
MacSyFinder

Release 2.0

Sophie Abby, Bertrand Néron

Oct 15, 2021

CONTENTS

1	User Guide	3
1.1	User Guide	3
2	Modeller Guide	47
2.1	Modeller Guide	47
3	Developer Guide	69
3.1	Developer Guide	69
4	Indices and tables	117
	Python Module Index	119
	Index	121

Note: A new version of MacSyFinder (v2) is available, see [here for an overview of the novelties](#). The search engine was improved, and some bugs/unwanted behaviors corrected. MacSyFinder's models for v2 are very similar, yet not compatible with those from v1. See here for details on [how to carry your models to v2](#).

MacSyFinder is a program to **model and detect macromolecular systems, genetic pathways**. . . in protein datasets. In prokaryotes, these systems have often evolutionarily conserved properties: they are made of **conserved components**, and are encoded in **compact loci** (conserved genetic architecture). The user models these systems with MacSyFinder to reflect these conserved features, and to allow their efficient detection.

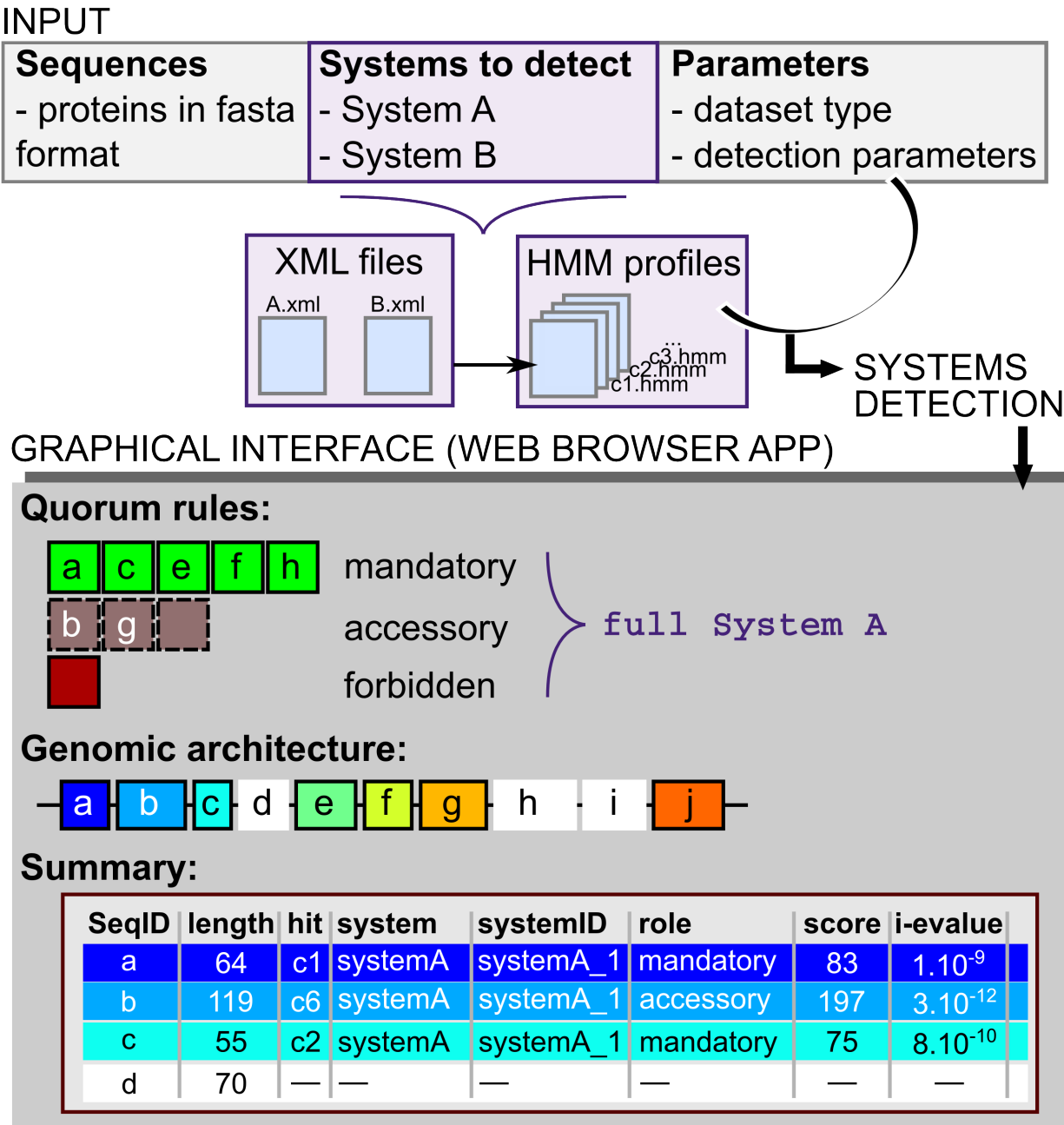
Criteria for systems detection include **component content (quorum)**, and **genomic co-localization**. Each component corresponds to a hidden Markov model (HMM) protein profile to perform sequence similarity searches with the program Hmmer.

In order to model macromolecular systems, the user:

- builds or gather from databanks **HMM protein profiles** for components of interest,
- defines **decision rules** for each system in a dedicated XML grammar (see [Macromolecular models](#)).

Note: If you use MacSyFinder, please cite:

Abby SS, Néron B, Ménager H, Touchon M, Rocha EPC (2014). MacSyFinder: A Program to Mine Genomes for Molecular Systems with an Application to CRISPR-Cas Systems. PLoS ONE 9(10): e110726. doi:10.1371/journal.pone.0110726



USER GUIDE

1.1 User Guide

1.1.1 Running MacSyFinder

What's new in MacSyFinder v2?

For Version 2, MacSyFinder was carried under [Python 3](#)

New features and search engine

The **search engine** was changed for a more intuitive and comprehensive exploration of putative systems.

The search is now more thorough and avoid undesirable side-effects of the previous search engine. Being more thorough, it now also includes a **scoring scheme** to build candidate systems from sets of detected components (clusters), and can offer several optimal “solutions” (sets of detected systems) based on a combinatorial exploration of detected clusters. See [here for more details](#).

Several **new features** were added, including:

- a **new type of gene component** “neutral” was added in order to provide more possibilities for systems’ modelling in macsy-models. See [here](#) for more details.
- more flexibility was introduced in the **search for systems’ components using HMMER**. It is now possible to use the *cut_ga* threshold when provided in the HMM profiles used for components’ similarity search. This enables to have a search tailored for each HMM profile, and thus component. See [here](#) for more details.
- a **new file structure** was created to better organize MacSyFinder’s packages (i.e. that include systems’ models and corresponding HMMER profiles). See [here](#) for details.
- a **tool** to easily install and distribute MacSyFinder’s packages was created. See [here](#) for more details on *macsy-data*.
- the **format for MacSyFinder’s models** has slightly changed, in order to offer more possibilities, and more readability. To see **how to carry models from v1 to v2**, visit [here](#).

Also, the search modes corresponding to “unordered” and “unordered_replicon” were merged into the “**unordered**” search mode - as they basically correspond to the same behaviour.

Note: In v2, output files were also re-defined. See [here for more details](#).

Dependencies

MacSyFinder v2 no longer requires the *formatdb* or *makeblastdb* tools from NCBI. However, new dependencies are used, but as they are Python libraries, it should be transparent for the user, and not require manual installations. See [here for details](#).

Models are more formalized

The models data are more formalized, with a well defined structure. For instance the definitions and profiles must be packed together in what we call a *macsy-model* package. If you intend to model new systems please refer to the *Modeller Guide*.

Models installation

We now provide a new tool to manage the models. See *Models installation with macsydata*.

Models configuration

The modeler can provide some specific configuration values released along the model package. See *Model configuration*.

Modeller helper tool

To help modellers create new models we provide a new helper tool *macsyprofile*, which analyses HMMER raw output files from results of a previous MacSyFinder run, to provide information on all hits even if filtered out. See *macsyprofile*.

Installation

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

MacSyFinder dependencies

Python version ≥ 3.7 is required to run MacSyFinder: <https://docs.python.org/3.7/index.html>

MacSyFinder 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 MacSyFinder. Otherwise, the paths to this executable must be specified in the command-line: see the *command-line options*.

MacSyFinder also relies on five Python library dependencies:

- colorlog
- pyyaml
- packaging
- networkx

- pandas

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

MacSyFinder Installation procedure

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

Archive overview

- **doc** => the documentation in html and pdf
- **etc** => a template of macsyfinder configuration file
- **test** => all what is needed for unitary tests
- **macsypy** => the macsyfinder python library
- **setup.py** => the installation script
- **requirements.txt** => the python dependencies
- **requirements_dev.txt** => the python dependencies for developers
- **COPYING** => the licensing
- **COPYRIGHT** => the copyright
- **README.md** => very brief macsyfinder overview
- **CONTRIBUTORS** => list of people who contributed to the code

Installation steps:

Make sure every required dependency/software is present.

By default MacSyFinder 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.

```
pip install --no-binary macsyfinder macsyfinder
```

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

```
pip install --user --no-binary macsyfinder macsyfinder
```

installation in a virtualenv

```
python3.7 -m venv macsyfinder
cd macsyfinder
source bin/activate
pip install --no-binary macsyfinder macsyfinder
```

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

```
source macsyfinder/bin/activate
```

Then run *macsyfinder* or *macsydata*.

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 MacSyFinder in the Python libraries of your system, you can install MacSyFinder in a virtual environment (<http://www.virtualenv.org/>).

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

Uninstalling MacSyFinder

To uninstall MacSyFinder (the last version installed), run:

```
(sudo) pip uninstall macsyfinder
```

If you install it in a virtualenv, just delete the virtual environment. For instance if you create a virtualenv name *macsyfinder*:

```
python3.7 -m venv macsyfinder
```

To delete it, remove the directory:

```
rm -R macsyfinder
```

Models installation with *macsydata*

Once MacSyFinder is installed you have access to an utility program to manage the models: *macsydata*

This script allows to search, download, install and get information from MacSyFinder models stored on github or locally installed. The general syntax for *macsydata* is:

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

To list all models available:

```
macsydata available
```

To search for models:

```
macsydata search TXSS
```

you can also search in models description:

```
macsydata search -S secretion
```

To install a model package:

```
macsydata install <model name>
```

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

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

To know how to cite a model package:

```
macsydata cite <model name>
```

To list all *macsydata* subcommands:

```
macsydata --help
```

To list all available options for a subcommand:

```
macsydata <subcommand> --help
```

MacSyFinder Quick Start

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

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

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 fasta format (for further details see [Input dataset](#)).

3. You need to have models to search in your input data. Please refer to [Macromolecular models](#) to create your own package of models. We will soon provide a set of predefined models for you to test.

4. Command lines:

- Type: `macsyfinder -h`

To see all options available. All command-line options are described in the [Command-line options section](#). In order to run MacSyFinder on your favorite dataset as soon as you have [installed it](#), you can simply follow the following steps:

- On a “metagenomic” (unordered) dataset for example: `macsyfinder --db-type unordered --sequence-db metagenome.fasta --models model_family all`

will detect all models of `model_family` modelled in `.xml` files placed in the “*my-models*” folder without taking into account any gene order.

- On a completely assembled genome (where the gene order is known, and is relevant for systems’ detection):

```
macsyfinder --db-type ordered_replicon --sequence-db mygenome.fasta
--models model_family ModelA ModelB
```

will detect the macromolecular systems described in the two models “*ModelA*” and “*ModelB*” in a complete genome from the “*ModelA.xml*” and “*ModelB.xml*” definition files placed in the folder “*my-models/model_family/definitions*”.

- If you want to run the same analysis as above but with models not installed by `macsydata`:

```
macsyfinder --db-type ordered_replicon --sequence-db mygenome.fasta
--models-dir my-models --models model_family ModelA ModelB
```

my-models is the directory containing the model packages. The models must follow the [macsy-models package](#) structure.

Note: Systems names have to be spelled in a case-sensitive way to run their detection from the command-line. The name of the System corresponds to the suffix defined for xml files (`.xml` by default), for example “*toto*” for a model defined in “*toto.xml*”.

The “*all*” keyword allows to detect all models available in the definitions folder in a single run. See the [Command-line options](#).

Input and Options of MacSyFinder

Input dataset

The input dataset must be a set of protein sequences in **Fasta format** (see http://en.wikipedia.org/wiki/FASTA_format).

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` parameters respectively, described in the [Command-line options](#) (see [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, MacSyFinder 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 *Command-line options* (see *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*).

