine_clusters

`macsylib.solution.combine_clusters(clusters: list[~macsylib.cluster.Cluster], true_loners: dict[slice(<class 'str'>, macsylib.hit.Loner | macsylib.hit.LonerMultiSystem, None)], multi_loci: bool = False) → list[tuple[Cluster]]`

generate the combinations of clusters, with loners and multi systems

Parameters

- **clusters** – the clusters to combines
- **true_loners** (dict the name of the function code by hit gene_ref.alternate_of as key and 1 *macsylib.cluster.Cluster* with the best a *macsylib.hit.Loner* or *macsylib.hit.LonerMultiSystem* hit as value) – the multi-systems hits
- **multi_loci** – True if the model is multi_loci false otherwise

Returns

all available combination of clusters

Return type

List of combination. a combination is a tuple of *macsylib.cluster.Cluster* objects

combine_multisystems

`macsylib.solution.combine_multisystems(rejected_candidates: list[RejectedCandidate], multi_systems: list[Cluster])`

Parameters

- **rejected_candidates**
- **multi_systems** – sequence of *macsylib.cluster.Cluster* each cluster must be composed of only one *macsylib.hit.MultiSystem* object

Returns

list of cluster combination with the multisystem

Return type

`[(macsylib.cluster.Cluster cluster1, cluster2, ...), (macsylib.cluster.Cluster cluster3, cluster4, ...)]`

find_best_solutions

`macsylib.solution.find_best_solutions(systems: list[System]) → tuple[list[Solution], float]`

Among the systems choose the combination of systems which does not share *macsylib.hit.CoreHit* and maximize the sum of systems scores

Parameters

systems – the systems to analyse

Returns

the list of *macsylib.solution.Solution* which represent one best solution then it's score.

Return type

tuple of 2 elements the best solutions and it's score

serialization

This module is a technical module where we can find the different way to serialize the results:

- the Systems found
- The best solutions (best combination of systems)
- The rejected candidates

SystemSerializer

class *macsylib.serialization.SystemSerializer*

handle the different way to serialize a system

__weakref__

list of weak references to the object

TsvSystemSerializer

class *macsylib.serialization.TsvSystemSerializer*

Handle System serialization in tsv format

serialize(*system*: *System*, *hit_system_tracker*: *HitSystemTracker*) → str

:param *macsylib.system.System* system: The system to serialize. :param *hit_system_tracker*: The *hit_system_tracker* which allow to know for each hit

in which system it is implied.

Returns

a serialisation of this system in tabulated separated value format each line represent a hit and have the following structure:

```
replicon\thit_id\tgene_name\thit_pos\tmodel_fqn\tsys_id\tsys_loci\tlocus_num\  
↪tsys_wholeness\tsys_score  
\tsys_occ\thit_gene_ref.alternate_of\thit_status\thit_seq_len\thit_i_eval\thit_  
↪score\thit_profile_cov  
\thit_seq_cov\tit_begin_match\thit_end_match\tcounterpart\tused_in_systems
```

TsvSolutionSerializer

class *macsylib.serialization.TsvSolutionSerializer*

Handle Solution (list of Systems) serialization in tsv format

__weakref__

list of weak references to the object

serialize(*solution*: [Solution](#), *sol_id*: int, *hit_system_tracker*: [HitSystemTracker](#)) → str

Parameters

- **solution** – the solution to serialize
- **sol_id** – the solution identifier
- **hit_system_tracker**

Returns

a serialisation of this solution (a list of systems) in tabulated separated value format each line represent a hit and have the same structure as system serialization [macsylib.serialization.TsvSystemSerializer.serialize\(\)](#) but with an extra column `sol_id` which is a technical id to identify the different solutions.

TsvLikelySystemSerializer

class `macsylib.serialization.TsvLikelySystemSerializer`

Handle potential System from unordered replicon serialization in tsv format

serialize(*system*: [LikelySystem](#), *hit_system_tracker*: [HitSystemTracker](#)) → str

Parameters

- **system** – The likely system to serialize. Used only for unordered db-type
- **hit_system_tracker** – The `hit_system_tracker` which allow to know for each hit in which system it is implied.

Returns

a serialisation of this system in tabulated separated value format each line represent a hit and have the following structure:

```
replicon\thit_id\tgene_name\thit_pos\tmodel_fqn\tsys_id\tsys_wholeness
\thit_gene_ref.alternate_of\thit_status\thit_seq_len\thit_i_eval\thit_score\
→thit_profile_cov
\thit_seq_cov\tit_begin_match\thit_end_match\tused_in_systems
```

Return type

str

TsvRejectedCandidatesSerializer

class `macsylib.serialization.TsvRejectedCandidatesSerializer`

Serialize Rejected Cluster in tsv format

__weakref__

list of weak references to the object

serialize(*candidates*: list[[RejectedCandidate](#)]) → str

Parameters

candidates – list of rejected candidates to serialize

TsvSpecialHitSerializer

class macsylib.serialization.TsvSpecialHitSerializer

Serialize special hits: *macsylib.hit.Loner* and *macsylib.hit.MultiSystem* in tsv format

__weakref__

list of weak references to the object

serialize(*best_hits*: Iterable[Loner] | Iterable[MultiSystem])

Parameters

best_hits (sequence of *macsylib.hit.Loner* or *macsylib.hit.MultiSystem* objects)
– the special hits to serialized

TxtSystemSerializer

class macsylib.serialization.TxtSystemSerializer

Handle System serialization in text

serialize(*system*: System, *hit_system_tracker*: HitSystemTracker) → str

Returns

a string representation of system readable by human

TxtLikelySystemSerializer

class macsylib.serialization.TxtLikelySystemSerializer

Handle System serialization in text

serialize(*system*: LikelySystem, *hit_system_tracker*: HitSystemTracker)

Parameters

- **system** – The likely system to serialize. Used only for unordered db-type
- **hit_system_tracker** – The hit_system_tracker which allow to know for each hit in which system it is implied.

Returns

a string representation of system readable by human

TxtUnikelySystemSerializer

class macsylib.serialization.TxtUnikelySystemSerializer

Handle System serialization in text

serialize(*system*: UnlikelySystem) → str

Parameters

system – The unlikely system to serialize. (used only if db-type is “unordered_replicon”)

Returns

a string representation of system readable by human

database

The “database” object handles the indexes of the sequence dataset in fasta format, and other useful information on the input dataset.

MacSyLib needs to have the length of each sequence and its position in the database to compute some statistics on Hmmer hits. Additionally, for ordered datasets (`db_type = 'gembase' or 'ordered_replicon'`), MacSyLib builds an internal “database” from these indexes to store information about replicons, their begin and end positions, and their topology.

The begin and end positions of each replicon are computed from the sequence file, and the topology from the parsing of the topology file (see *Topology files*).

Thus it also builds an index (with *.idx* suffix) that is stored in the same directory as the sequence dataset. If this file is found in the same folder than the input dataset, MacSyLib will use it. Otherwise, it will build it.

database API reference

Indexes

class `macsylib.database.Indexes`(*cfg*: `Config`)

Handle the indexes for macsylib:

- find the indexes required by macsylib to compute some scores, or build them.

__init__(*cfg*: `Config`) → None

The constructor retrieves the file of indexes in the case they are not present or the user asked for build indexes (`-idx`) Launch the indexes building.

Parameters

cfg (`macsylib.config.Config` object) – the configuration

__iter__() → `Iterator[tuple[str, str, int]]`

Raises

MacsylibError – if the indexes are not build

Returns

an iterator on the indexes

To use it the index must be built.

__weakref__

list of weak references to the object

_build_my_indexes(*index_dir*: `str`) → `str`

Build macsylib indexes. These indexes are stored in a file.

The file format is the following:

- the first line is the path of the sequence-db indexed
- one entry per line, with each line having this format:
- sequence id;sequence length;sequence rank

_index_dir(*build*: `bool = False`) → `str`

search where to store(`build=True`) read indexes

Parameters

build – if check the index-dir permissions to write

Returns

The directory where read or write the indexes

Raises

ValueError – if the directory specify by `--index-dir` option does not exist or if `build = True` `index-dir` is not writable

build(*force: bool = False*) → str

Build the indexes from the sequence data set in fasta format,

Parameters

force – If True, force the index building even if the index files are present in the sequence data set folder

Returns

the path to the index

find_my_indexes() → str | None

Returns

the file of macsylib indexes if it exists in the dataset folder, None otherwise.