Command-line options

Optional arguments:

```
-h, --help          Show the help message and exit

-m [MODELS [MODELS ...]], --models [MODELS [MODELS ...]]
    The models to search. The --models option can be set several
    ↪times.'
    For each --models options 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.'
    '--models TXSS Flagellum T2SS'
    means MSF will search for models TXSS/Flagellum and
    ↪TXSS/T2SS
    '--models TXSS all'
    means for all models found in model package TXSS
    '--models CRISPRcas/subtyping all'
    means MSF will search for all models described in the
    ↪CRISPRCas/subtyping subfamily.
    (required unless --previous-run is set)
```

Input dataset options:

```
--sequence-db SEQUENCE_DB
    Path to the sequence dataset in fasta format.
    (required unless --previous-run is set)

--db-type {ordered_replicon,gembase,unordered}
    The type of dataset to deal with. "unordered" corresponds
    to a non-assembled genome,
    "ordered_replicon" to an assembled genome,
    and "gembase" to a set of replicons where sequence identifiers
    follow this convention: ">RepliconName SequenceID".
    (required unless --previous-run is set)

--replicon-topology {linear,circular}
    The topology of the replicons
    (this option is meaningful only if the db_type is
    'ordered_replicon' or 'gembase').

--topology-file TOPOLOGY_FILE
    Topology file path. The topology file allows to specify a
    ↪topology
```

(continues on next page)

(continued from previous page)

```

    (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
    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 INTER_GENE_MAX_SPACE 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.
    The first value must match to a model, the second to a number_
↳of components.
    This option can be repeated several times:
    "--inter-gene-max-space TXSS/T2SS 12 --inter-gene-max-space_
↳TXSS/Flagellum 20
--min-mandatory-genes-required MIN_MANDATORY_GENES_REQUIRED MIN_MANDATORY_GENES_
↳REQUIRED
    The minimal number of mandatory genes required for model_
↳assessment.
    The first value must correspond to a model fully qualified name,
↳ the second value to an integer.
    This option can be repeated several times:
    "--min-mandatory-genes-required TXSS/T2SS 15 --min-
↳mandatory-genes-required TXSS/Flagellum 10"
--min-genes-required MIN_GENES_REQUIRED 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.
    This option can be repeated several times:
    "--min-genes-required TXSS/T2SS 15 --min-genes-required_
↳TXSS/Flagellum 10
--max-nb-genes MAX_NB_GENES 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.
    This option can be repeated several times:
    "--max-nb-genes TXSS/T2SS 5 --max-nb-genes TXSS/Flagellum 10
↳"
--multi-loci 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
    "--multi-loci model_familyA/model_1,model_familyB/model_2"

```

Options for Hmmer execution and hits filtering:

```

--hmmmer HMMER          Path to the hmmsearch program.
                        If it is not specify rely on the PATH
                        (default: hmmsearch)
--e-value-search 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 Mf try to applied a threshold per profile by
↳using the
                        hmmmer -cut-ga option. This is possible only if the Ga bit
↳score is present in the profile otherwise MF switch to use the
                        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)
--i-evalue-sel I_EVALUE_SEL
                        Maximal independent e-value for Hmmer hits to be selected for
↳system detection.
                        (default:0.001)
--coverage-profile COVERAGE_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 MANDATORY_WEIGHT
                        the weight (score) of a mandatory component when scoring
↳clusters
                        (default:1.0)
--accessory-weight ACCESSORY_WEIGHT
                        the weight (score) of an accessory component when scoring
↳clusters
                        (default:0.5)
--exchangeable-weight EXCHANGEABLE_WEIGHT
                        the weight modifier for the score of a component that is
↳exchangeable
                        (default:0.8)
--redundancy-penalty REDUNDANCY_PENALTY
                        the weight modifier for the score of a component that is
↳already present in another cluster
                        (default:1.5)
--loner-multi-system-weight 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 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.
-o OUT_DIR, --out-dir OUT_DIR
    Path to the directory where to store results.
    if out-dir is specified res-search-dir will be ignored.
--index-dir 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 RES_SEARCH_SUFFIX
    The suffix to give to Hmmer raw output files. (default: .search_
    ↪hmm.out)
--res-extract-suffix RES_EXTRACT_SUFFIX
    The suffix to give to filtered hits output files. (default: .
    ↪res_hmm_extract)
--profile-suffix 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:

```

-w WORKER, --worker WORKER
    Number of workers to be used by MacSyFinder.
    In the case the user wants to run MacSyFinder in a multi-thread
    ↪mode.
        (0 mean all cores will be used).
        (default: 1)
-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 macsyfinder.log)
    (default: False)
--version
    show program's version number and exit
-l, --list-models
    display the all models installed in generic location and quit.
--cfg-file CFG_FILE
    Path to a MacSyFinder configuration file to be used.
--previous-run PREVIOUS_RUN
    Path to a previous MacSyFinder 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.

```

(continues on next page)

(continued from previous page)

```

        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, --hmmer

```

Note: For some command line examples, have a look [here](#), or at the [MacSyFinder Quick Start](#) section.

Configuration file

Options to run MacSyFinder can be specified in a configuration file. The [Config object](#) handles all configuration options for MacSyFinder. Three locations are parsed to find configuration files:

- \$PREFIX/etc/macsyfinder/macsyfinder.conf
- \$(HOME)/.macsyfinder/macsyfinder.conf
- ./macsyfinder.conf

Moreover these three locations options can be passed on the command-line.

Each file can define options, and in the end all options are integrated. If an option is specified several times:

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

\$PREFIX/etc/macsyfinder/macsyfinder.conf < \$(HOME)/.macsyfinder/macsyfinder.conf < macsyfinder.conf < “command-line” options

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 MacSyFinder, 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*)
- *loner_multi_system_weight* (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 = *macsyfinder.log* in directory of the run)

Example of a configuration file

```

[base]
prefix = /path/to/macsyfinder/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/macsyfinder/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, ~/.macsyfinder/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 (“macsyfinder.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)/.macsyfinder/macsyfinder.conf < macsyfinder.conf < "command-line" options`` *models_dir* is a single path to a directory where macsyfinder can find models.

But the *system_models_dir* can be set in general configuration file `$PREFIX/etc/macsyfinder/macsyfinder.conf`. *system_models_dir* manage a list of locations where macsyfinder can find models. The order of locations is important, it reflects the precedence rule (The models found in last location supersede models found in previous location).

By default `system_models_dir` is set to `$PREFIX/share/macsyfinder/data/models`, `$HOME/.macsyfinder/data`

In-house input files

Gembase format

In order to allow the users to run MacSyFinder 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, MacSyFinder will be able to treat each replicon separately to assess macromolecular systems’ presence.

For instance:

```
>PlasmidA_0001 YP_003225072.1 | putative stcE protein
MKLKYLSCMILASLAMGAFATAADNNSAIYFNTTQPVNDLQGGLAEEVK
FAQSQILSAHPKEGESQQHLTSLRKSLLLRLVKADDKTPVQVEARDAND
KILGTLTLSPSSSLPDTVYHLDGVPADGIDFTPQNGTKKIINTVAEVNKL
SDASGSSIKSYLANALVEIQTANGRWIRDMYLPQGAELEGKMVRFVSYA
GYNSTVFYGDRKVTLVSGNTLLFKYVNGQWFRSGELENNRIAYAQHTWSA
ELPAHWIVPGLNLVIKQGNLSGSLNDINVGAPGELLHTIDIGMLTTPRG
RFDFAKDKEAHREYFQTIPVSRMIVNNYAPLHLKEVMLPTGTLLTDADPG
>PlasmidA_0002 YP_003225073.1 | type II secretion protein EtpC
MLFFLSSRRDRNLFIKDIALKMLTPNWVLCVILLIAGYQLVSVIRHFWLT
PATASADLSHVSSETAVTDEHTEENFVFTLFGTASPPLSEGKVQKTTSS
LSDDLSSGGDLVGRGILYSSVTEHSVAIFAHNNRQFSLGIGEKVPGYDAT
ISAIFSDHIVINYQGNASLPLRYDNPAKRNAQDDNNLIVGPVTTQANFR
VKNIFDIMSLSPTVNNTLSGYRLSPGKASSLFYNAGLHDNDLAVLLNGS
ELRDTRQAKQIMKQLTELKEIKITVERDGQLYDAFIAVGEN
....
>ChromosomeA_0001 YP_003573410.1 | adhesin-like protein
MKKLFLFAALLMTGFAYFSCEDVVDNPAQDPAQSWNYSVSVKFADFDFNG
AVDENSVPYTYKAPTTLTYVLNEENTLMGTITTTDAAPAIGDYGTIAGTLTG
SIGNNLIITTKIGNDLTKQDGTLSKSAIENGIVQTAEVPIKIYNANSGLT
TASAKMDNTAAIAYTSLGYIKGGDKILFVEGNQTFEWTVNEEFDPTSTD
LYIALPMNTDPETEYTISSDSKDGTRGGTFKLADYPTLAAGKVSNIYIGG
IPFIQTGVDLTWKDAYMRTPNNTWYMNNINNGWPATFSQEVEDGKSFIV
TQSGPTLDSLNVVVGVTGKEVNTLNNIRLGKDRSINIGDKHGWEYDYG
THDIYGWGAKANVTLIGENECETLYIQCPATKKGEGTLNKNLSIDSYGS
>ChromosomeA_0020 YP_003573411.1 | hypothetical protein
MKRIVLITLVSILTTFQAIAQVANGFYRVQNNASSRYITLRDNAVGTVDY
SSTNVDSLNIWTSGFDKVKSNPASIIYVEQHDSKYDLKVQGTGIYAITG
GRTYLELRPKDSGYILAVTYNGMEGRLYDSEEDVDGEGYVKRSGNSAYQY
WSFIPVDTENNYIGLQPTVQVGDNYGTLYASYPFKAASSGIKFYYVDAI
....
>NC_001548_0015 YP_003225080.1 | type II secretion protein EtpJ (translation)
MSQQRVKGFTLLEMLLALAVFAALSISAFQVLQSGIRAHLSQDKVRRLA
ELQRGGSQIERDLMQMPRHSRGSEGLLLAAPHLLKSDDWGISFTRNSWL
NPAGMLPRPELQWVGYYRLRQQLERLSYFYVDHPSGIAPDVRVLEGVHA
FRLRFFVNGTWQARWDSTSILPQAVEVTLMDDFAELTRLFLVSKETA
```

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). Mac-

SyFinder search results will thus be reported for each of these three replicons.

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

MacSyFinder provides different types of output files. At each run, MacSyFinder creates a new folder, whose name is based on a fixed prefix and a random suffix, for instance “macsyfinder-20130128_08-57-46”. MacSyFinder output files are stored in this run-specific folder.

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 are provided when MacSyFinder 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 are provided, 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 MacSyFinder** 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 *macsyfinder.report*.

- *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_clusters.txt* - This file lists candidate clusters of systems' components that were rejected by MacSyFinder during the search process, and were thus not assigned to a candidate system.
- *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 MacSyFinder, 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 MacSyFinder 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:

```

# macsyfinder 20200217.dev
# macsyfinder --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)
  - 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

```

(continues on next page)

(continued from previous page)

```

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', 2
↪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)
    - 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 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
- **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 20210428.dev
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py37/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_
→--previous-run tests/data/functional_test_gembase/ -o functional_test_gembase
# 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      used_in
GCF_000005845      GCF_000005845_000970      T4P_pilC      97      TFF-SF/
→T4P      GCF_000005845_T4P_106      3      1      0.556      7.
→800      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_106      3      1      0.556      7.
→800      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_106      3      1      0.556      7.
→800      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_106      3      2      0.556      7.
→800      1      T4P_pilW      accessory      187      3.3e-08      34
→500      0.625      0.401      6      80
```

(continues on next page)

(continued from previous page)

GCF_000005845	GCF_000005845_025690	T4P_fimT	2569	TFF-SF/
↪T4P	GCF_000005845_T4P_106	3	2	0.556 7.
↪800	1	T4P_fimT	accessory	156 2.5e-06 28.
↪500	0.939	0.397	5	66
GCF_000005845	GCF_000005845_026740	T4P_pilT	2674	TFF-SF/
↪T4P	GCF_000005845_T4P_106	3	-1	0.556 7.
↪800	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_106	3	-2	0.556 7.
↪800	1	T4P_pilD	mandatory	269 1.3e-87 294.
↪000	1.000	0.859	30	260 GCF_000005845_T2SS_83
GCF_000005845	GCF_000005845_030590	T4P_pilQ	3059	TFF-SF/
↪T4P	GCF_000005845_T4P_106	3	3	0.556 7.
↪800	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_106	3	3	0.556 7.
↪800	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_106	3	3	0.556 7.
↪800	1	T4P_pilM	accessory	259 1.1e-09 39.
↪300	0.988	0.598	8	162

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_gspO	2693	TFF-SF/T4P	GCF_
↪000005845_T4P_25	3	-2	0.556 7.800		

best_solution.tsv and all_best_solutions.tsv

Since MacSyFinder 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
- **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 20210428.dev
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py37/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_
↪--previous-run tests/data/functional_test_gembase/ -o functional_test_gembase
# 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      used_in
GCF_000005845      GCF_000005845_000970      T4P_pilC      97      TFF-SF/
↪T4P         GCF_000005845_T4P_106      3      1      0.556      7.
↪800         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_106      3      1      0.556      7.
↪800         1      T4P_pilB      mandatory      461      8.9e-152      506.
↪100         0.948      0.850      62      453
```

(continues on next page)

(continued from previous page)

(continued from previous page)						
GCF_000005845	GCF_000005845_000990	T4P_pilA	99	TFF-SF/		
↪T4P	GCF_000005845_T4P_106	3	1	0.556	7.	
↪800	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_106	3	2	0.556	7.	
↪800	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_106	3	2	0.556	7.	
↪800	1	T4P_fimT	accessory	156	2.5e-06	28.
↪500	0.939	0.397	5	66		
GCF_000005845	GCF_000005845_026740	T4P_pilT	2674	TFF-SF/		
↪T4P	GCF_000005845_T4P_106	3	-1	0.556	7.	
↪800	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_106	3	-2	0.556	7.	
↪800	1	T4P_pilD	mandatory	269	1.3e-87	294.
↪000	1.000	0.859	30	260	GCF_000005845_T2SS_83	
GCF_000005845	GCF_000005845_030590	T4P_pilQ	3059	TFF-SF/		
↪T4P	GCF_000005845_T4P_106	3	3	0.556	7.	
↪800	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_106	3	3	0.556	7.	
↪800	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_106	3	3	0.556	7.	
↪800	1	T4P_pilM	accessory	259	1.1e-09	39.
↪300	0.988	0.598	8	162		

Example of *all_best_solutions.tsv* files

```
# macsyfinder 20210428.dev
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py37/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_
→--previous-run tests/data/functional_test_gembase/ -o functional_test_gembase
# 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 used_in
1           GCF_000005845 GCF_000005845_000970 T4P_
→pilC       97            TFF-SF/T4P   GCF_000005845_T4P_
→106        3             1            0.556          7.800         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_
→106        3             1            0.556          7.800         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_
→106        3             1            0.556          7.800         1            T4P_
→pilA       accessory     146          1.1e-19        71.200        0.859        0.
→473        5             73
```

(continues on next page)

(continued from previous page)

1	GCF_000005845	GCF_000005845_025680	T4P_				
→pilW	2568	TFF-SF/T4P	GCF_000005845_T4P_				
→106	3	2	0.556	7.800	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_				
→106	3	2	0.556	7.800	1	T4P_	
→fimT	accessory	156	2.5e-06	28.500	0.939	0.	
→397	5	66					
1	GCF_000005845	GCF_000005845_026740	T4P_				
→pilT	2674	TFF-SF/T4P	GCF_000005845_T4P_106	3	-		
→1	0.556	7.800	1	T4P_			
→pilT	mandatory	326	1.1e-117	393.600	0.		
→944	0.979	3	321				
1	GCF_000005845	GCF_000005845_026930	T2SS_				
→gspO	2693	TFF-SF/T4P	GCF_000005845_T4P_106	3	-		
→2	0.556	7.800	1	T4P_			
→pilD	mandatory	269	1.3e-87	294.000	1.		
→000	0.859	30	260	GCF_000005845_T2SS_83			
1	GCF_000005845	GCF_000005845_030590	T4P_				
→pilQ	3059	TFF-SF/T4P	GCF_000005845_T4P_				
→106	3	3	0.556	7.800	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_				
→106	3	3	0.556	7.800	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_				
→106	3	3	0.556	7.800	1	T4P_	
→pilM	accessory	259	1.1e-09	39.300	0.988	0.	
→598	8	162					
1	GCF_000005845	GCF_000005845_029970	T2SS_				
→gspC	2997	TFF-SF/T2SS	GCF_000005845_T2SS_				
→82	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_029980	T2SS_				
→gspD	2998	TFF-SF/T2SS	GCF_000005845_T2SS_				
→82	1	1	0.857	9.000	1	T2SS_	
→gspD	mandatory	650	2e-72	243.100	0.995	0.	
→283	427	610					
1	GCF_000005845	GCF_000005845_029990	T2SS_				
→gspE	2999	TFF-SF/T2SS	GCF_000005845_T2SS_				
→82	1	1	0.857	9.000	1	T2SS_	
→gspE	mandatory	493	4e-196	652.300	0.995	0.	
→850	65	483					
1	GCF_000005845	GCF_000005845_030000	T2SS_				
→gspF	3000	TFF-SF/T2SS	GCF_000005845_T2SS_				
→82	1	1	0.857	9.000	1	T2SS_	
→gspF	mandatory	398	9.9e-131	436.400	0.		
→997	0.839	61	394				
1	GCF_000005845	GCF_000005845_030010	T2SS_				
→gspG	3001	TFF-SF/T2SS	GCF_000005845_T2SS_				
→82	1	1	0.857	9.000	1	T2SS_	
→gspG	mandatory	145	5.5e-39	133.000	1.		
→000	0.607	8	95				

(continues on next page)

(continued from previous page)

2	GCF_000006725	GCF_000006725_000270	T4P_				
→pilT	4269	TFF-SF/T4P	GCF_000006725_T4P_145	2	-		
→1	0.667	8.500	1	T4P_			
→pilT	mandatory	344	1.8e-172	573.700	0.		
→994	0.985	2	340				
2	GCF_000006725	GCF_000006725_001050	T4P_				
→fimT	4347	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→fimT	accessory	187	1.4e-19	70.900	0.955	0.	
→337	14	76					
2	GCF_000006725	GCF_000006725_001060	T4P_				
→pilV	4348	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→pilV	accessory	158	3.2e-12	47.200	1.000	0.	
→354	10	65					
2	GCF_000006725	GCF_000006725_001070	T4P_				
→pilW	4349	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→pilW	accessory	316	1.1e-14	55.400	0.625	0.	
→237	12	86					
2	GCF_000006725	GCF_000006725_001080	T4P_				
→pilX	4350	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→pilX	accessory	195	2.1e-11	44.500	1.000	0.	
→303	12	70					
2	GCF_000006725	GCF_000006725_001090	T4P_				
→pilY	4351	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→pilY	accessory	1230	8.3e-144	480.900	0.		
→998	0.415	608	1118				
2	GCF_000006725	GCF_000006725_001100	T4P_				
→pilE	4352	TFF-SF/T4P	GCF_000006725_T4P_				
→145	2	1	0.667	8.500	1	T4P_	
→pilE	mandatory	139	1.4e-17	64.700	0.833	0.	
→396	16	70					

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: 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 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 MacSyFinder 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 20210428.dev
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py37/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_
↳--previous-run tests/data/functional_test_gembase/ -o functional_test_gembase
```

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	1	0
GCF_000006725	0	1	1	0
GCF_000006745	1	1	2	1
GCF_000006765	0	3	1	0
GCF_000006845	0	0	1	0
GCF_000006905	0	1	0	1
GCF_000006925	0	0	1	0
GCF_000006945	0	0	1	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 = '<macsyfinder_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_000005845	0	1	1	0	0
2	GCF_000006725	0	1	1	0	0
3	GCF_000006725	0	1	1	0	0
4	GCF_000006745	1	1	2	1	0
5	GCF_000006745	1	1	2	1	0
6	GCF_000006745	1	1	1	1	0
7	GCF_000006765	0	3	1	0	1
8	GCF_000006845	0	0	1	0	0
9	GCF_000006905	0	1	0	0	1
10	GCF_000006925	0	0	1	0	0
11	GCF_000006945	0	0	1	0	0

rejected_clusters.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 MacSyFinder 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.

```
# macsyfinder 20200511.dev
# /macsyfinder --sequence-db data/base/GCF_000006745.fasta --models TFF-SF all --
↪models-dir data/models/ --db-type gembase -w 4
# Rejected clusters:

Cluster:
- model: T4P
- hits: (GCF_000005845_025680, T4P_pilW, 2568), (GCF_000005845_025690, T4P_fimT, ↪
↪2569)
Cluster:
- model: T4P
- hits: (GCF_000005845_026930, T2SS_gspO, 2693)
Cluster:
- model: T4P
- hits: (GCF_000005845_030080, T2SS_gspO, 3008)
These clusters have 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)
These clusters have been rejected because:
The quorum of mandatory genes required (3) is not reached: 0
The quorum of genes required (3) is not reached: 2
=====
```


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 MacSyFinder, 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.

```
# macsyfinder 20201028.dev
# macsyfinder --sequence-db tests/data/base/one_replicon.fasta --db-type unordered --
↳models-dir tests/data/models -m TFF-SF T4P_single_locus
# 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
```

(continues on next page)

(continued from previous page)

```

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)
- 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='#')

```

```

# macsyfinder 20201028.dev
# macsyfinder --sequence-db tests/data/base/one_replicon.fasta --db-type unordered --
↳models-dir tests/data/models -m TFF-SF T4P_single_locus
# Likely Systems found:

replicon    hit_id  gene_name      hit_pos model_fqn      sys_id  sys_wholeness  hit_
↳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_000006845_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_000006845_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_000006845_014280    T4P_pilD      1428    TFF-SF/T4P_single_locus_
↳Unordered_T4P_single_locus_1    0.889    T4P_pilD      mandatory    264    1.
↳8e-82 272.300 1.000    0.829    28    264

```

(continues on next page)

(continued from previous page)