Return type

string

RepliconInfo

Module to handle sequences and their indexes

class macsylib.database.**RepliconInfo**(*topology, min, max, genes*)

handle information about a replicon

topology

The type of replicon topology ‘linear or ‘circular’

min

The position of the last gene of the replicon in the sequence dataset.

max

The position of the last gene of the replicon in the sequence dataset.

genes

A list of genes belonging to the replicon. Each genes is representing by a tuple (str seq_id, int length)

genes

Alias for field number 3

max

Alias for field number 2

min

Alias for field number 1

topology

Alias for field number 0

RepliconDB

class `macsylib.database.RepliconDB`(*cfg*: [Config](#))

Stores information (topology, min, max, [genes]) for all replicons in the `sequence_db` the Replicon object must be instantiated only for `sequence_db` of type 'gembase' or 'ordered_replicon'

__contains__(*replicon_name*: *str*) → bool

Parameters

replicon_name – the name of the replicon

Returns

True if `replicon_name` is in the `repliconDB`, false otherwise.

__getitem__(*replicon_name*: *str*) → [RepliconInfo](#)

Parameters

replicon_name – the name of the replicon to get information on

Returns

the `RepliconInfo` for the provided `replicon_name`

Raise

`KeyError` if `replicon_name` is not in `repliconDB`

__init__(*cfg*: [Config](#)) → None

Parameters

cfg ([macsylib.config.Config](#) object) – The configuration object

Note

This class can be instantiated only if the `db_type` is 'gembase' or 'ordered_replicon'

__weakref__

list of weak references to the object

_fill_gembase_min_max(*topology*: *dict*[*slice*(<class 'str'>, *typing.Literal*['linear', 'ciircular'], None)],
default_topology: *~typing.Literal*['linear', 'ciircular']) → None

For each `replicon_name` of a gembase dataset, it fills the internal dictionary with a namedtuple `RepliconInfo`

Parameters

- **topology** – the topologies for each replicon (parsed from the file specified with the option `--topology-file`)
- **default_topology** – the topology provided by the `config.replicon_topology`

_fill_ordered_min_max(*default_topology*: *Literal*['linear', 'ciircular'] | None = None) → None

For the `replicon_name` of the `ordered_replicon` sequence base, fill the internal dict with `RepliconInfo`

Parameters

default_topology (*string*) – the topology provided by `config.replicon_topology`

_fill_topology() → *dict*[*slice*(<class 'str'>, <class 'str'>, None)]

Fill the internal dictionary with min and max positions for each `replicon_name` of the `sequence_db`

get(*replicon_name*: str, *default*: Any = None) → *RepliconInfo*

Parameters

- **replicon_name** – the name of the replicon to get information
- **default** – the value to return if the replicon_name is not in the RepliconDB

Returns

the RepliconInfo for replicon_name if replicon_name is in the repliconDB, else default. If default is not given, it is set to None, so that this method never raises a KeyError.

guess_if_really_gembase() → bool

Count the number of replicon with only one sequence if this number is above a threshold may be it's not gembase. for instance the following sequence have id compliant with the gembase id syntax but it's not it only contains one replicon ('ordered replicon')

```
>1E10S0A0cP00_0010 D GTG TGA 483 2027 Valid dnaA 1545 _PA0001_NP_064721.1_ PA0001 1 483
2027
MSVELWQQCVDLLRDELPSQQFNTWIRPLQVEAEGDELRVYAPNRFVLDW
>0200S001A0c_0P1E0 D ATG TAA 2056 3159 Valid dnaN 1104 _PA0002_NP_064722.1_ PA0002 1
2056 3159
MHFTIQREALLKPLQLVAGVVERRQTLPLVLSNVLLVVEGQQLSLTGTDL
>0000310E00S0c_1PA D ATG TGA 3169 4278 Valid recF 1110 _PA0003_NP_064723.1_ PA0003 1
3169 4278
MSLTRSVTAVRNLHPVTLSPSPRINILYGDNGSGKTSVLEAIHLLGLAR
>c_01000A0PS00014E D ATG TGA 4275 6695 Valid gyrB 2421 _PA0004_NP_064724.1_ PA0004 1
4275 6695
MSENNTYDSSSIKVLKGLDAVRKRPGMYIGDITDDGTGLHHMVFEVVDNSI
>07700ES100A0cP01_ C ATG TGA 91521 94826 Valid icmF1 3306 _PA0077_NP_248767.1_ PA0077
1 91521 94826
MQSLAEVSAPDAASVAT
```

Returns

False if most replicon contains only one sequence, True otherwise

items() → list[tuple[str, *RepliconInfo*]]

Returns

a copy of the RepliconDB as a list of (replicon_name, RepliconInfo) pairs

iteritems() → Iterator[tuple[str, *RepliconInfo*]]

Returns

an iterator over the RepliconDB as a list (replicon_name, RepliconInfo) pairs

replicon_infos() → list[*RepliconInfo*]

Returns

a copy of the RepliconDB as list of replicons info

Return type

RepliconInfo instance

replicon_names() → list[str]

Returns

a copy of the RepliconDB as a list of replicon_names

fasta_iter

macsylib.database.**fasta_iter**(*fasta_file: TextIO*) → Iterator[tuple[str, str, int]]

Parameters

fasta_file – the file containing all input sequences in fasta format.

Author

<http://biostar.stackexchange.com/users/36/brentp>

Returns

for a given fasta file, it returns an iterator which yields tuples (string id, string comment, int sequence length)

errors

The errors specific to macsylib and macsydata

error API reference

error

Manage MacSyLib specific errors

exception macsylib.error.**EmptyFileError**

Raised when fasta file does not contains sequences

exception macsylib.error.**MacsyDataLimitError**

Raised when the maximum number of GitHub api call is reached

exception macsylib.error.**MacsydataError**

Raised when error is encounter during model package handling

exception macsylib.error.**MacsylibError**

The base class for MacSyLib specific exceptions.

__weakref__

list of weak references to the object

exception macsylib.error.**ModelInconsistencyError**

Raised when a definition model is not consistent.

exception macsylib.error.**OptionError**

Raised when command line option is not set properly

exception macsylib.error.**SystemDetectionError**

Raised when the detection of systems from Hits encountered a problem.

exception macsylib.error.**Timeout**

Raised when best solution reach the timeout

utils

Here some useful functions in the rest of macsylib code

utils API reference

get_def_to_detect

`macsylib.utils.get_def_to_detect(models: list[tuple[str, tuple[str]]], model_registry: ModelRegistry) → tuple[list[DefinitionLocation], str, str]`

Parameters

- **models** (list of tuple with the following structure: `[('model_fqn', ('def1', 'def2', ...)), ('model_2', ('def1', ...)), ...]`) – the list of models to detect as returned by `config.models`.
- **model_registry** – the models registry for this run.

Returns

the definitions to parse

Raises

ValueError – if a model name provided in `models` is not in `model_registry`.

get_replicon_names

`macsylib.utils.get_replicon_names(genome_path, db_type) → list[str]`

threads_available

`macsylib.utils.threads_available() → int`

Returns

The maximal number of threads available. It's nice with cluster scheduler or linux. On Mac it uses the number of physical cores

parse_time

`macsylib.utils.parse_time(user_time: int | str) → int`

parse user-friendly time and return it in seconds user time supports units as s h m d for sec min hour day or a combination of them 1h10m50s means 1 hour 10 minutes 50 seconds all terms will be converted in seconds and added

Parameters

user_time

Returns

seconds

Raise

ValueError if `user_time` is not parseable

package

Allow to handles model package either on localhost or from a remote location. the model packages can be stored in github organization to be downloaded and installed locally. The classes below are used by `msl_data`, which is the entry point to manipulate models package.

package API reference

AbstractModelIndex

class macylib.model_package.**AbstractModelIndex**(*cache: str | None = None*)

This the base class for ModelIndex. This class cannot be implemented, it must be subclassed

__init__(*cache: str | None = None*) → None

__weakref__

list of weak references to the object

unarchive_package(*path: str*) → str

Unarchive and uncompress a package under *<remote cache>/<organization name>/<package name>/<vers>/<package name>*

Parameters

path (*str*)

Returns

The path to the package

LocalModelIndex

class macylib.model_package.**LocalModelIndex**(*cache: str | None = None*)

It allow to manage installation from a local package (tarball)

__init__(*cache: str | None = None*) → None

RemoteModelIndex

class macylib.model_package.**RemoteModelIndex**(*org: str = 'macy-models', cache: str | None = None*)

This class allow to interact with ModelIndex on github

__init__(*org: str = 'macy-models', cache: str | None = None*) → None

Parameters

org – The name of the organization on github where are stored the models

_url_json(*url: str*) → dict

Get the url, deserialize the data as json

Parameters

url (*str*) – the url to download

Returns

the json corresponding to the response url

download(*pack_name: str, vers: str, dest: str | None = None*) → str

Download a package from a GitHub repos and save it as *<remote cache>/<organization name>/<package name>/<vers>.tar.gz*

Parameters

- **pack_name** (*str*) – the name of the package to download
- **vers** (*str*) – the version of the package to download
- **dest** (*str*) – The path to the directory where save the package This directory must exist
If dest is None, the macylib cache will be used

Returns

The package archive path.

get_metadata(*pack_name: str, vers: str = 'latest'*) → dict

Fetch the metadata_path from a remote package

Parameters

- **pack_name** (*str*) – The package name
- **vers** (*str*) – The package version

Returns

the metadata_path corresponding to this package/version

Return type

dictionary corresponding of the yaml parsing of the metadata_path file.

list_package_vers(*pack_name: str*) → list[str]

List all available versions from GitHub model repos for a given package

Parameters

pack_name (*str*) – the name of the package

Returns

the list of the versions

list_packages() → list[str]

list all model packages available on a model repos

Returns

The list of package names.

remote_exists() → bool

check if the remote exists and is an organization

Returns

True if the Remote url point to a GitHub Organization, False otherwise

ModelPackage

class macsylib.model_package.**ModelPackage**(*path: str*)

This class Modelize a package of Models a package is a directory with the name of the models family it must contain at least - a subdirectory definitions - a subdirectory profiles - a file metadata.yml it is also recommended to add a file for licensing and copyright and a README. for further explanation see documentation: modeler guide > package

__init__(*path: str*) → None

Parameters

path (*str*) – The of the package root directory

__weakref__

list of weak references to the object

_check_metadata() → tuple[list[str], list[str]]

Check the QA of package metadata_path

Returns

errors and warnings

Return type

tuple of 2 lists ([str error_1, ...], [str warning_1, ...])

_check_model_conf() → tuple[list[str], list[str]]

check if a model configuration file is present in the package (model_conf.xml) if the syntax of this file is good.

Returns**_check_model_consistency()** → tuple[list, list]

check if each xml seems well write, each genes have an associated profile, etc.

Returns**_check_profiles(profile_suffix='.hmm')**

check if there is only one profile per hmm file

Returns**Return type****_check_structure()** → tuple[list[str], list[str]]

Check the QA structure of the package

Returns

errors and warnings

Return type

tuple of 2 lists ([str error_1, ...], [str warning_1, ...])

_find_readme() → str | None

find the README file

Returns

The path to the README file or None if there is no file.

_load_metadata() → dict[slice(<class 'str'>, <class 'str'>, None)]

Open the metadata_path file and de-serialize it's content :return:

check() → tuple[list[str], list[str]]

Check the QA of this package

help() → str

return the content of the README file

info() → str**Returns**

some information about the package

property metadata: dict[slice(<class 'str'>, <class 'str'>, None)]**Returns**

The parsed metadata as a dict

scripts

The are 4 entry points.

- macydata: which allow to manage the models
- macyprofile: an utility dedicated to modelers which gather information about hmmer output

API reference

macsydata

This is the entypoint to the `msl_data` command `mmsl_data` allow the user to manage the MacSylib models

```
macsylib.scripts.macsydata._find_all_installed_packages(models_dir: str | None = None,  
                                                         package_name: str = 'macsylib') →  
                                                         ModelRegistry
```

Parameters

- **models_dir** – The path where packages can be find.
- **package_name** – the name of the high level tool that embed macsylib

Returns

all models installed

```
macsylib.scripts.macsydata._find_installed_package(model_pack_name: str, models_dir: str | None =  
                                                    None, package_name: str = 'macsylib') →  
                                                    ModelLocation | None
```

search if a package names *pack_name* is already installed

Parameters

- **model_pack_name** – the name of the family model to search
- **models_dir** – The path where package can be find.
- **package_name** – the name of the high level tool that embed macsylib, for instance: ‘macsyfinder’

Returns

The model location corresponding to the *pack_name*

```
macsylib.scripts.macsydata._get_remote_available_versions(model_pack_name: str, org: str) →  
                                                         list[str]
```

Ask the organization *org* the available version for the package *pack_name* :param *model_pack_name*: the name of the models package :param *org*: The remote organization to query :return: list of available version for the package

```
macsylib.scripts.macsydata._search_in_desc(pattern: str, remote: RemoteModelIndex, m_packages:  
                                           list[str], match_case: bool = False) → tuple[str, str, str]
```

Parameters

- **pattern** – the substring to search packages descriptions
- **remote** – the uri of the macsy-models index
- **m_packages** – list of model packages to search in
- **match_case** – True if the search is case-sensitive, False otherwise

Returns

```
macsylib.scripts.macsydata._search_in_pack_name(pattern: str, remote: RemoteModelIndex,  
                                                  m_packages: list[str], match_case: bool = False) →  
                                                  list[tuple[str, str, dict]]
```

Parameters

- **pattern** – the substring to search packages names

- **remote** – the uri of the macsy-models index
- **m_packages** – list of model packages to search in
- **match_case** – True if the search is case-sensitive, False otherwise

Returns

`macsylib.scripts.macsydata.build_arg_parser(header: str, version: str, package_name: str = 'macsylib', tool_name: str = 'msl_data') → ArgumentParser`

Build argument parser.

Parameters

- **header** – the header of console script
- **args** – The arguments provided on the command line
- **package_name** – the name of the higher package that embed the macsylib (eg 'macsyfinder')
- **tool_name** – the name of this tool as it appear in pyproject.toml

Returns

The arguments parsed

`macsylib.scripts.macsydata.cmd_name(args: Namespace) → str`

Return the name of the command being executed (scriptname + operation).

Example

`msl_data uninstall`

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_available(args: Namespace) → None`

List Models available on macsy-models :param args: the arguments passed on the command line :return: None

`macsylib.scripts.macsydata.do_check(args: Namespace) → None`

Parameters

args – the arguments passed on the command line

Return type

None

`macsylib.scripts.macsydata.do_cite(args: Namespace) → None`

How to cite an installed model.

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_download(args: Namespace) → str | None`

Download tarball from remote models' repository.

Parameters

args (argparse.Namespace object) – the arguments passed on the command line

`macsylib.scripts.macsydata.do_freeze(args: Namespace) → None`

display all models installed with their respective version, in requirement format.

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_help(args: Namespace) → None`

Display on stdout the content of readme file if the readme file does not exist display a message to the user see `macsylib.package.help()`

Parameters

args – the arguments passed on the command line (the package name)

Returns

None

Raises

ValueError – if the package name is not known.

`macsylib.scripts.macsydata.do_info(args: Namespace) → None`

Show information about installed model.

Parameters

args – the arguments passed on the command line

Raises

ValueError – if the package is not found locally

`macsylib.scripts.macsydata.do_init_package(args: Namespace) → None`

Create a template for data package

- skeleton for metadata.yml
- definitions directory with a skeleton of models.xml
- profiles directory
- skeleton for README.md file
- COPYRIGHT file (if holders option is set)
- LICENSE file (if model_license option is set)

Parameters

args – The parsed commandline subcommand arguments

Returns

None

`macsylib.scripts.macsydata.do_install(args: Namespace) → None`

Install new models in macsylib local models repository.

Parameters

args – the arguments passed on the command line

Raises

- **RuntimeError** – if there is problem is installed package
- **ValueError** – if the package and/or version is not found

`macsylib.scripts.macsydata.do_list(args: Namespace) → None`

List installed models.

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_search(args: Namespace) → None`

Search macsy-models for Model in a remote index. by default search in package name, if option -S is set search also in description by default the search is case-insensitive except if option `-match-case` is set.

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_show_definition(args: Namespace) → None`

display on stdout the definition if only a package or sub-package is specified display all model definitions in the corresponding package or subpackage

for instance

TXSS+/bacterial T6SSii T6SSiii

display models *TXSS+/bacterial/T6SSii* and *TXSS+/bacterial/T6SSiii*

TXSS+/bacterial all or *TXSS+/bacterial*

display all models contains in *TXSS+/bacterial subpackage*

Parameters

args – the arguments passed on the command line

`macsylib.scripts.macsydata.do_show_package(args: Namespace) → None`

Display the structure of an installed model package. The family, sub families and models in tree-like format

Parameters

args – the passed on the command line (the package name)

Returns

None

Raises

ValueError – if the package is not found.

`macsylib.scripts.macsydata.do_uninstall(args: Namespace) → None`

Remove models from macsylib local models repository.

Parameters

args – the arguments passed on the command line

Raises

ValueError – if the package is not found locally

`macsylib.scripts.macsydata.init_logger(level: Literal['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'] | int = 'INFO', out: bool = True) → Logger`

Parameters

- **level** – The logger threshold could be a positive int or string among: 'CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG'
- **out** – if the log message must be displayed

Returns

logger

```

macsylib.scripts.macsydata.main(args: list[str] = None, header: str = '\n\n * *\n* * * * *\n* * * * *\n
* *\n * _ * _ * *\n * _ _ _ _ _ | | _ | | _ _ | _ _ \n | ' _ ' \V _ | | /
_ ' \ _ ' | _ / _ ' \n | | | | | \ _ \ | | ( | | ( | | | ( | \n | | | |
| | _ / | _ _ _ \ _ _ \ _ _ \ _ _ \ _ _ \n * | _ _ | * *\n * * * * *\n
* * * * *\n* *\n\nmssl_data - Model Management Too\n",
version="mssl_data 1.0.3 \nPython 3.13.5 (main, Aug 9 2025, 20:37:54)
[GCC 14.3.0]\n\nMacSyLib is distributed under the terms of the GNU
General Public License (GPLv3).\nSee the COPYING file for details.\n\nIf
you use this software please cite:\nNéron, Bertrand; Denise, Rémi; Coluzzi,
Charles; Touchon, Marie; Rocha, Eduardo P.C.; Abby, Sophie
S.\nMacSyFinder v2: Improved modelling and search engine to identify
molecular systems in genomes.\nPeer Community Journal, Volume 3
(2023), article no. e28. doi : 10.24072/pcjour-
nal.250.\nhttps://peercommunityjournal.org/articles/10.24072/pcjournal.250/\nand
don't forget to cite models used:\nmacsydata cite <model>\n",
package_name: str = 'macsylib', tool_name='mssl_data') → None

```

Main entry point.