Unordered	GCF_000006845_001060	T4P_pilN	106	TFF-SF/T4P_single_locus_
↪Unordered_T4P_single_locus_1	0.889	T4P_pilN		mandatory 199 2.
↪3e-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.

```
# macsyfinder 20201113.dev
# macsyfinder --sequence-db tests/data/base/one_replicon.fasta --db-type unordered --
↳models-dir tests/data/models -m TFF-SF all
# 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 MacSyFinder.

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/macsyfinder_results/
    macsyfinder-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 MacSyFinder's execution:

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

1.1.2 MacSyFinder functioning

Macromolecular models

MacSyFinder 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 MacSyFinder's modelling scheme and the section on [Functioning](#) for the principles of the MacSyFinder's search engine.

A **MacSyFinder 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 [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 **macsy-model package** NEW in V2.

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

Installing models

How to install new models

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

`macsydata <subcommand> [options]`

The main sub-commands are

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

macsydata is NEW in V2

Where the models are located

MacSyFinder looks at several locations to find macsy-models.

system-wide installation

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

user-wide installation

If you don't own rights to install system-wide, you can install models in the MacSyFinder's cache located in your home: `$HOME/.macsyfinder/data/`. *macsydata* 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 specify a specific location with the `--models-dir` [command-line option](#). The path must point at a directory that contains macsy-model packages as described [above](#).

MacSyFinder's search engine

Functioning overview

MacSyFinder is run from the command-line using 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

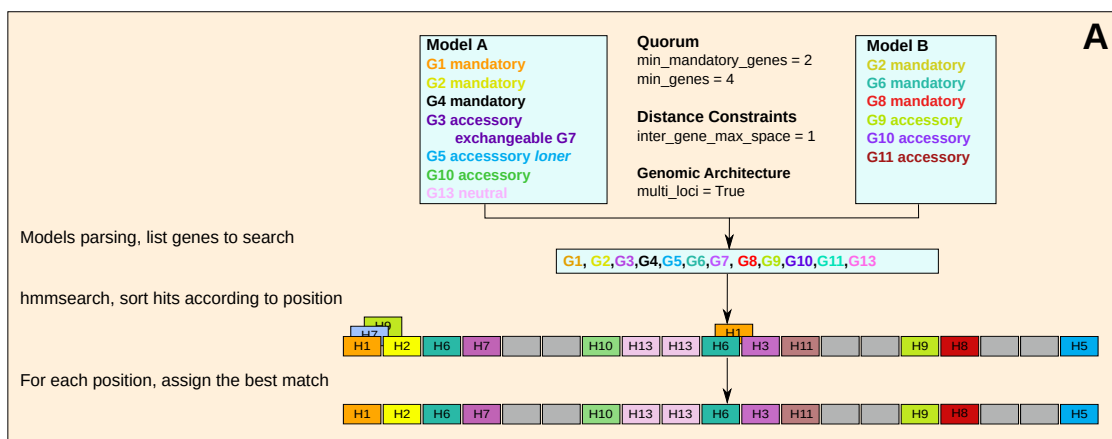
Initially, MacSyFinder **searches for the components** of the *System(s)* to detect by sequence similarity search.

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 MacSyFinder'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 (`-w` command-line option). See the [Command-line options](#), 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).

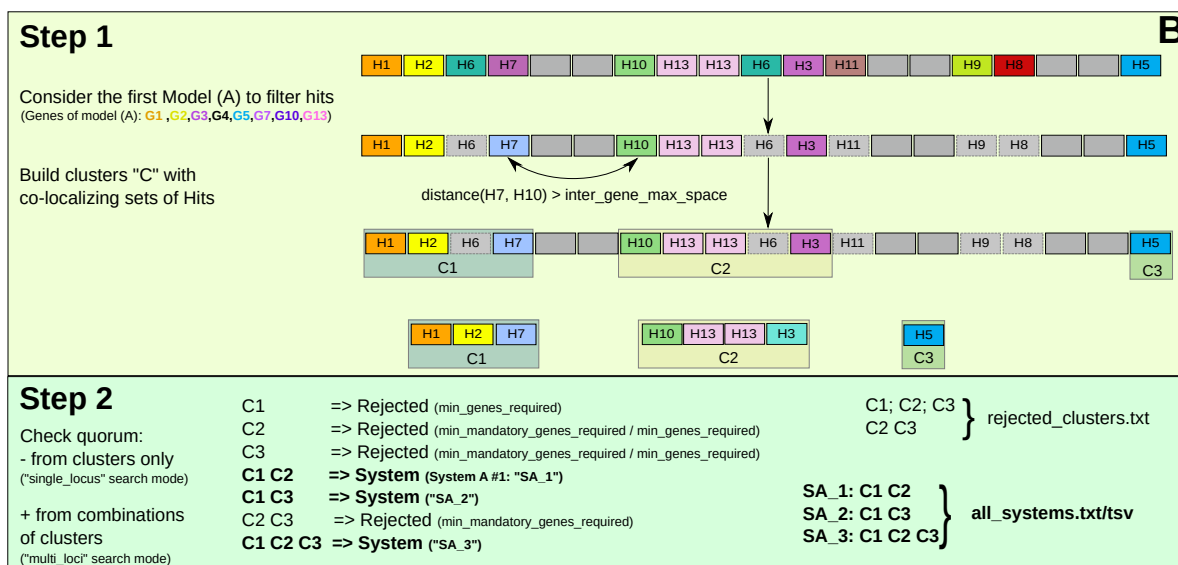
In the case of **ordered datasets** (*ordered_replicon* or *gembase* search mode), the hits are filtered to keep only hits related to the system's model we are looking for. These hits are used to build **clusters of co-localized genes** as defined in the *macsy-model files*. These clusters are then screened 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. MacSyFinder searches for sets of contiguous hits to build **clusters**, following the (**co-localization criterion**) for each replicon, as defined in the MacSyFinder'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_clusters.txt* file.

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 MacSyFinder generates the final *output files*.

C. Computing candidate Systems' scores (ordered mode)

This step only applies to the most powerful search mode, i.e., on **ordered datasets**. The whole step is NEW in V2

The **new search engine** implemented since version 2.0 of MacSyFinder better 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 function `macsypy.system.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_clusters.txt* output file.

- 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:

Scoring scheme reminder:

Clusters:

mandatory: +1.0
accessory: +0.5
neutral: +0.0
exchangeable: *0.75

Adjunction rules for redundant components ("Penalty"):

mandatory: -1.5
accessory: -1.5

Compute Systems' scores given Model (A):

Model A

G1 mandatory

G2 mandatory

G4 mandatory

G3 accessory

exchangeable G7

G5 accessory loner

G10 accessory

G13 neutral

SA_1

H1 H2 H7

C1

H10 H13 H13 H12

C2

SA_2

H1 H2 H7

C1

H5

C3

SA_3

H1 H2 H7

G1

H10 H13 H13 H12

C2

H5

C3

C1 score

C2 score

"Penalty"

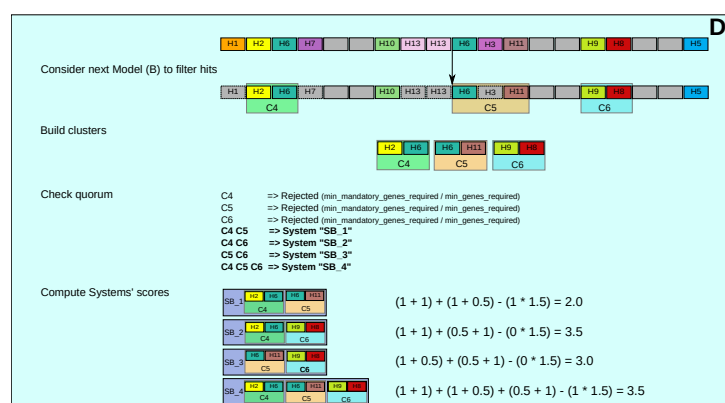
$(1 + 1 + 1 \times 0.75) + (0.5 + 0.5) - (0) = 3.75$

$(1 + 1 + 1 \times 0.75) + (0.5) - (0) = 3.25$

$(1 + 1 + 1 \times 0.75) + (0.5 + 0.5) + (0.5) - (0) = 4.25$

C

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



This search for candidate *Systems* 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)

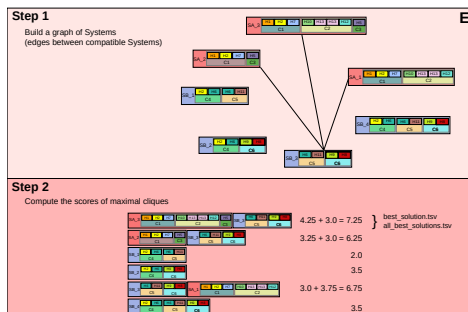
At the end of the previous step MacSyFinder has computed all potential *Systems* present in the replicon, made of combinations of Clusters and *loner* components that fulfill the model's requirements, which are themselves made of a subset of Hits (remember, Hits are at 1st filtered and treated separately 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*.



1.1.3 Frequently Asked Questions

Frequently Asked Questions

How to report an issue?

If you encounter a problem while running MacSyFinder, 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 MacSyFinder used

- the exact error message, and if applicable, the *macsyfinder.log* and *macsyfinder.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 cite MacSyFinder and published macy-models?

- [Abby et al. 2014](#), *PLoS ONE* for the **general principles of MacSyFinder** (version 1), and the corresponding set of Cas systems (CasFinder, 1st version).
- [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 (TXSS-can: 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.

What do MacSyFinder command lines look like?

Here are a few examples of command line formation:

To browse interactive help:

```
macsyfinder -h
```

The minimal command line, to search all systems with models from the “TFF-SF” set of models (installed with *macsydata*):

```
macsyfinder --db-type ordered_replicon --sequence-db genome.fasta  
--models TFF-SF all
```

To search for several systems (ModelA and ModelB) from the “model_family” set of models that can be found in the “./my-models” folder:

```
macsyfinder --db-type ordered_replicon --sequence-db genome.fasta  
--models model_family ModelA ModelB --models-dir ./my-models
```

To alter the search parameters and allow a maximal distance between components of 20 for the T2SS and 15 for the Tad pilus:

```
macsyfinder --db-type ordered_replicon --sequence-db genome.  
fasta --models TFF-SF all --inter-gene-max-space T2SS 20  
--inter-gene-max-space Tad 15
```

To alter the search parameters and allow the Tad pilus to be made of multiple loci:

```
macsyfinder --db-type ordered_replicon --sequence-db genome.fasta
--models TFF-SF all --multi-loci Tad
```

In *gembase* or *ordered_replicon* mode *macsyfinder* need to index the sequence-db. By default, this index is write beside the sequence-db file. But sometimes the directory where the sequence-db is located is not writable, in centralized shared data in multi user environnement for instance. To avoid to copy sequences in other location, you could specify an alternate directory for the index with `-index-dir` (This directory must exists):

```
macsyfinder --db-type ordered_replicon --sequence-db genome.fasta
--index-dir my-indexes --models TFF-SF all
```

See also the [MacSyFinder Quick Start](#) section for more examples.

What search mode to be used?

Depending on the type of dataset you have, you will have to adapt MacSyFinder'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 command-line options [here](#) and on MacSyFinder's functioning [here](#).

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.

How to search for multiple systems at once?

- It is possible to search for only some systems from a macsy-model package. In this case, the command-line should be formed as follows:

```
macsyfinder --models TXSS Flagellum T2SS --sequence-db mygenomes.fasta --db-type_
↳ gembase
```

This would run the search of the systems “Flagellum” and “T2SS” in the dataset “mygenomes.fasta”.

- To run the search of all the models contained in a macsy-model package, use the following:

```
macsyfinder --models TXSS all --sequence-db mygenomes.fasta --db-type gembase
macsyfinder --models CRISPRCas all --sequence-db mygenomes.fasta --db-type gembase
macsyfinder --models CRISPRCas/typing all --sequence-db mygenomes.fasta --db-type_
↳ gembase
```

You can see that the *all* keyword can not only be applied to an entire macsy-model package and its entire hierarchy, but can also be ran on all the systems from a macsy-model sub-directory.

When can the option *–previous-run* be used?

The option *–previous-run* enables to avoid running the HMM profile search and the hits extraction when the set of systems to search and the replicons to analyse are exactly the same between runs. This enables to alter the features of the systems to be searched for, i.e. basically any feature found in the XML file of the corresponding models:

- the maximal distance allowed between components to be considered as part of a same locus *–inter-gene-max-space*
- the minimal number of components to be found to infer a full system *–min-mandatory-genes-required* and *–min-genes-required*
- the general genomic architecture of the system *–multi-loci*

This also means that there are a number of options that are incompatible with *–previous-run*, including:

```
--config, --sequence-db, --profile-suffix, --res-extract-suffix, --e-value-res, --db-
↳ type, --hmmer
```

Which output file to be used to get ONE solution?

Since version 2 of MacSyFinder, a combinatorial exploration of the possible sets of systems is performed. A scoring scheme has been set up to differentiate between solutions, in order to provide the user with the most complete set of systems as possible given the searched models. This score is maximal for the “best solution”. This also means that some solutions might get the same maximal score. In this case, one can wonder how to find all the equivalent solutions, and an other, how to simply pick one solution among the best, whichever it is. We thus propose several kind of *output files*.

- All equivalent best solutions are found in the *all_best_solutions.tsv* file.
- One best solution is given in the *best_solution.tsv* file.

Note: For those more familiar with the output files from MacSyFinder v1, the file *best_solution.tsv* is the closest from the previous output file *macsyfinder.report*.

Where to find MacSyFinder models?

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

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

What are the rules for options precedence?

MacSyFinder 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 *macsyfinder.conf* file either in */etc/macsyfinder/* or in *virtualenv/etc/macsyfinder/* in case of a *virtualenv* this configuration affects only the MacSyFinder version installed in this *virtualenv*
- **home:** the *~/macsyfinder/macsyfinder.conf* file
- **model:** the *model_conf.xml* file at the root of the model package
- **project:** the *macsyfinder.conf* file found in the directory where the *macsyfinder* command was run
- **cfgfile:** any configuration file specified by the user on the command line (conflicts with the *-previous-run* option)
- **previous-run:** the *macsyfinder.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

MODELLER GUIDE

2.1 Modeller Guide

2.1.1 Modelling Systems with MacSyFinder

Models Package

MacSyFinder 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 MacSyFinder's modelling scheme and the section on *Functioning* for the principles of the MacSyFinder's search engine.

A **MacSyFinder 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 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 **macy-model package** NEW in V2.

Structure of a macy-model package

A macy-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
```

If the package contains sub-families:

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

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

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 *macsydata 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 [Creative Commons](#) *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 *macsydata*
  ↳ searches) (required)
vers: The package version (required)
cite: The publication(s) to cite by the user when the package is used (optional, used
  ↳ by `macsydata 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, pT4SSt, pT4SSi, T5aSS, T5bSS, T5bSS, T6SSi, T6SSii, T6SSiii,
↳Flagellum, Tad, T9SS).
vers: 0.0a1
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: 2014-2021, Institut Pasteur, CNRS
```

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

Note:

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

Model configuration

The modeler has the possibility to specify some options that are specific to their package, different than the MacSyFinder 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_value_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 *macsyfinder.conf* file either in */etc/macsyfinder/* or in *virtualenv/etc/macsyfinder/* in case of a *virtualenv* this configuration affects only the MacSyFinder version installed in this *virtualenv*
- **home:** the *~/macsyfinder/macsyfinder.conf* file
- **model:** the *model_conf.xml* file at the root of the model package
- **project:** the *macsyfinder.conf* file found in the directory where the *macsyfinder* command was run
- **cfgfile:** any configuration file specified by the user on the command line (conflicts with the *–previous-run* option)
- **previous-run:** the *macsyfinder.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>
    <loner_multi_system>0.7</loner_multi_system>
  </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

MacSyFinder 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 MacSyFinder's modelling scheme and the section on *Functioning* for the principles of the MacSyFinder's search engine.

A **MacSyFinder 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 [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 **macsy-model package** NEW in V2.

Principles, and how to write macsy-models definitions

Macsy-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 macsy-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.

Macsy-model Components

Four distinct **types of components** can be used to model the System's content. Components correspond to Gene objects in MacSyFinder'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

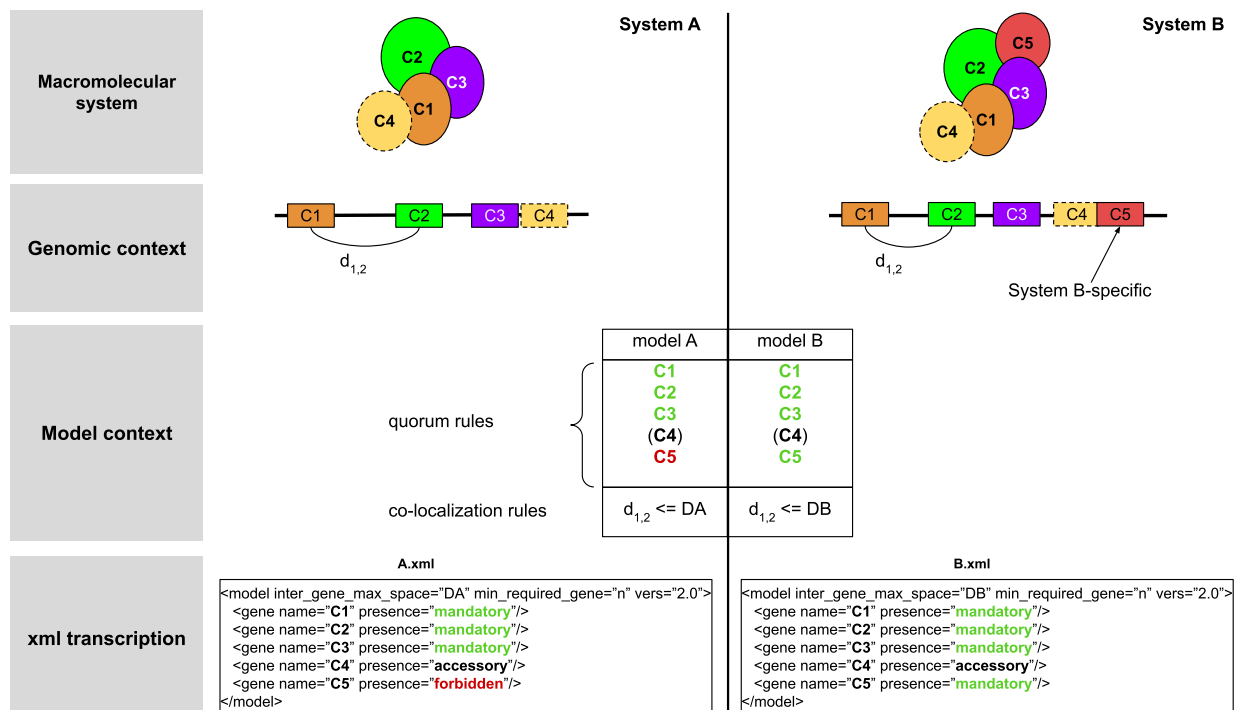
Beyond its list of Components, a MacSyFinder's model of a System is defined by the genomic organization of its components. This genomic organization can be defined in several ways:

- 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 macsy-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 MacSyFinder, 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 whaleness 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 systems’ occurrences - and thus be considered part of several systems. (*default false*).
- **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 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 macsy-model definition in XML:

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

fault 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.

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

Helper Tool

macyprofile

To help develop new models we provide the tool *macyprofile* which is to be used as post treatment.

It is ran over a previous macyfinder 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 MacSyFinder run.
- it writes down the results in a file in *tsv* format *hmm_coverage.tsv*.

```
usage: macyprofile [-h] [--coverage-profile COVERAGE_PROFILE]
                  [--i-evalue-sel I_EVALUATE_SEL]
                  [--best-hits {score,i_eval,profile_coverage}] [-p PATTERN]
                  [-o OUT] [-f] [-V] [-v] [--mute]
                  previous_run
```

```

      *           *           *           *           * *           *
*           *           *           *           *           *
**      *      *      *      *           *           *
      _ _      *           _ _           _ _      _ _      *
```

(continues on next page)

(continued from previous page)

```

      | \ / | _ _ _ / _ | _ _ | _ \ _ _ _ / _ ( ) | _ _
      | \| / | / _ ` | / _ \| _ \| | | | | ) | ' _ / _ \| | | | / _ \
      | | | | ( _ | ( _ _ ) | | | | _ / | | | ( ) | _ | | | _ /
      | _ | _ | \ _ , _ | \ _ | _ _ / \ _ , | _ | _ | \ _ / | _ | _ | \ _ |
          *               | _ _ / *               *
*      *      * *      *      **      *      *      *      *
*      *      *      *      *      *      *
          *               *      *      *      *

```

MacSyProfile - MacSyFinder profile helper tool

positional arguments:

- previous_run The path to a macsyfinder results directory.

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 for systems detection. (default no threshold)
- i-evalue-sel I_EVALUATE_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 macsyfinder.log) (default: False)

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

For instance

```
macsyprofile macsyfinder-2021XXXX_XX-XX-XX
```

will analyse the HMMER raw outputs stored in *macsyfinder-2021XXXX_XX-XX-XX/hmmer_results* directory and the results will be stored in *macsyfinder-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 MacSyFinder 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:

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

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

```

macsyprofile --pattern 'ComM_com?C' -f macsyfinder-2021XXXX_XX-XX-XX
parsing macsyfinder-2021XXXX_XX-XX-XX/hmmer_results/ComM_comEC.search_hmm.out
parsing macsyfinder-2021XXXX_XX-XX-XX/hmmer_results/ComM_comGC.search_hmm.out
found 16 hits
result is in 'macsyfinder-2021XXXX_XX-XX-XX/hmm_coverage.tsv'

```

The *macsyprofile* 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.

```

# macsyprofile 2.0rc1
# macsyprofile --pattern ComM* --coverage-profile 0.5 macsyfinder-2021202_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 *macsyprofile* 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)

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 `macsydata check` command. You have to run it from within the folder containing your package files. It will report:
 - everything is clear: *macsydata* displays the next step totake 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. *macsydata checks* only tagged packages. It is also the duty of the model provider to setup a tag with the same name as the version in the *metadata.yml* file. It is recommended to follow a versioning scheme describe here: <https://www.python.org/dev/peps/pep-0440/#public-version-identifiers>
 3. When your pull request (PR) is accepted, the model package becomes automatically available to the community through the *macsydata* 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 *macsydata* tool.

Note: `macsydata 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 `macsydata check` outputs:

Your package is syntactically correct:

```
macsydata 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
    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:

```

macsydata 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.
macsydata 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:

```

macsydata 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 MacSyFinder'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 MacSyFinder 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 T5SS

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 MacSyFinder 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.

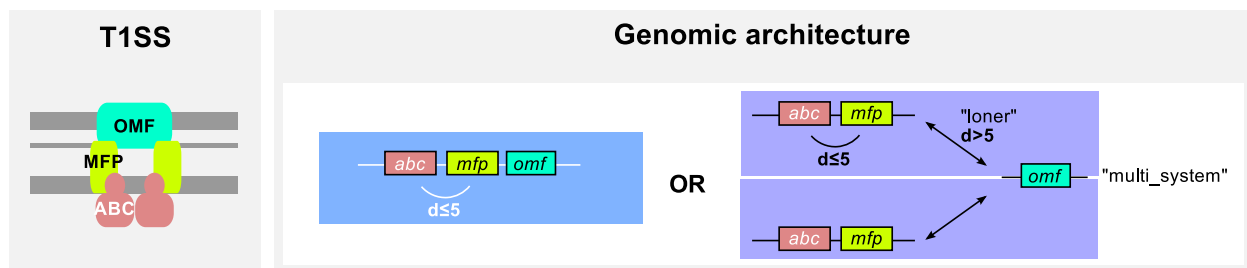
3. Describing their genetic architecture

According to the litterature, 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 MacSyFinder 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 MacSyFinder 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](#).

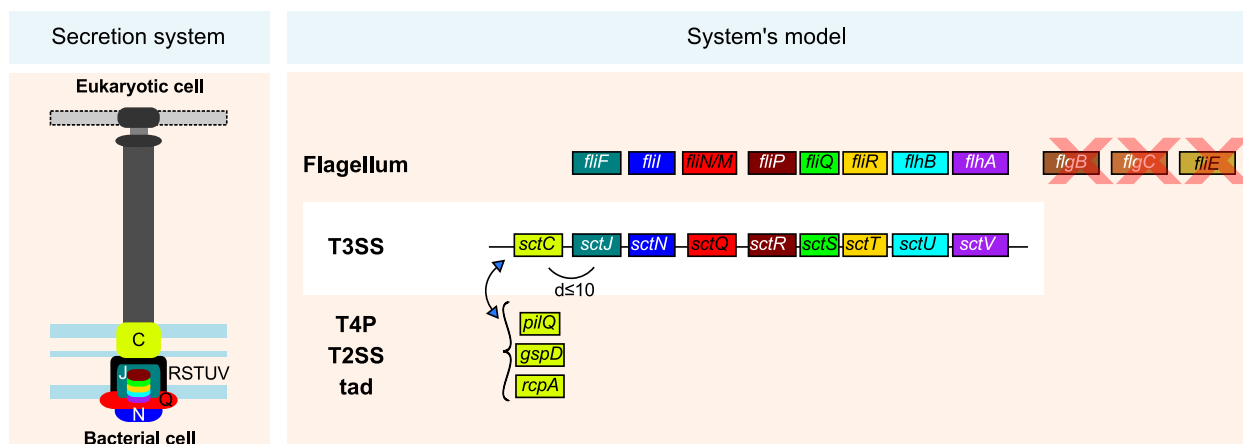
2. Dealing with components with varied evolutionary origins

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, MacSyFinder 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"/>
```

(continues on next page)

(continued from previous page)

```

<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 Carrying models from v1 to v2

Carrying models from v1 to v2

Models from v1 are not compatible straight away with v2. For those who had designed MacSyFinder's models for Version 1 and would like to carry them for Version 2, here are the changes to consider:

- the keyword “system” was changed: `<system>` ::arrow:: `<model>`
- the keyword `<system_ref>` was removed. For a given systems' package, each gene has to be defined only once in a macsy-model. There is no need anymore to reference which model it is from, when used as a component in another system's model.
- now the version of the macsy-models' type has to be documented as a feature of the “model” keyword, like this:
`vers = “2.0”`
- the following keywords have been replaced (but see *below* for more details):
 - homologs => exchangeables
 - analogs => exchangeables

Note: “exchangeable” is not a feature anymore, but is replaced by the keyword “exchangeables”.

Note: These changes in the grammar used to specify model is also accompanied by a change on how to organize folders with models and profiles. In particular, the new file architecture enables an *easier shipping* of the developed macsy-models. See *here* for more details.

Here follow some examples of updates from v1 to v2.

1. A very simple model.

TISS.xml under **v1**:

```

<system inter_gene_max_space="5" min_mandatory_genes_required="3" min_genes_required=
↪ "3">
  <gene name="T1SS_abc" presence="mandatory"/>
  <gene name="T1SS_mfp" presence="mandatory"/>
  <gene name="T1SS_omf" presence="mandatory" loner="1" multi_system="1"/>
</system>

```

TISS.xml under **v2**:

```
<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>
```

Note: In a nutshell, the minimal changes from v1 to v2 for a simple macsy-model listing components are the following:

- `<system> => <model>`
- `vers = "2.0"`

2. A model with homologs.

Tad.xml under **v1**:

```
<system inter_gene_max_space="5" min_mandatory_genes_required="4" min_genes_required=
↪ "6" multi_loci="0">
  <gene name="Tad_rcpA" presence="mandatory">
    <homologs>
      <gene name="T2SS_gspD" system_ref="T2SS"/>
      <gene name="T4P_pilQ" system_ref="T4P"/>
      <gene name="T3SS_sctC" system_ref="T3SS"/>
    </homologs>
  </gene>
  <gene name="Tad_tadA" presence="mandatory"/>
  <gene name="Tad_tadB" presence="mandatory"/>
  <gene name="Tad_tadC" presence="mandatory"/>
  <gene name="Tad_tadV" presence="mandatory"/>
  <gene name="Tad_tadZ" presence="mandatory"/>
  <gene name="Tad_flp" presence="accessory"/>
  <gene name="Tad_tadE" presence="accessory"/>
  <gene name="Tad_tadF" presence="accessory"/>
</system>
```

Tad.xml under **v2**:

```
<model inter_gene_max_space="5" min_mandatory_genes_required="4" min_genes_required="6"
↪ " multi_loci="0" vers="2.0">

  <gene name="Tad_rcpA" presence="mandatory"/>
  <gene name="Tad_tadA" presence="mandatory"/>
  <gene name="Tad_tadB" presence="mandatory"/>
  <gene name="Tad_tadC" presence="mandatory"/>
  <gene name="Tad_tadV" presence="mandatory"/>
  <gene name="Tad_tadZ" presence="mandatory"/>
  <gene name="Tad_flp" presence="accessory"/>
  <gene name="Tad_tadE" presence="accessory"/>
  <gene name="Tad_tadF" presence="accessory"/>