Parameters

- **args** – the arguments passed on the command line (before parsing)
- **header** – the header of console script
- **package_name** – the name of the higher package that embed the macsylib (eg 'macsyfinder')
- **tool_name** – the name of this tool as it appear in pyproject.toml

macsylib.scripts.macsydata.**verbosity_to_log_level**(verbosity: int) → int

transform the number of -v option in loglevel :param verbosity: number of -v option on the command line :return: an int corresponding to a logging level

macsyprofile

class macsylib.scripts.macsyprofile.**HmmProfile**(gene_name: str, gene_profile_lg: int, hmmer_output: str, cfg: [Config](#))

Handle the HMM output files

__init__(gene_name: str, gene_profile_lg: int, hmmer_output: str, cfg: [Config](#))

Parameters

- **gene_name** – the name of the gene corresponding to the profile search reported here
- **hmmer_output** – The path to the raw Hmmer output file
- **cfg** – the configuration object

__weakref__

list of weak references to the object

_build_my_db(hmmer_output: str) → dict[slice(<class 'str'>, None, None)]

Build the keys of a dictionary object to store sequence identifiers of hits.

Parameters

hmmer_output – the path to the hmmersearch output to parse.

Returns

a dictionary containing a key for each sequence id of the hits

_fill_my_db(*db: dict[slice(<class 'str'>, tuple[int, int], None)]*) → None

Fill the dictionary with information on the matched sequences

Parameters

db – the database containing all sequence id of the hits.

_hit_start(*line: str*) → bool

Parameters

line – the line to parse

Returns

True if it's the beginning of a new hit in Hmmer raw output files. False otherwise

_parse_hmm_body(*hit_id: str, gene_profile_lg: int, seq_lg: int, coverage_threshold: float, replicon_name: str, position_hit: int, i_value_sel: float, b_grp: list[list[str]]*) → list[[CoreHit](#)]

Parse the raw Hmmer output to extract the hits, and filter them with threshold criteria selected (“coverage_profile” and “i_value_select” command-line parameters)

Parameters

- **hit_id** – the sequence identifier
- **gene_profile_lg** – the length of the profile matched
- **seq_lg** – the length of the sequence
- **coverage_threshold** – the minimal coverage of the profile to be reached in the Hmmer alignment for hit selection.
- **replicon_name** – the identifier of the replicon
- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_value_sel** – the maximal i-value (independent evalule) for hit selection
- **b_grp** – the Hmmer output lines to deal with (grouped by hit)

Returns

a sequence of hits

_parse_hmm_header(*h_grp: str*) → str

Parameters

h_grp – the sequence of string return by groupby function representing the header of a hit

Returns

the sequence identifier from a set of lines that corresponds to a single hit

parse() → list[[LightHit](#)]

parse a hmm output file and extract all hits and do some basic computation (coverage profile)

Returns

The list of extracted hits

```
class macsylib.scripts.macsyprofile.LightHit(gene_name: str, id: str, seq_length: int, replicon_name:
                                             str, position: int, i_eval: float, score: float,
                                             profile_coverage: float, sequence_coverage: float,
                                             begin_match: int, end_match: int)
```

Handle hmm hits

__eq__(*other*)

Return self==value.

```
__init__(gene_name: str, id: str, seq_length: int, replicon_name: str, position: int, i_eval: float, score: float, profile_coverage: float, sequence_coverage: float, begin_match: int, end_match: int) → None
```

```
__repr__()
```

Return repr(self).

```
__str__() → str
    Return str(self).
```

__weakref__
list of weak references to the object

```
macsylib.scripts.macsyprofile.get_gene_name(path: str, suffix: str) → str
```

Parameters

- **path** – The path to the hmm output to analyse
- **suffix** – the suffix of the hmm output file

Returns

the name of the analysed gene

```
macsylib.scripts.macsyprofile.get_profile_len(path: str) → int
```

Parse the HMM profile to extract the length and the presence of GA bit threshold

Parameters

path – The path to the hmm profile used to produce the hmm search output to analyse

Returns

the length, presence of ga bit threshold

[illegible]

Parameters

tool_name – The name of the high level tool

Returns

the long description of the macsylib version

```
macsylib.scripts.macsyprofile.init_logger(level: Literal['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'] | int = 'INFO', out: bool = True)
```

Parameters

- **level** – The logger threshold could be a positive int or string among: ‘CRITICAL’, ‘ERROR’, ‘WARNING’, ‘INFO’, ‘DEBUG’
- **out** – if the log message must be displayed

Returns

logger

[illegible]

main entry point

Parameters

- **args** – the arguments passed on the command line without the program name
- **package_name** – the name of the higher package that embed the macsylib (eg ‘macsyfinder’)
- **tool_name** – the name of this tool as it appear in pyproject.toml
- **log_level** – the output verbosity

macsylib.scripts.macsyprofile.**parse_args**(*header: str, args: list[str], package_name='macsylib', tool_name: str = 'msl_profile'*) → Namespace

Build argument parser.

Parameters

- **header** – the header of console scriot
- **args** – The arguments provided on the command line
- **package_name** – the name of the higher package that embed the macsylib (eg ‘macsyfinder’)
- **tool_name** – the name of this tool as it appear in pyproject.toml

Returns

The arguments parsed

macsylib.scripts.macsyprofile.**result_header**(*cmd: list[str], model: str, model_vers: str, tool_name='msl_profile'*) → str

Parameters

cmd – the command use dto launch this analyse

Model

The name of model family

Model_vers

The version of the model

Returns

The header of the result file

macsylib.scripts.macsyprofile.**verbosity_to_log_level**(*verbosity: int*) → int

transform the number of -v option in loglevel :param verbosity: number of -v option on the command line :return: an int corresponding to a logging level

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

m

- `macsylib.cluster`, [126](#)
- `macsylib.database`, [148](#)
- `macsylib.definition_parser`, [102](#)
- `macsylib.error`, [151](#)
- `macsylib.model_conf_parser`, [96](#)
- `macsylib.scripts.macsydata`, [156](#)
- `macsylib.scripts.macsyprofile`, [160](#)
- `macsylib.search_genes`, [141](#)

Symbols

- `__call__()` (*macsylib.system.MetaSetOfHits* method), 133
- `__contains__()` (*macsylib.cluster.Cluster* method), 124
- `__contains__()` (*macsylib.database.RepliconDB* method), 149
- `__contains__()` (*macsylib.gene.GeneBank* method), 109
- `__contains__()` (*macsylib.model.ModelBank* method), 105
- `__delattr__()` (*macsylib.hit.HitWeight* method), 122
- `__eq__()` (*macsylib.hit.CoreHit* method), 116
- `__eq__()` (*macsylib.hit.HitWeight* method), 122
- `__eq__()` (*macsylib.hit.ModelHit* method), 119
- `__eq__()` (*macsylib.model.Model* method), 106
- `__eq__()` (*macsylib.registries.DefinitionLocation* method), 100
- `__eq__()` (*macsylib.registries.ModelLocation* method), 98
- `__eq__()` (*macsylib.scripts.macsyprofile.LightHit* method), 161
- `__eq__()` (*macsylib.solution.Solution* method), 142
- `__getitem__()` (*macsylib.database.RepliconDB* method), 149
- `__getitem__()` (*macsylib.gene.GeneBank* method), 109
- `__getitem__()` (*macsylib.model.ModelBank* method), 105
- `__getitem__()` (*macsylib.registries.ModelRegistry* method), 97
- `__gt__()` (*macsylib.hit.CoreHit* method), 118
- `__gt__()` (*macsylib.hit.ModelHit* method), 119
- `__gt__()` (*macsylib.model.Model* method), 106
- `__gt__()` (*macsylib.registries.DefinitionLocation* method), 100
- `__gt__()` (*macsylib.registries.ModelLocation* method), 98
- `__gt__()` (*macsylib.solution.Solution* method), 142
- `__hash__()` (*macsylib.gene.CoreGene* method), 110
- `__hash__()` (*macsylib.gene.ModelGene* method), 111
- `__hash__()` (*macsylib.hit.CoreHit* method), 118
- `__hash__()` (*macsylib.hit.HitWeight* method), 123
- `__hash__()` (*macsylib.hit.ModelHit* method), 119
- `__hash__()` (*macsylib.model.Model* method), 106
- `__hash__()` (*macsylib.registries.DefinitionLocation* method), 100
- `__init__()` (*macsylib.cluster.Cluster* method), 124
- `__init__()` (*macsylib.config.Config* method), 91
- `__init__()` (*macsylib.config.MacsyDefaults* method), 90
- `__init__()` (*macsylib.database.Indexes* method), 147
- `__init__()` (*macsylib.database.RepliconDB* method), 149
- `__init__()` (*macsylib.definition_parser.DefinitionParser* method), 103
- `__init__()` (*macsylib.gene.CoreGene* method), 110
- `__init__()` (*macsylib.gene.Exchangeable* method), 113
- `__init__()` (*macsylib.gene.GeneBank* method), 110
- `__init__()` (*macsylib.gene.ModelGene* method), 111
- `__init__()` (*macsylib.hit.AbstractCounterpartHit* method), 120
- `__init__()` (*macsylib.hit.CoreHit* method), 118
- `__init__()` (*macsylib.hit.HitWeight* method), 123
- `__init__()` (*macsylib.hit.Loner* method), 121
- `__init__()` (*macsylib.hit.LonerMultiSystem* method), 122
- `__init__()` (*macsylib.hit.ModelHit* method), 119
- `__init__()` (*macsylib.hit.MultiSystem* method), 121
- `__init__()` (*macsylib.model.Model* method), 106
- `__init__()` (*macsylib.model.ModelBank* method), 106
- `__init__()` (*macsylib.model_conf_parser.ModelConfParser* method), 96
- `__init__()` (*macsylib.model_package.AbstractModelIndex* method), 153
- `__init__()` (*macsylib.model_package.LocalModelIndex* method), 153
- `__init__()` (*macsylib.model_package.ModelPackage* method), 154
- `__init__()` (*macsylib.model_package.RemoteModelIndex* method), 153
- `__init__()` (*macsylib.profile.Profile* method), 115
- `__init__()` (*macsylib.profile.ProfileFactory* method), 114
- `__init__()` (*macsylib.registries.DefinitionLocation* method), 100

<code>__init__()</code> (<i>macsylib.registries.ModelLocation method</i>), 98	<code>__str__()</code> (<i>macsylib.profile.Profile method</i>), 115
<code>__init__()</code> (<i>macsylib.registries.ModelRegistry method</i>), 97	<code>__str__()</code> (<i>macsylib.registries.DefinitionLocation method</i>), 100
<code>__init__()</code> (<i>macsylib.report.HMMReport method</i>), 138	<code>__str__()</code> (<i>macsylib.registries.ModelLocation method</i>), 98
<code>__init__()</code> (<i>macsylib.scripts.macsyprofile.HmmProfile method</i>), 160	<code>__str__()</code> (<i>macsylib.registries.ModelRegistry method</i>), 97
<code>__init__()</code> (<i>macsylib.scripts.macsyprofile.LightHit method</i>), 161	<code>__str__()</code> (<i>macsylib.report.HMMReport method</i>), 138
<code>__init__()</code> (<i>macsylib.solution.Solution method</i>), 142	<code>__str__()</code> (<i>macsylib.scripts.macsyprofile.LightHit method</i>), 162
<code>__init__()</code> (<i>macsylib.system.AbstractClusterizedHits method</i>), 134	<code>__str__()</code> (<i>macsylib.system.LikelySystem method</i>), 137
<code>__init__()</code> (<i>macsylib.system.AbstractSetOfHits method</i>), 134	<code>__str__()</code> (<i>macsylib.system.RejectedCandidate method</i>), 136
<code>__init__()</code> (<i>macsylib.system.AbstractUnordered method</i>), 137	<code>__str__()</code> (<i>macsylib.system.System method</i>), 135
<code>__init__()</code> (<i>macsylib.system.HitSystemTracker method</i>), 133	<code>__str__()</code> (<i>macsylib.system.UnlikelySystem method</i>), 138
<code>__init__()</code> (<i>macsylib.system.MatchMaker method</i>), 131	<code>__weakref__</code> (<i>macsylib.cluster.Cluster attribute</i>), 125
<code>__init__()</code> (<i>macsylib.system.OrderedMatchMaker method</i>), 132	<code>__weakref__</code> (<i>macsylib.config.Config attribute</i>), 91
<code>__init__()</code> (<i>macsylib.system.RejectedCandidate method</i>), 136	<code>__weakref__</code> (<i>macsylib.config.MacsyDefaults attribute</i>), 91
<code>__init__()</code> (<i>macsylib.system.System method</i>), 135	<code>__weakref__</code> (<i>macsylib.config.NoneConfig attribute</i>), 96
<code>__init__()</code> (<i>macsylib.system.UnlikelySystem method</i>), 138	<code>__weakref__</code> (<i>macsylib.database.Indexes attribute</i>), 147
<code>__iter__()</code> (<i>macsylib.database.Indexes method</i>), 147	<code>__weakref__</code> (<i>macsylib.database.RepliconDB attribute</i>), 149
<code>__iter__()</code> (<i>macsylib.gene.GeneBank method</i>), 110	<code>__weakref__</code> (<i>macsylib.definition_parser.DefinitionParser attribute</i>), 103
<code>__iter__()</code> (<i>macsylib.model.ModelBank method</i>), 106	<code>__weakref__</code> (<i>macsylib.error.MacsylibError attribute</i>), 151
<code>__iter__()</code> (<i>macsylib.solution.Solution method</i>), 142	<code>__weakref__</code> (<i>macsylib.gene.CoreGene attribute</i>), 110
<code>__len__()</code> (<i>macsylib.model.ModelBank method</i>), 106	<code>__weakref__</code> (<i>macsylib.gene.GeneBank attribute</i>), 110
<code>__len__()</code> (<i>macsylib.profile.Profile method</i>), 115	<code>__weakref__</code> (<i>macsylib.gene.ModelGene attribute</i>), 111
<code>__lt__()</code> (<i>macsylib.hit.CoreHit method</i>), 118	<code>__weakref__</code> (<i>macsylib.hit.CoreHit attribute</i>), 118
<code>__lt__()</code> (<i>macsylib.hit.ModelHit method</i>), 119	<code>__weakref__</code> (<i>macsylib.hit.HitWeight attribute</i>), 123
<code>__lt__()</code> (<i>macsylib.model.Model method</i>), 107	<code>__weakref__</code> (<i>macsylib.hit.ModelHit attribute</i>), 119
<code>__lt__()</code> (<i>macsylib.registries.DefinitionLocation method</i>), 100	<code>__weakref__</code> (<i>macsylib.model.Model attribute</i>), 107
<code>__lt__()</code> (<i>macsylib.registries.ModelLocation method</i>), 98	<code>__weakref__</code> (<i>macsylib.model.ModelBank attribute</i>), 106
<code>__lt__()</code> (<i>macsylib.solution.Solution method</i>), 142	<code>__weakref__</code> (<i>macsylib.model_conf_parser.ModelConfParser attribute</i>), 96
<code>__repr__()</code> (<i>macsylib.hit.HitWeight method</i>), 123	<code>__weakref__</code> (<i>macsylib.model_package.AbstractModelIndex attribute</i>), 153
<code>__repr__()</code> (<i>macsylib.registries.ModelLocation method</i>), 98	<code>__weakref__</code> (<i>macsylib.model_package.ModelPackage attribute</i>), 154
<code>__repr__()</code> (<i>macsylib.scripts.macsyprofile.LightHit method</i>), 162	<code>__weakref__</code> (<i>macsylib.profile.Profile attribute</i>), 115
<code>__setattr__()</code> (<i>macsylib.hit.HitWeight method</i>), 123	<code>__weakref__</code> (<i>macsylib.profile.ProfileFactory attribute</i>), 115
<code>__str__()</code> (<i>macsylib.cluster.Cluster method</i>), 124	<code>__weakref__</code> (<i>macsylib.registries.DefinitionLocation attribute</i>), 100
<code>__str__()</code> (<i>macsylib.gene.GeneStatus method</i>), 114	<code>__weakref__</code> (<i>macsylib.registries.ModelLocation attribute</i>), 98
<code>__str__()</code> (<i>macsylib.gene.ModelGene method</i>), 111	<code>__weakref__</code> (<i>macsylib.registries.ModelRegistry attribute</i>), 97
<code>__str__()</code> (<i>macsylib.hit.AbstractCounterpartHit method</i>), 120	<code>__weakref__</code> (<i>macsylib.report.HMMReport attribute</i>), 139
<code>__str__()</code> (<i>macsylib.hit.CoreHit method</i>), 118	
<code>__str__()</code> (<i>macsylib.hit.ModelHit method</i>), 119	
<code>__str__()</code> (<i>macsylib.model.Model method</i>), 107	