</model>
```

Note: The *homologs* and *analogs* keyword having disappeared, it is not necessary anymore to list homologous components (e.g., those that may match several HMM profiles during the sequence similarity search), unless they are *exchangeables*.

3. A model with exchangeable homologs.

T3SS.xml under **v1**:

```
<system inter_gene_max_space="10" min_mandatory_genes_required="7" min_genes_required=
↪ "7" multi_loci="1">
  <gene name="T3SS_sctC" presence="mandatory" exchangeable="1">
    <homologs>
      <gene name="T2SS_gspD" system_ref="T2SS"/>
      <gene name="T4P_pilQ" system_ref="T4P"/>
      <gene name="Tad_rcpA" system_ref="Tad"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctJ" presence="mandatory">
    <homologs>
      <gene name="Flg_sctJ_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctN" presence="mandatory">
    <homologs>
      <gene name="Flg_sctN_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctQ" presence="mandatory">
    <homologs>
      <gene name="Flg_sctQ_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctR" presence="mandatory">
    <homologs>
      <gene name="Flg_sctR_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctS" presence="mandatory">
    <homologs>
      <gene name="Flg_sctS_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctT" presence="mandatory">
    <homologs>
      <gene name="Flg_sctT_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctU" presence="mandatory">
    <homologs>
      <gene name="Flg_sctU_FLG" system_ref="Flagellum"/>
    </homologs>
  </gene>
  <gene name="T3SS_sctV" presence="mandatory">
    <homologs>
```

(continues on next page)

(continued from previous page)

```

        <gene name="Flg_sctV_FLG" system_ref="Flagellum"/>
    </homologs>
</gene>
<gene name="Flg_fliE" presence="forbidden" system_ref="Flagellum"/>
<gene name="Flg_flgB" presence="forbidden" system_ref="Flagellum"/>
<gene name="Flg_flgC" presence="forbidden" system_ref="Flagellum"/>
</system>

```

T3SS.xml under v2:

```

<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>

```

Note:

- As only the secretin component ‘T3SS_sctC’ was exchangeable in its role within T3SS with its homologs T2SS_gspD, T4P_pilQ and Tad_rcpA, these three components are now set as *exchangeables* (they can functionally *replace* the component ‘T3SS_sctC’), and all other *homologs* do not need to be listed anymore.
- The keyword *system_ref* is not needed anymore. Therefore, the **v2** definition of T3SS is way more compact than that for **v1**.

2.1.3 Frequently Asked Questions

Frequently Asked Questions

How to report an issue?

If you encounter a problem while running MacSyFinder, 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 MacSyFinder used
- the exact error message, and if applicable, the *macsyfinder.log* and *macsyfinder.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?

MacSyFinder 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.

DEVELOPER GUIDE

3.1 Developer Guide

3.1.1 MacSyFinder implementation overview

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

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. Technically it's a list of Systems. 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.

MacSyFinder project structure

A brief overview of the files and directory constituting the MacSyFinder project

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

etc This directory contains a template to configure macsyfinder. It's allow to set some configuration available for each run and avoid to specify them at each run on the command line. This file is in *ini* format.

macsypy This the MacSyFinder python library Inside macsypy there is a subdirectory *scripts* which are the entry points for *macsyfinder* and *macsydata*

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

utils Contains a binary *setsid* needed macsyfinder to parallelize some steps. Usually *setsid* is provides by the system, but some macOS version does not provide it.

CITATION.yml A file indicating how to cite macsyfinder in yaml format.

CONTRIBUTORS A file containing the list of code contributors.

CONTRIBUTING A guide on how to contribute to the project.

COPYRIGHT The macsyfinder copyrights.

COPYING The licencing. MacSyFinder is released under GPLv3.

README.md Brief information about the project.

requirements.txt The list of python dependencies needed by macsyfinder. do not forget to install hmmsearch which is not handle by python packet manager *pip*

requirements_dev.txt The list of extra dependencies needed if you want to contribute to the code.

setup.py The installation recipe.

MacSyFinder architecture overview

An overview of the main classes.

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

MacSyFinder functioning overview

In this section I'll give you an idea of the macsyfinder functioning at very high grain coarse.

As all program the entrypoint is the main function The goal of *macsyfinder.main* is to parse the command line. Then to creates a *configuration* object and also initialize the logger. After that it call *main_search_systems* which contains the macsyfinder logic

The first *main_search_systems* task is to create models asked by the user on the command line. So a *DefinitionParser* is instantiated and the *ModelBank* and *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.

- **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.RejectedCluster*.

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

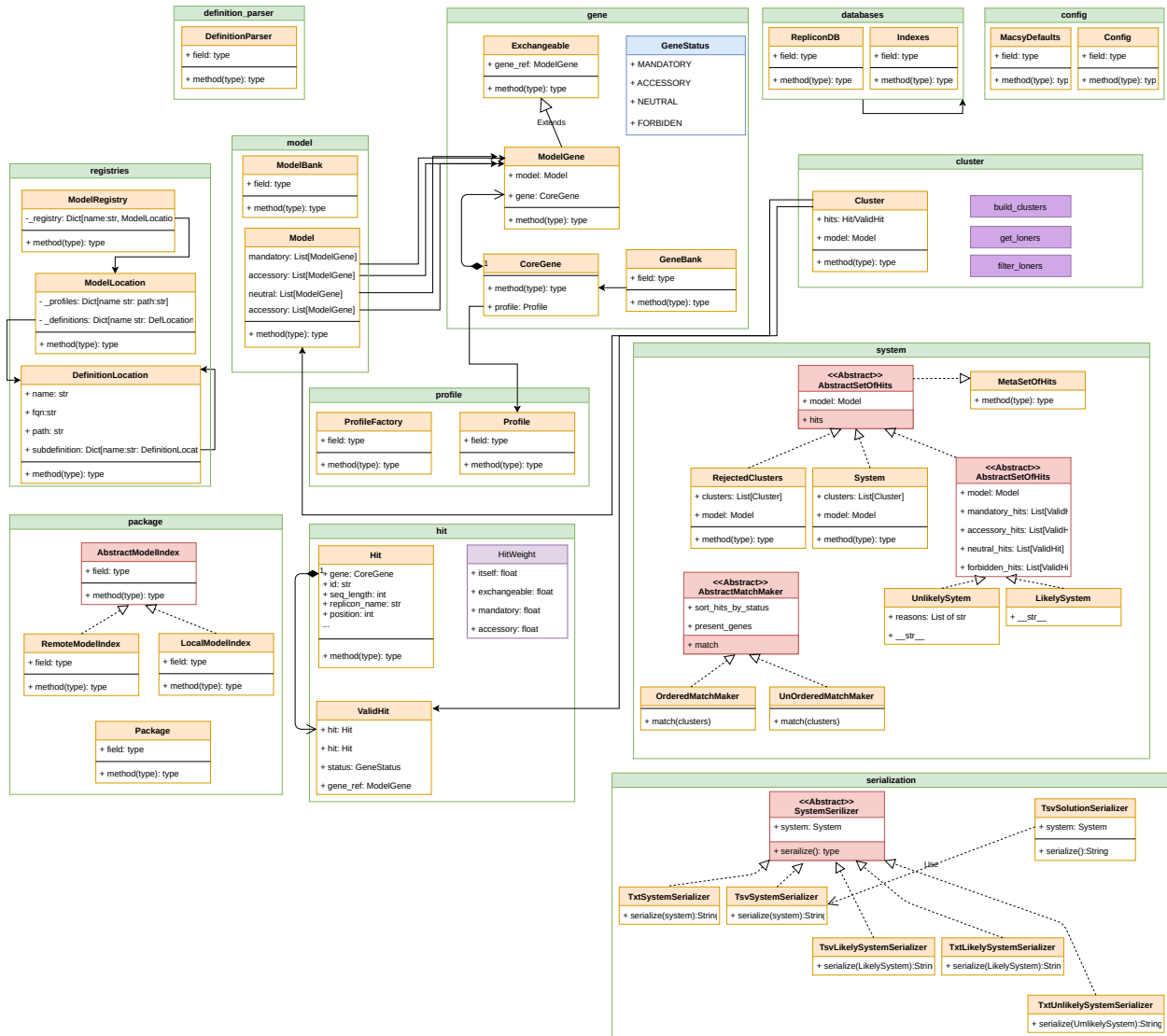


Fig. 1: The macsyfinder classes diagram. The classes are not details. only the main attributes allowing us to understand the interaction are mentioned.

- 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

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

Among all this potential systems, MSF compute the best combination. `macsypy.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 configuration file, or the command-line (`-d` parameter). 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 (`macsypy.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 (`macsypy.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 CoreGene must be created by using the GeneBank factory.

A ModelGene (`macsypy.gene.ModelGene`) which encapsulate a CoreGene and is linked to a Model. Instead CoreGene, several ModelGene with the same name may coexists in `macsyfinder`, in different Models and hold different values for attributes as `inter_gene_max_space`, ... Each ModelGene points out its Model of origin (`macsypy.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 `macsypy.gene.GeneBank` object. The `gene_bank` and `model_bank` are filled by the `system_parser` (`macsypy.definition_parser.ModelDefinitionParser`)

The Profile object

Each “CoreGene” component corresponds to a “Profile”. The “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” (*macsypy.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*).

3.1.2 MacSyFinder API documentation

configuration

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

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

Config

class *macsypy.config.Config* (*defaults, parsed_args*)

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

__init__ (*defaults, parsed_args*)

Store macsyfinder 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/macsyfinder/macsyfinder.conf
 user home conf = ~/.macsyfinder/macsyfinder.conf
 model conf = model_conf.xml at the root of the model package
 project conf = macsyfinder.conf where the analysis is run
 previous run = macsyfinder.conf in previous run results dir
 command line = the options set on the command line

Parameters

- **defaults** (a *MacsyDefaults* object) –
- **parsed_args** (a *argspace.Namespace* object) – the command line arguments parsed

__weakref__

list of weak references to the object (if defined)

_config_file_2_dict (*file*)

Parse a configuration file in ini format in dictionary

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

Returns the parsed options

Return type dict

_set_command_line_config (*parsed_args*)

Parameters **parsed_args** (*argparse.Namespace* object.) – the argument set on the command line

_set_db_type (*value*)

set value for 'db_type' option

Parameters **value** (*str*) – the value for db_type, allowed values are : 'ordered_replicon', 'gembase', 'unordered'

Raises **ValueError** – if value is not allowed

_set_default_config ()

set the value coming from MacsyDefaults

_set_inter_gene_max_space (*value*)

set value for 'inter_gene_max_space' 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_max_nb_genes (*value*)

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*)

set value for 'min_genes_required' 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_mandatory_genes_required (*value*)

set value for 'min_mandatory_genes_required' 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_model_config (*model_conf_path*)

Set the options from the model package model_conf.xml file

Parameters **model_conf_path** (*str*) – The path to the model_conf.xml file

_set_models (*value*)

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

if value come from command_line ['modell', 'def1', 'def2', 'def3']

if value come from config file 'set_1', 'T9SS T3SS T4SS_typeI') [(model_family, [def_name1,...]),...]

_set_models_dir (*path*)

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

_set_multi_loci (*value*)

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

_set_options (*options*)

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*)

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

Parameters prev_config_path –

_set_project_config_file (*config_path*)

Set the options from the `macsyfinder.conf` present in the current directory

Parameters config_path (*str*) – the path to the configuration file

_set_replicon_topology (*value*)

set the default replicon topology

Parameters value (*str*) – 'circular' or 'linear'

_set_sequence_db (*path*)

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

_set_system_models_dir (*value*)

Parameters value –

Returns

_set_system_wide_config (*config_path*)

set the options from the system wide configuration file

Parameters config_path (*str*) –

_set_topology_file (*path*)

test if the path exists and set it in config

Parameters path (*str*) – the path to the topology file

_set_user_config_file (*config_path*)

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

Parameters config_path (*str*) – The path to the configuration path

_set_user_wide_config (*config_path*)

Set the options from the `~/macsyfinder/macsyfinder.conf` file

Parameters config_path (*str*) – The path to the `~/macsyfinder/macsyfinder.conf`

_str_2_tuple (*value*)

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

Parameters **value** (*str*) – the string to parse

Returns

Return type [(model_fqn, int), ..]

hit_weights ()

Returns the options used in scoring systems (mandatory_weight, accessory_weight, itself_weight, exchangeable_weight, loner_multi_system_weight)

Return type dict

hammer_dir ()

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

inter_gene_max_space (*model_fqn*)

Parameters **model_fqn** (*str*) – the model fully qualified name

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

Return type int or None

log_level ()

Returns the verbosity output level

Return type int

max_nb_genes (*model_fqn*)

Parameters **model_fqn** (*str*) – the model fully qualified name

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

Return type int or None

min_genes_required (*model_fqn*)

Parameters **model_fqn** (*str*) – the model fully qualified name

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

Return type int or None

min_mandatory_genes_required (*model_fqn*)

Parameters **model_fqn** (*str*) – the model fully qualified name

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

Return type int or None

models_dir ()

Returns

multi_loci (*model_fqn*)

Parameters **model_fqn** (*str*) – the model fully qualified name

Returns True if the model is multi loci, False otherwise

Return type bool

out_dir ()

Returns the path to the directory where the results are stored

save (*path_or_buf=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()
alias to `config.Config.out_dir()`

class `macsypy.config.MacsyDefaults` (***kwargs*)
Handle all default values for macsyfinder. 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__ (***kwargs*)

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 (if defined)

model_conf_parser

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

ModelConfParser

class `macsypy.model_conf_parser.ModelConfParser` (*path*)
Handle `model_conf.xml` configuration file.

__init__ (*path*)

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

__weakref__
list of weak references to the object (if defined)

_get_model_conf_node()
Find the root of the document

Returns the document root of `model_conf`

_parse_section (*section_node*, *allowed_elements*)
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()

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*)

Parse the node 'filtering' containing the filtering options configuration

Parameters **filtering_node** (:class"*Et.ElementTree* object) – the node 'filtering'

Returns the configuration option/value about the filtering

Return type dict

parse_weights() (*weights_node*)

Parse the node 'weights' containing the scoring weight configuration

Parameters **weights_node** (:class"*Et.ElementTree* object) – the node 'weights'

Returns the configuration option/value about the scores

Return type dict

registries

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

registries API reference

class `macsypy.registries.DefinitionLocation` (*name=None*, *subdefinitions=None*,
path=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*)

Return self==value.

__gt__ (*other*)

Return self>value.

__hash__ ()

Return hash(self).

__init__ (*name=None*, *subdefinitions=None*, *path=None*)

Initialize self. See help(type(self)) for accurate signature.

__lt__ (*other*)

Return self<value.

__str__ ()

Return str(self).

__weakref__

list of weak references to the object (if defined)

add_subdefinition (*subdefinition*)

add new sub category of definitions to this definition

Parameters subdefinition (*DefinitionLocation* object) – the new definition to add as subdefinition.