<code>__weakref__</code> (<i>macsylib.scripts.macsyprofile.HmmProfile</i> attribute), 160	<code>sylib.database.RepliconDB</code> method), 149
<code>__weakref__</code> (<i>macsylib.scripts.macsyprofile.LightHit</i> attribute), 162	<code>_fill_gene_bank()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 103
<code>__weakref__</code> (<i>macsylib.serialization.SystemSerializer</i> attribute), 144	<code>_fill_my_db()</code> (<i>macsylib.report.HMMReport</i> method), 139
<code>__weakref__</code> (<i>macsylib.serialization.TsvRejectedCandidates</i> attribute), 145	<code>_fill_my_db()</code> (<i>macsylib.scripts.macsyprofile.HmmProfile</i> method), 160
<code>__weakref__</code> (<i>macsylib.serialization.TsvSolutionSerializer</i> attribute), 144	<code>_fill_ordered_min_max()</code> (<i>macsylib.database.RepliconDB</i> method), 149
<code>__weakref__</code> (<i>macsylib.serialization.TsvSpecialHitSerializer</i> attribute), 146	<code>_fill_topology()</code> (<i>macsylib.database.RepliconDB</i> method), 149
<code>__weakref__</code> (<i>macsylib.solution.Solution</i> attribute), 142	<code>_find_all_installed_packages()</code> (in module <i>macsylib.scripts.macsydata</i>), 156
<code>__weakref__</code> (<i>macsylib.system.AbstractSetOfHits</i> attribute), 134	<code>_find_installed_package()</code> (in module <i>macsylib.scripts.macsydata</i>), 156
<code>__weakref__</code> (<i>macsylib.system.HitSystemTracker</i> attribute), 133	<code>_find_readme()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155
<code>__weakref__</code> (<i>macsylib.system.MatchMaker</i> attribute), 131	<code>_get_model_conf_node()</code> (<i>macsylib.model_conf_parser.ModelConfParser</i> method), 96
<code>_build_my_db()</code> (<i>macsylib.report.HMMReport</i> method), 139	<code>_get_model_node()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 103
<code>_build_my_db()</code> (<i>macsylib.scripts.macsyprofile.HmmProfile</i> method), 160	<code>_get_remote_available_versions()</code> (in module <i>macsylib.scripts.macsydata</i>), 156
<code>_build_my_indexes()</code> (<i>macsylib.database.Indexes</i> method), 147	<code>_get_replicon_name()</code> (<i>macsylib.report.GembaseHMMReport</i> method), 141
<code>_check_metadata()</code> (<i>macsylib.model_package.ModelPackage</i> method), 154	<code>_get_replicon_name()</code> (<i>macsylib.report.GeneralHMMReport</i> method), 140
<code>_check_model_conf()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155	<code>_get_replicon_name()</code> (<i>macsylib.report.HMMReport</i> method), 139
<code>_check_model_consistency()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155	<code>_get_replicon_name()</code> (<i>macsylib.report.OrderedHMMReport</i> method), 140
<code>_check_profiles()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155	<code>_hit_start()</code> (<i>macsylib.report.HMMReport</i> method), 139
<code>_check_replicon_consistency()</code> (<i>macsylib.cluster.Cluster</i> method), 125	<code>_hit_start()</code> (<i>macsylib.scripts.macsyprofile.HmmProfile</i> method), 161
<code>_check_structure()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155	<code>_index_dir()</code> (<i>macsylib.database.Indexes</i> method), 147
<code>_check_syntax()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 103	<code>_load_metadata()</code> (<i>macsylib.model_package.ModelPackage</i> method), 155
<code>_config_file_2_dict()</code> (<i>macsylib.config.Config</i> method), 91	<code>_parse_exchangeable()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 103
<code>_create_exchangeable_map()</code> (<i>macsylib.system.MatchMaker</i> method), 131	<code>_parse_genes()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 104
<code>_create_model()</code> (<i>macsylib.definition_parser.DefinitionParser</i> method), 103	<code>_parse_hmm_body()</code> (<i>macsylib.report.HMMReport</i> method), 139
<code>_fill_gembase_min_max()</code> (<i>mac-</i>	

`_parse_hmm_body()` (*macsylib.scripts.macsyprofile.HmmProfile* method), 161
`_parse_hmm_header()` (*macsylib.report.HMMReport* method), 140
`_parse_hmm_header()` (*macsylib.scripts.macsyprofile.HmmProfile* method), 161
`_parse_section()` (*macsylib.model_conf_parser.ModelConfParser* method), 96
`_profile_features()` (*macsylib.profile.Profile* method), 115
`_scan_definitions()` (*macsylib.registries.ModelLocation* method), 98
`_scan_profiles()` (*macsylib.registries.ModelLocation* method), 98
`_search_in_desc()` (in module *macsylib.scripts.macsydata*), 156
`_search_in_pack_name()` (in module *macsylib.scripts.macsydata*), 156
`_set_command_line_config()` (*macsylib.config.Config* method), 91
`_set_db_type()` (*macsylib.config.Config* method), 91
`_set_default_config()` (*macsylib.config.Config* method), 91
`_set_inter_gene_max_space()` (*macsylib.config.Config* method), 91
`_set_log_level()` (*macsylib.config.Config* method), 92
`_set_max_nb_genes()` (*macsylib.config.Config* method), 92
`_set_min_genes_required()` (*macsylib.config.Config* method), 92
`_set_min_mandatory_genes_required()` (*macsylib.config.Config* method), 92
`_set_model_config()` (*macsylib.config.Config* method), 92
`_set_models()` (*macsylib.config.Config* method), 92
`_set_models_dir()` (*macsylib.config.Config* method), 92
`_set_multi_loci()` (*macsylib.config.Config* method), 93
`_set_no_cut_ga()` (*macsylib.config.Config* method), 93
`_set_options()` (*macsylib.config.Config* method), 93
`_set_previous_run_config()` (*macsylib.config.Config* method), 93
`_set_project_config_file()` (*macsylib.config.Config* method), 93
`_set_replicon_topology()` (*macsylib.config.Config* method), 93
`_set_sequence_db()` (*macsylib.config.Config* method), 93
`_set_system_models_dir()` (*macsylib.config.Config* method), 93
`_set_system_wide_config()` (*macsylib.config.Config* method), 93
`_set_topology_file()` (*macsylib.config.Config* method), 93
`_set_user_config_file()` (*macsylib.config.Config* method), 94
`_set_user_wide_config()` (*macsylib.config.Config* method), 94
`_sorted_systems()` (*macsylib.solution.Solution* method), 142
`_str_2_tuple()` (*macsylib.config.Config* method), 94
`_url_json()` (*macsylib.model_package.RemoteModelIndex* method), 153

A

`AbstractClusterizedHits` (class in *macsylib.system*), 134
`AbstractCounterpartHit` (class in *macsylib.hit*), 120
`AbstractModelIndex` (class in *macsylib.model_package*), 153
`AbstractSetOfHits` (class in *macsylib.system*), 134
`AbstractUnordered` (class in *macsylib.system*), 137
`accessory_hits` (*macsylib.system.AbstractUnordered* property), 137
`add()` (*macsylib.registries.ModelRegistry* method), 97
`add_exchangeable()` (*macsylib.gene.Exchangeable* method), 114
`add_exchangeable()` (*macsylib.gene.ModelGene* method), 111
`add_model()` (*macsylib.model.ModelBank* method), 106
`add_new_gene()` (*macsylib.gene.GeneBank* method), 110
`add_subdefinition()` (*macsylib.registries.DefinitionLocation* method), 100
`all()` (*macsylib.registries.DefinitionLocation* method), 100
`allowed_hits` (*macsylib.system.AbstractUnordered* property), 137
`alternate_of()` (*macsylib.gene.Exchangeable* method), 114
`alternate_of()` (*macsylib.gene.ModelGene* method), 111
`average_wholeness` (*macsylib.solution.Solution* property), 142

B

`best_hit()` (*macsylib.report.HMMReport* method), 140
`build()` (*macsylib.database.Indexes* method), 148
`build_arg_parser()` (in module *macsylib.scripts.macsydata*), 157
`build_clusters()` (in module *macsylib.cluster*), 128

C

[check\(\)](#) (*macsylib.model_package.ModelPackage method*), 155
[check_consistency\(\)](#) (*macsylib.definition_parser.DefinitionParser method*), 104
[closest_hit\(\)](#) (*in module macsylib.cluster*), 128
[Cluster](#), 86
[Cluster](#) (*class in macsylib.cluster*), 124, 126
[clusterize_hits_around_key_genes\(\)](#) (*in module macsylib.cluster*), 129
[clusterize_hits_on_distance_only\(\)](#) (*in module macsylib.cluster*), 129
[cmd_name\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[codemeta.json](#), 87
[combine_clusters\(\)](#) (*in module macsylib.solution*), 143
[combine_multisystems\(\)](#) (*in module macsylib.solution*), 143
[compute_best_MSHit\(\)](#) (*in module macsylib.hit*), 123
[Config](#) (*class in macsylib.config*), 91
[CONTRIBUTING](#), 87
[CONTRIBUTORS](#), 87
[COPYING](#), 87
[COPYRIGHT](#), 87
[core_gene](#) (*macsylib.gene.ModelGene property*), 111
[CoreGene](#) (*class in macsylib.gene*), 110
[CoreHit](#) (*class in macsylib.hit*), 116
[count\(\)](#) (*macsylib.system.AbstractSetOfHits method*), 134
[counterpart](#) (*macsylib.hit.AbstractCounterpartHit property*), 120