```
class macsypy.registries.ModelLocation (path=None, profile_dir=None, def_dir=None, profile_suffix='.hmm', relative_path=False)
```

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

```
__eq__ (other)
```

Return self==value.

```
__gt__ (other)
```

Return self>value.

```
__init__ (path=None, profile_dir=None, def_dir=None, profile_suffix='.hmm', relative_path=False)
```

Parameters

- **path** (*str*) – if it's an installed model, path is the absolute path to a model family. otherwise path is None, and profile_dir and def_dir must be specified.
- **profile_dir** (*str*) – the absolute path to the directory which contains the hmm profiles files.
- **def_dir** (*str*) – The absolute path to the directory which contains the models definitions (xml files) or submodels.
- **profile_suffix** (*str*) – the suffix of hmm files
- **relative_path** (*bool*) – True if you want to work with relative path, False to work with absolute path.

Raise MacsypyError if path is set and profile_dir or def_dir is set

Raise MacsypyError if profile_dir is set but not def_dir and vice versa

```
__lt__ (other)
```

Return self<value.

```
__str__ ()
```

Return str(self).

```
__weakref__
```

list of weak references to the object (if defined)

```
_scan_definitions (model_def=None, def_path=None)
```

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

Parameters

- **model_def** (*DefinitionLocation*) – the current model definition to add new sub-model location
- **def_path** (*string*) – the absolute path to analyse

Returns a definition location

Return type *DefinitionLocation* object

```
_scan_profiles (path, profile_suffix='.hmm', relative_path=False)
```

Store all hmm profiles associated to the model

```
get_all_definitions (root_def_name=None)
```

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

Parameters `root_def_name` – string

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

Return type list of :class: `DefinitionLocation`` object

Raises **ValueError** – if `root_def_name` does not match with any definitions

get_definition (*fqn*)

Parameters `fqn` (*string.*) – 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/defintions/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.

Return type a `DefinitionLocation` object.

Raise `valueError` if `fqn` does not match with any model definition.

get_definitions ()

Returns

get_profile (*name*)

Parameters `name` (*string.*) – the name of the profile to retrieve (without extension).

Returns the absolute path of the hmm profile.

Return type string.

Raise `KeyError` if `name` does not match with any profiles.

get_profiles_names ()

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

Return type str

class `macsypy.registries.ModelRegistry`

scan canonical directories to register the different models available in global macsyfinder share data location (depending installation `/usr/share/data/models`) or can be overload with the location specify in the macsyfinder configuration (either in config file or command line)

__getitem__ (*name*)

Parameters `name` (*string*) –

Returns the model corresponding to name.

Return type `ModelLocation` object.

Raises **KeyError** – if name does not match any `ModelLocation` registered.

__init__ ()

Initialize self. See `help(type(self))` for accurate signature.

__str__ ()

Return `str(self)`.

__weakref__

list of weak references to the object (if defined)

add (*model_loc*)

Parameters *model_loc* (*ModelLocation* object) – the model location to add to the registry

models ()

Returns the list of models

Return type list of *ModelLocation* object

macsypy.registries.join_def_path (*args)

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

macsypy.registries.scan_models_dir (*models_dir*, *profile_suffix*='hmm', *relative_path*=False)

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 [*macsypy.registries.ModelLocation*, ...]

macsypy.registries.split_def_name (*fqn*)

Parameters *fqn* (*string*) – 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']

Return type list of string

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 ModelGene) 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 ModelGene (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.

DefinitionParser

```
class macsypy.definition_parser.DefinitionParser(cfg,    model_bank,    gene_bank,
                                                model_registry, profile_factory)
```

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

```
__init__(cfg, model_bank, gene_bank, model_registry, profile_factory)
```

Parameters

- **cfg** (*macsypy.config.Config* object) – the configuration object of this run
- **model_bank** (*macsypy.model.ModelBank* object) – the model factory
- **gene_bank** (*macsypy.gene.GeneBank* object) – the gene factory

- **model_registry** (`macsypy.registry.ModelRegistry` object) – The registry with all model location
- **profile_factory** (`macsypy.profil.ProfilFactory` object) – The profile factory

__weakref__

list of weak references to the object (if defined)

__check_syntax (*model_node*, *path*)

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

Parameters

- **model_node** (`Et.Element` object) – the node correponding to the model
- **path** (*str*) – the path of the definition.

Returns None

Raises *ModelInconsistencyError* – if an error is encountered in the document.

__create_model (*def_loc*, *model_node*)**Parameters**

- **def_loc** (`macsypy.registries.DefinitionLocation` object) – the definition location to parse.
- **model_node** (`Et.ElementTree` object.) – the node corresponding to the model.

Returns the model corresponding to the definition location.

Return type `macsypy.model.Model` object.

__fill_gene_bank (*model_node*, *model_location*, *def_loc*)

find all gene node and add them to the gene_bank

Parameters

- **model_node** (`Et.ElementTree` object.) –
 param model_node the node corresponding to the model.
- **model_location** (class:`macsypy.registries.ModelLocation` object.) –
- **def_loc** (*the node corresponding to the 'model' tag*) – a definition location to parse.

Returns None

__get_model_node (*def_loc*)

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

__parse_exchangeable (*node*, *gene_ref*, *curr_model*)

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

Parameters

- **node** (`xml.etree.ElementTree.Element` object.) – a “node” corresponding to the gene element in the XML hierarchy
- **gene_ref** (class:`macsypy.gene.ModelGene` object) – the gene which this gene is homolog to

- **curr_model** (*macsypy.model.Model* object) – the model being parsed .

Returns the gene object corresponding to the node

Return type *macsypy.gene.Exchangeable* object

_parse_genes (*model, model_node*)

Create genes belonging to the models. Each gene is directly added to the model in it's right category ('mandatory', 'accessory', ...)

Parameters

- **model** (*macsypy.model.Model* object) – the Model currently parsing
- **model_node** (:class"*Et.ElementTree* object) – the element 'model'

check_consistency (*models*)

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 (list of *class:macsypy.model.Model* object) – the list of models to check

Raise *macsypy.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*)

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 (list of *macsypy.registry.DefinitionLocation*) – 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 3 component classes:

- genes which are mandatory
- genes which are accessory
- genes which are forbidden

each genes can have homolgs or analogs. Analogs and Homolgs refer to genes described in an other model.

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

ModelBank

```
class macsypy.model.ModelBank
    Store all Models objects.

    __contains__(model)
        Implement the membership test operator
        Parameters model (macsypy.model.Model object) – the model to test
        Returns True if the model is in the Model factory, False otherwise
        Return type boolean

    __getitem__(fqm)
        Parameters fqm (string) – the fully qualified name of the model
        Returns the model corresponding to the fqm.
        Return type macsypy.model.Model object
        Raises KeyError – if the model corresponding to the name does not exists