D

[DefinitionLocation](#) (*class in macsylib.registries*), 100
[DefinitionParser](#) (*class in macsylib.definition_parser*), 102
[do_available\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_check\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_cite\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_download\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_freeze\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_help\(\)](#) (*in module macsylib.scripts.macsydata*), 157
[do_info\(\)](#) (*in module macsylib.scripts.macsydata*), 158
[do_init_package\(\)](#) (*in module macsylib.scripts.macsydata*), 158
[do_install\(\)](#) (*in module macsylib.scripts.macsydata*), 158
[do_list\(\)](#) (*in module macsylib.scripts.macsydata*), 158

[do_search\(\)](#) (*in module macsylib.scripts.macsydata*), 158
[do_show_definition\(\)](#) (*in module macsylib.scripts.macsydata*), 159
[do_show_package\(\)](#) (*in module macsylib.scripts.macsydata*), 159
[do_uninstall\(\)](#) (*in module macsylib.scripts.macsydata*), 159
[doc](#), 86
[download\(\)](#) (*macsylib.model_package.RemoteModelIndex method*), 153

E

[EmptyFileError](#), 151
[Exchangeable](#) (*class in macsylib.gene*), 113
[exchangeables](#) (*macsylib.gene.ModelGene property*), 111
[execute\(\)](#) (*macsylib.profile.Profile method*), 115
[extract\(\)](#) (*macsylib.report.HMMReport method*), 140

F

[family_name](#) (*macsylib.model.Model property*), 107
[family_name](#) (*macsylib.registries.DefinitionLocation property*), 100
[fasta_iter\(\)](#) (*in module macsylib.database*), 151
[filter\(\)](#) (*macsylib.model.Model method*), 107
[find_best_solutions\(\)](#) (*in module macsylib.solution*), 143
[find_my_indexes\(\)](#) (*macsylib.database.Indexes method*), 148
[forbidden_hits](#) (*macsylib.system.AbstractUnordered property*), 137
[fulfilled_function\(\)](#) (*macsylib.cluster.Cluster method*), 125, 126
[fulfilled_function\(\)](#) (*macsylib.system.AbstractClusterizedHits method*), 134
[functions](#) (*macsylib.cluster.Cluster property*), 125, 126

G

[GembaseHMMReport](#) (*class in macsylib.report*), 141
[GeneBank](#) (*class in macsylib.gene*), 109
[GeneralHMMReport](#) (*class in macsylib.report*), 140
[genes](#) (*macsylib.database.RepliconInfo attribute*), 148
[genes\(\)](#) (*macsylib.model.Model method*), 107
[genes_fqn\(\)](#) (*macsylib.gene.GeneBank method*), 110
[GeneStatus](#) (*class in macsylib.gene*), 114
[get\(\)](#) (*macsylib.database.RepliconDB method*), 149
[get_all_definitions\(\)](#) (*macsylib.registries.ModelLocation method*), 99
[get_best_hit_4_func\(\)](#) (*in module macsylib.hit*), 123
[get_best_hits\(\)](#) (*in module macsylib.hit*), 124
[get_def_to_detect\(\)](#) (*in module macsylib.utils*), 152

[get_definition\(\)](#) (*macsylib.registries.ModelLocation* method), 99
[get_definitions\(\)](#) (*macsylib.registries.ModelLocation* method), 99
[get_gene\(\)](#) (*macsylib.model.Model* method), 107
[get_gene_name\(\)](#) (in module *macsylib.scripts.macsypipeline*), 162
[get_hits_encoding_multisystem\(\)](#) (*macsylib.system.System* method), 135
[get_loners\(\)](#) (*macsylib.system.System* method), 135
[get_metadata\(\)](#) (*macsylib.model_package.RemoteModelIndex* method), 154
[get_multisystems\(\)](#) (*macsylib.system.System* method), 135
[get_profile\(\)](#) (*macsylib.profile.ProfileFactory* method), 115
[get_profile\(\)](#) (*macsylib.registries.ModelLocation* method), 99
[get_profile_len\(\)](#) (in module *macsylib.scripts.macsypipeline*), 162
[get_profiles_names\(\)](#) (*macsylib.registries.ModelLocation* method), 99
[get_replicon_names\(\)](#) (in module *macsylib.utils*), 152
[get_version_message\(\)](#) (in module *macsylib.scripts.macsypipeline*), 162
[getter_maker\(\)](#) (*macsylib.system.MetaSetOfHits* method), 133
[guess_if_really_gembase\(\)](#) (*macsylib.database.RepliconDB* method), 150

H

[help\(\)](#) (*macsylib.model_package.ModelPackage* method), 155
[hit](#) (*macsylib.hit.ModelHit* property), 119
[hit_weights](#) (*macsylib.cluster.Cluster* property), 125, 127
[hit_weights\(\)](#) (*macsylib.config.Config* method), 94
[hits](#) (*macsylib.cluster.Cluster* property), 125, 127
[hits](#) (*macsylib.system.AbstractUnordered* property), 137
[hits](#) (*macsylib.system.RejectedCandidate* property), 136
[hits](#) (*macsylib.system.System* property), 135
[hits_number](#) (*macsylib.solution.Solution* property), 142
[hits_positions](#) (*macsylib.solution.Solution* property), 143
[HitSystemTracker](#) (class in *macsylib.system*), 133
[HitWeight](#) (class in *macsylib.hit*), 122
[hmmer_dir\(\)](#) (*macsylib.config.Config* method), 94
[Hmmpipeline](#) (class in *macsylib.scripts.macsypipeline*), 160
[HMMReport](#) (class in *macsylib.report*), 138

I

[Indexes](#) (class in *macsylib.database*), 147
[info\(\)](#) (*macsylib.model_package.ModelPackage* method), 155
[init_logger\(\)](#) (in module *macsylib.scripts.macsydata*), 159
[init_logger\(\)](#) (in module *macsylib.scripts.macsypipeline*), 162
[inter_gene_max_space](#) (*macsylib.gene.ModelGene* property), 112
[inter_gene_max_space](#) (*macsylib.model.Model* property), 108
[inter_gene_max_space\(\)](#) (*macsylib.config.Config* method), 94
[is_a\(\)](#) (in module *macsylib.cluster*), 129
[is_accessory\(\)](#) (*macsylib.gene.ModelGene* method), 112
[is_compatible\(\)](#) (*macsylib.system.System* method), 135
[is_exchangeable](#) (*macsylib.gene.Exchangeable* property), 114
[is_exchangeable](#) (*macsylib.gene.ModelGene* property), 112
[is_forbidden\(\)](#) (*macsylib.gene.ModelGene* method), 112
[is_mandatory\(\)](#) (*macsylib.gene.ModelGene* method), 112
[items\(\)](#) (*macsylib.database.RepliconDB* method), 150
[iteritems\(\)](#) (*macsylib.database.RepliconDB* method), 150

J

[join_def_path\(\)](#) (in module *macsylib.registries*), 101

L

[LightHit](#) (class in *macsylib.scripts.macsypipeline*), 161
[LikelySystem](#) (class in *macsylib.system*), 137
[list_package_vers\(\)](#) (*macsylib.model_package.RemoteModelIndex* method), 154
[list_packages\(\)](#) (*macsylib.model_package.RemoteModelIndex* method), 154
[LocalModelIndex](#) (class in *macsylib.model_package*), 153
[loci_nb](#) (*macsylib.system.System* property), 135
[loci_num](#) (*macsylib.system.System* property), 135
[log_level\(\)](#) (*macsylib.config.Config* method), 94
[Loner](#) (class in *macsylib.hit*), 121
[loner](#) (*macsylib.cluster.Cluster* property), 125, 127
[loner](#) (*macsylib.gene.ModelGene* property), 113
[loner](#) (*macsylib.hit.AbstractCounterpartHit* property), 120
[loner](#) (*macsylib.hit.Loner* property), 121

loner (*macsylib.hit.ModelHit* property), 119
 LonerMultiSystem (class in *macsylib.hit*), 122

M

MacsydataError, 151
 MacsyDataLimitError, 151
 MacsyDefaults (class in *macsylib.config*), 90
 macsylib, 86
 macsylib.cluster
 module, 126
 macsylib.database
 module, 148
 macsylib.definition_parser
 module, 102
 macsylib.error
 module, 151
 macsylib.model_conf_parser
 module, 96
 macsylib.scripts.macsydata
 module, 156
 macsylib.scripts.macsyprofile
 module, 160
 macsylib.search_genes
 module, 141
 MacsylibError, 151
 main() (in module *macsylib.scripts.macsydata*), 159
 main() (in module *macsylib.scripts.macsyprofile*), 162
 mandatory_hits (*macsylib.system.AbstractUnordered*
 property), 137
 match() (*macsylib.system.OrderedMatchMaker*
 method), 132
 match() (*macsylib.system.UnorderedMatchMaker*
 method), 132
 MatchMaker (class in *macsylib.system*), 131
 max (*macsylib.database.RepliconInfo* attribute), 148
 max_nb_genes (*macsylib.model.Model* property), 108
 max_nb_genes() (*macsylib.config.Config* method), 94
 merge() (*macsylib.cluster.Cluster* method), 125, 127
 metadata (*macsylib.model_package.ModelPackage*
 property), 155
 MetaDefLoc (class in *macsylib.registries*), 99
 MetaSetOfHits (class in *macsylib.system*), 133
 min (*macsylib.database.RepliconInfo* attribute), 148
 min_genes_required (*macsylib.model.Model* prop-
 erty), 108
 min_genes_required() (*macsylib.config.Config*
 method), 95
 min_mandatory_genes_required (mac-
 sylib.model.Model property), 108
 min_mandatory_genes_required() (mac-
 sylib.config.Config method), 95
 Model, 86
 Model (class in *macsylib.model*), 106
 model (*macsylib.gene.ModelGene* property), 113

Model family, 86
 model_family_name (*macsylib.gene.CoreGene* prop-
 erty), 110
 ModelBank (class in *macsylib.model*), 105
 ModelConfParser (class in *mac-
 sylib.model_conf_parser*), 96
 ModelDefinition, 86
 ModelGene (class in *macsylib.gene*), 111
 ModelHit (class in *macsylib.hit*), 119
 ModelInconsistencyError, 151
 ModelLocation (class in *macsylib.registries*), 98
 ModelPackage (class in *macsylib.model_package*), 154
 ModelRegistry (class in *macsylib.registries*), 97
 models() (*macsylib.registries.ModelRegistry* method),
 97
 models_dir() (*macsylib.config.Config* method), 95
 module
 macsylib.cluster, 126
 macsylib.database, 148
 macsylib.definition_parser, 102
 macsylib.error, 151
 macsylib.model_conf_parser, 96
 macsylib.scripts.macsydata, 156
 macsylib.scripts.macsyprofile, 160
 macsylib.search_genes, 141
 multi_loci (*macsylib.model.Model* property), 108
 multi_loci (*macsylib.system.System* property), 136
 multi_loci() (*macsylib.config.Config* method), 95
 multi_model (*macsylib.gene.ModelGene* property), 113
 multi_model (*macsylib.hit.ModelHit* property), 119
 multi_system (*macsylib.cluster.Cluster* property), 126,
 127
 multi_system (*macsylib.gene.ModelGene* property),
 113
 multi_system (*macsylib.hit.AbstractCounterpartHit*
 property), 120
 multi_system (*macsylib.hit.ModelHit* property), 120
 multi_system (*macsylib.hit.MultiSystem* property), 121
 MultiSystem (class in *macsylib.hit*), 121

N

name (*macsylib.gene.CoreGene* property), 110
 name (*macsylib.model.Model* property), 108
 neutral_hits (*macsylib.system.AbstractUnordered*
 property), 137
 NoneConfig (class in *macsylib.config*), 96

O

occurrence() (*macsylib.system.System* method), 136
 OptionError, 151
 OrderedHMMReport (class in *macsylib.report*), 140
 OrderedMatchMaker (class in *macsylib.system*), 132
 out_dir() (*macsylib.config.Config* method), 95

P

parse() (*macsylib.definition_parser.DefinitionParser* method), 105
parse() (*macsylib.model_conf_parser.ModelConfParser* method), 96
parse() (*macsylib.scripts.macsyprofile.HmmProfile* method), 161
parse_args() (in module *macsylib.scripts.macsyprofile*), 163
parse_filtering() (*macsylib.model_conf_parser.ModelConfParser* method), 96
parse_time() (in module *macsylib.utils*), 152
parse_weights() (*macsylib.model_conf_parser.ModelConfParser* method), 97
position (*macsylib.system.AbstractSetOfHits* property), 134
present_genes() (*macsylib.system.MatchMaker* method), 132
Profile (class in *macsylib.profile*), 115
profile (*macsylib.gene.CoreGene* property), 111
ProfileFactory (class in *macsylib.profile*), 114
pyproject.toml, 87

R

README.md, 87
reasons (*macsylib.system.RejectedCandidate* property), 136
reasons (*macsylib.system.UnlikelySystem* property), 138
RejectedCandidate (class in *macsylib.system*), 136
remote_exists() (*macsylib.model_package.RemoteModelIndex* method), 154
RemoteModelIndex (class in *macsylib.model_package*), 153
replace() (*macsylib.cluster.Cluster* method), 126, 127
replicon_infos() (*macsylib.database.RepliconDB* method), 150
replicon_name (*macsylib.cluster.Cluster* property), 126, 128
replicon_name (*macsylib.system.AbstractSetOfHits* property), 134
replicon_names() (*macsylib.database.RepliconDB* method), 150
RepliconDB (class in *macsylib.database*), 149
RepliconInfo (class in *macsylib.database*), 148
result_header() (in module *macsylib.scripts.macsyprofile*), 163
root_name() (*macsylib.registries.DefinitionLocation* class method), 100

S

save() (*macsylib.config.Config* method), 95

save_extract() (*macsylib.report.HMMReport* method), 140
scaffold_to_cluster() (in module *macsylib.cluster*), 129
scan_models_dir() (in module *macsylib.registries*), 101
score (*macsylib.cluster.Cluster* property), 126, 128
score (*macsylib.solution.Solution* property), 143
score (*macsylib.system.System* property), 136
search_genes() (in module *macsylib.search_genes*), 141
serialize() (*macsylib.serialization.TsvLikelySystemSerializer* method), 145
serialize() (*macsylib.serialization.TsvRejectedCandidatesSerializer* method), 145
serialize() (*macsylib.serialization.TsvSolutionSerializer* method), 144
serialize() (*macsylib.serialization.TsvSpecialHitSerializer* method), 146
serialize() (*macsylib.serialization.TsvSystemSerializer* method), 144
serialize() (*macsylib.serialization.TxtLikelySystemSerializer* method), 146
serialize() (*macsylib.serialization.TxtSystemSerializer* method), 146
serialize() (*macsylib.serialization.TxtUnlikelySystemSerializer* method), 146
set_status() (*macsylib.gene.ModelGene* method), 113
setup.py, 87
Solution, 86
Solution (class in *macsylib.solution*), 142
sort_hits_by_status() (*macsylib.system.MatchMaker* method), 132
sort_model_hits() (in module *macsylib.hit*), 123
split_cluster_on_key_genes() (in module *macsylib.cluster*), 130
split_def_name() (in module *macsylib.registries*), 101
split_fqn() (*macsylib.registries.DefinitionLocation* class method), 100
status (*macsylib.gene.Exchangeable* property), 114
status (*macsylib.gene.ModelGene* property), 113
System, 86
System (class in *macsylib.system*), 135
SystemDetectionError, 151
systems (*macsylib.solution.Solution* property), 143
SystemSerializer (class in *macsylib.serialization*), 144

T

tests, 86
threads_available() (in module *macsylib.utils*), 152
Timeout, 151
topology (*macsylib.database.RepliconInfo* attribute), 148

TsvLikelySystemSerializer (class in *macsylib.serialization*), 145
 TsvRejectedCandidatesSerializer (class in *macsylib.serialization*), 145
 TsvSolutionSerializer (class in *macsylib.serialization*), 144
 TsvSpecialHitSerializer (class in *macsylib.serialization*), 146
 TsvSystemSerializer (class in *macsylib.serialization*), 144
 TxtLikelySystemSerializer (class in *macsylib.serialization*), 146
 TxtSystemSerializer (class in *macsylib.serialization*), 146
 TxtUnikelySystemSerializer (class in *macsylib.serialization*), 146

U

unarchive_package() (*macsylib.model_package.AbstractModelIndex* method), 153
 UnlikelySystem (class in *macsylib.system*), 138
 UnorderedMatchMaker (class in *macsylib.system*), 132

V

verbosity_to_log_level() (in module *macsylib.scripts.macsydata*), 160
 verbosity_to_log_level() (in module *macsylib.scripts.macsyprofile*), 163
 version (*macsylib.registries.ModelLocation* property), 99

W

wholeness (*macsylib.system.AbstractSetOfHits* property), 134
 worker_cpu() (in module *macsylib.search_genes*), 141
 working_dir() (*macsylib.config.Config* method), 95