    __init__()
        Initialize self. See help(type(self)) for accurate signature.

    __iter__()
        Return an iterator object on the models contained in the bank

    __len__()
        Returns the number of models stored in the bank
        Return type integer

    __weakref__
        list of weak references to the object (if defined)

    add_model(model)
        Parameters model (macsypy.model.Model object) – the model to add
        Raise KeyError if a model with the same name is already registered.
```

Model

class macsypy.model.**Model** (*fqn*, *inter_gene_max_space*, *min_mandatory_genes_required=None*,
min_genes_required=None, *max_nb_genes=None*, *multi_loci=False*)
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*)

Parameters *other* – the other model to compare

Returns True if this fully qualified name is equal to other fully qualified name. False otherwise.

Return type boolean

__gt__ (*other*)

Parameters *other* – the other model to compare

Returns True if this fully qualified name is greater than to other fully qualified name. False otherwise.

Return type boolean

__hash__ ()

Returns

__init__ (*fqn*, *inter_gene_max_space*, *min_mandatory_genes_required=None*,
min_genes_required=None, *max_nb_genes=None*, *multi_loci=False*)

Parameters

- **fqn** (*string*) – the fully qualified name of the model CRISPR-Cas/sub-typing/CAS-TypeIE
- **inter_gene_max_space** (*integer*) – the maximum distance between two genes (co-localization parameter)
- **min_mandatory_genes_required** (*integer*) – the quorum of mandatory genes to define this model
- **min_genes_required** (*integer*) – the quorum of genes to define this model
- **max_nb_genes** (*integer*) – 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** (*boolean*) –

Raises *ModelInconsistencyError* – if an error is found in model logic. For instance *genes_required > min_mandatory_genes_required*

__lt__ (*other*)

Parameters *other* – the other model to compare

Returns True if this fully qualified name is lesser than to other fully qualified name. False otherwise.

Return type boolean

__str__()

Return str(self).

__weakref__

list of weak references to the object (if defined)

property family_name

Returns the family name of the model for instance 'CRISPRCas' or 'TXSS'

Return type str

filter(hits)

filter out the hits according to this model. 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 embed in the ModelGene) only the hits related to genes implied in the model are kept.

Parameters **hits** (list of `macsypy.report.Hit` object) – list of hits to filter

Returns list of hits

Return type list of `macsypy.report.Hit` object

property genes

Returns all the genes without the exchangeables which are described in the model.

Return type set of `macsypy.gene.ModelGene` objects.

get_gene(gene_name)

Parameters **gene_name** (*string*) – the name of the gene to get

Returns the gene corresponding to gene_name.

Return type a `macsypy.gene.ModelGene` object.

Raise `KeyError` the model does not contain any gene with name gene_name.

property inter_gene_max_space

Returns set the maximum distance allowed between 2 genes for this model

Return type integer

property max_nb_genes

Returns the maximum number of genes to assess the model presence.

Return type int (or None)

property min_genes_required

Returns get the minimum number of genes to assess for the model presence.

Return type integer

property min_mandatory_genes_required

Returns get the quorum of mandatory genes required for this model

Return type integer

property multi_loci

Returns True if the model is authorized to be inferred from multiple loci, False otherwise

Return type boolean

property name

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 (*macsypy.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 *macsypy.gene.GeneBank* object. The `gene_bank` and `model_bank` (*macsypy.model.ModelBank* object) are instantiated in `macsypy.scripts.macsyfinder.main()` function and filled by a `definition_parser` (*macsypy.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 MacSyFinder run several instances of ModelGene and/or Exchangeable with the same name may coexists . But in A whole macsyfinder run there is only one instance `core_sctn_flg` and `core_sctn`.

GeneBank

class `macsypy.gene.GeneBank`

Store all Gene objects. Ensure that genes are instantiated only once.

__contains__ (*gene*)

Implement the membership test operator

Parameters *gene* (`macsypy.gene.CoreGene` object) – the gene to test

Returns True if the gene is in, False otherwise

Return type boolean

__getitem__ (*key*)

Parameters *key* (*tuple (string, string)*) – 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.

Return type `macsypy.gene.CoreGene` object

Raises **KeyError** – if the key does not exist in GeneBank.

__init__ ()

Initialize self. See help(type(self)) for accurate signature.

__iter__ ()

Return an iterator object on the genes contained in the bank

__weakref__

list of weak references to the object (if defined)

add_new_gene (*model_location, name, profile_factory*)

Create a gene and store it in the bank. If the same gene (same name) is add twice, it is created only the first time.

Parameters

- **model_location** (`macsypy.registry.ModelLocation` object) – the location where the model family can be found.
- **name** (*str*) – the name of the gene to add
- **profile_factory** (`profile.ProfileFactory` object.) – The Profile factory

genes_fqn ()

Returns the fully qualified name for all genes in the bank

Return type str

Gene

There is two classes to modelize a gene: `macsypy.gene.CoreGene` and `macsypy.gene.ModelGene`. The `CoreGene` are created using the `macsypy.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 different 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 `macsypy.hit.Hit` object are link to a `CoreGene`, whereas the `macsypy.hit.ValidHit` *ref_gene* attribute reference a `macsypy.gene.ModelGene`

CoreGene

class `macsypy.gene.CoreGene` (*model_location*, *name*, *profile_factory*)

Modelize gene attach to a profile. It can be only one instance with the the same name (family name, gene name)

__hash__ ()

Return hash(self).

__init__ (*model_location*, *name*, *profile_factory*)

Initialize self. See help(type(self)) for accurate signature.

__weakref__

list of weak references to the object (if defined)

property model_family_name

The name of the model family for instance 'CRISPRCas' or 'TXSS'

property name

The name of the gene a hmm profile with the same name must exists.

property profile

The HMM protein Profile corresponding to this gene `macsypy.profile.Profile` object

ModelGene

class `macsypy.gene.ModelGene` (*gene*, *model*, *loner=False*, *multi_system=False*, *inter_gene_max_space=None*)

Handle Gene describe in a Model

__hash__ ()

Return hash(self).

__init__ (*gene*, *model*, *loner=False*, *multi_system=False*, *inter_gene_max_space=None*)

Handle gene described in a Model

Parameters

- **gene** (a `macsypy.gene.CoreGene` object.) – a gene link to a profile
- **model** (`macsypy.model.Model` object.) – the model that owns this Gene
- **loner** (*boolean*.) – True if the Gene can be isolated on the genome (with no contiguous genes), False otherwise.
- **multi_system** (*boolean*.) – True if this Gene can belong to different occurrences of this System.
- **inter_gene_max_space** (*integer*) – the maximum space between this Gene and another gene of the System.

__str__ ()

Print the name of the gene and of its exchangeable genes.

__weakref__

list of weak references to the object (if defined)

add_exchangeable (*exchangeable*)

Add a exchangeable gene to this Gene

Parameters **exchangeable** (`macsypy.gene.Exchangeable` object) – the exchangeable to add

alternate_of()

Returns the gene to which this one is an exchangeable to (reference gene), or itself if it is a first level gene.

Return type *macsypy.gene.ModelGene* object

property exchangeables

Returns the list of genes which can replace this one without any effect on the model

Return type list of *macsypy.gene.ModelGene* objects

property inter_gene_max_space

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 the value set at the System level.

Return type integer.

is_accessory(model)

Returns True if the gene is within the *accessory* genes of the model, False otherwise.

Parameters model (*macsypy.model.Model* object.) – the query of the test

Return type boolean.

property is_exchangeable

Returns True if this gene is describe in the model as an exchangeable. False if ot is describe as first level gene.

is_forbidden(model)

Returns True if the gene is within the *forbidden* genes of the model, False otherwise.

Parameters model (*macsypy.model.Model* object.) – the query of the test

Return type boolean.

is_mandatory(model)

Returns True if the gene is within the *mandatory* genes of the model, False otherwise.

Parameters model (*macsypy.model.Model* object.) – the query of the test

Return type boolean.

property loner

Returns True if the gene can be isolated on the genome, False otherwise

Return type boolean

property model

Returns the Model that owns this Gene

Return type *macsypy.model.Model* object

property multi_system

Returns True if this Gene can belong to different occurrences of **the model** (and can be used for multiple System assessments), False otherwise.

Return type boolean.

Exchangeable

class `macsypy.gene.Exchangeable` (*c_gene*, *gene_ref*)

Handle Exchangeables. Exchangeable are ModelGene which can replaced functionally an other ModelGene. Biologically it can be Homolog or Analog

__init__ (*c_gene*, *gene_ref*)

Parameters

- **c_gene** (*macsypy.gene.CoreGene* object.) – the gene
- **gene_ref** (*macsypy.gene.ModelGene* object.) – the gene to which the current can replace it.

add_exchangeable (*exchangeable*)

This method should never be called, it's a security to avoid to add exchangeable to an exchangeable.

Parameters **exchangeable** (*macsypy.gene.Exchangeable*) –

Raises *MacsypyError* –

alternate_of ()

Returns the gene to which this one is an exchangeable to (reference gene)

Return type *macsypy.gene.ModelGene* object

property is_exchangeable

Returns True

GeneStatus

class `macsypy.gene.GeneStatus`

Handle status of Gene GeneStatus can take 4 value:

- MANDATORY
- ACCESSORY
- FORBIDDEN
- NEUTRAL

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.

ProfileFactory

class `macsypy.profile.ProfileFactory` (*cfg*)

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*)

Initialize self. See `help(type(self))` for accurate signature.

__weakref__

list of weak references to the object (if defined)

get_profile (*gene, model_location*)

Parameters

- **gene** (`macsypy.gene.Gene` or `macsypy.gene.Homolog` or `macsypy.gene.Analog` object) – the gene associated to this profile
- **model_location** (`macsypy.registries.ModelLocation` object.) – 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.

Return type `macsypy.profile.Profile` object

Profile

class `macsypy.profile.Profile` (*gene, cfg, path*)

Handle a HMM protein profile

__init__ (*gene, cfg, path*)

Parameters

- **gene** (`macsypy.secretion.Gene` object) – the gene corresponding to this profile
- **cfg** (`macsypy.config.Config` object) – the configuration
- **path** (*string*) – the path to the hmm profile.

__len__ ()

Returns the length of the HMM protein profile

Return type `int`

__str__ ()

Print the name of the corresponding gene and the path to the HMM profile.

__weakref__

list of weak references to the object (if defined)

_profile_features ()

Parse the HMM profile to extract the length and the presence of GA bit threshold

Returns the length, presence of ga bit threshold

Return type `tuple(int length, bool ga_threshold)`

execute ()

Launch the Hmmer search (`hmmsearch` executable) with this profile

Returns an object storing information on the results of the HMM search (`HMMReport`)

Return type `macsypy.report.HMMReport` object

hit

A Hit is created when *hmmsearch* find similarities between a profile and protein of the input dataset

hit

class `macsypy.hit.Hit` (*gene*, *hit_id*, *hit_seq_length*, *replicon_name*, *position_hit*, *i_eval*, *score*, *profile_coverage*, *sequence_coverage*, *begin_match*, *end_match*)
Handle the hits filtered from the Hmmer search. The hits are instantiated by `HMMReport.extract()`
method

`__eq__` (*other*)

Return True if two hits are totally equivalent, False otherwise.

Parameters *other* (`macsypy.report.Hit` object) – the hit to compare to the current object

Returns the result of the comparison

Return type boolean

`__gt__` (*other*)

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* (`macsypy.report.Hit` object) – the hit to compare to the current object

Returns True if self is > other, False otherwise

`__hash__` ()

To be hashable, it's needed to be put in a set or used as dict key

`__init__` (*gene*, *hit_id*, *hit_seq_length*, *replicon_name*, *position_hit*, *i_eval*, *score*, *profile_coverage*, *sequence_coverage*, *begin_match*, *end_match*)

Parameters

- **gene** (`macsypy.gene.CoreGene` object) – the gene corresponding to this profile
- **hit_id** (*str*) – the identifier of the hit
- **hit_seq_length** (*int*) – the length of the hit sequence
- **replicon_name** (*str*) – the name of the replicon
- **position_hit** (*int*) – the rank of the sequence matched in the input dataset file
- **i_eval** (*float*) – the best-domain evaluate (i-evalue, “independent evaluate”)
- **score** (*float*) – the score of the hit
- **profile_coverage** (*float*) – percentage of the profile that matches the hit sequence
- **sequence_coverage** (*float*) – percentage of the hit sequence that matches the profile
- **begin_match** (*int*) – where the hit with the profile starts in the sequence
- **end_match** (*int*) – where the hit with the profile ends in the sequence

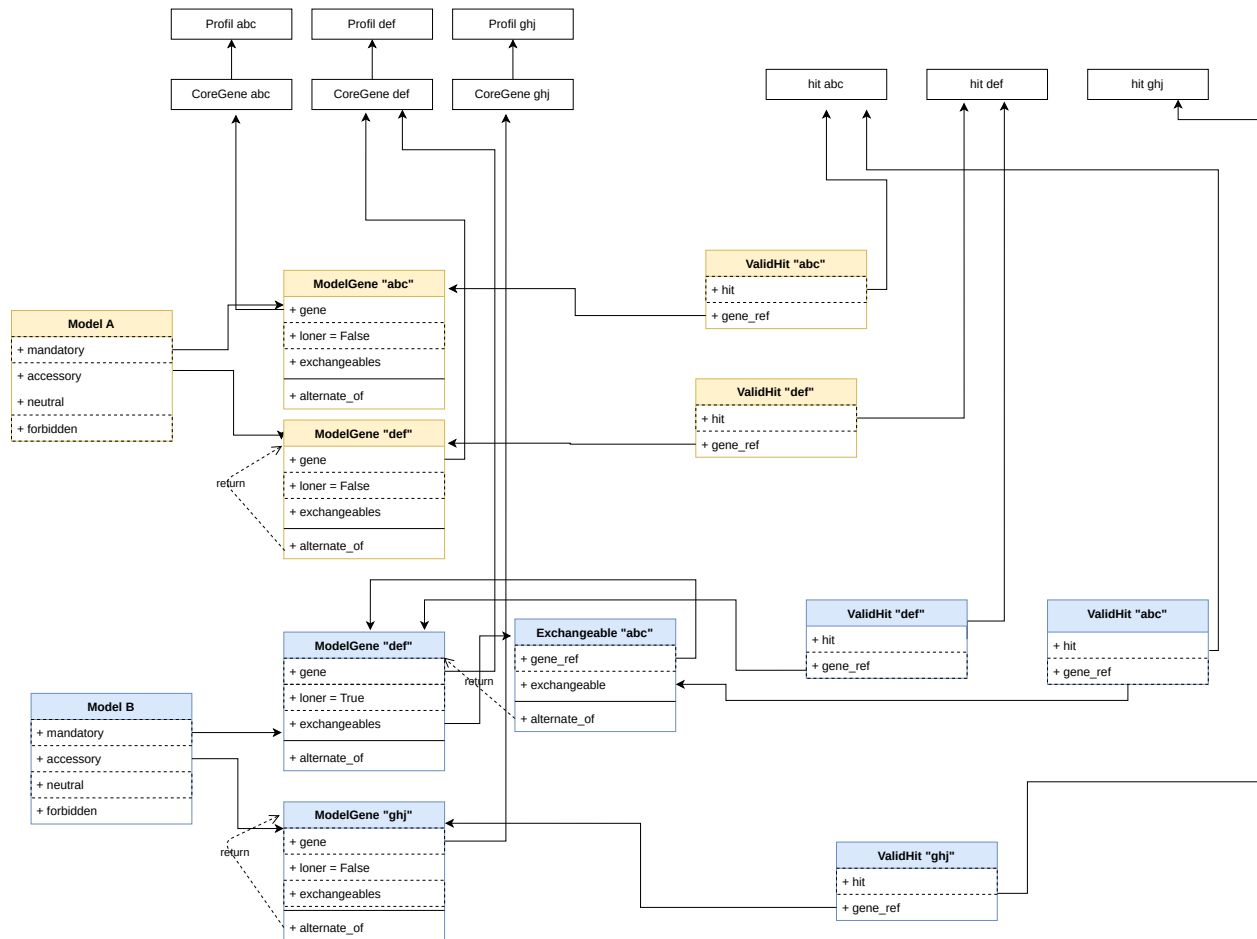


Fig. 2: A diagram showing the interaction between CoreGene, ModelGene, Model, Hit, ValidHit interactions. The diagram above represents the models, genes and hit generated from the definitions below.

```

<model name="A" inter_gene_max_space="2">
  <gene name="abc" presence="mandatory"/>
  <gene name="def" presence="accessory"/>
</model>

<model name="B" inter_gene_max_space="5">
  <gene name="def" presence="mandatory"/>
  <exchangeables>
    <gene name="abc"/>
  </exchangeables>
  <gene name="ghj" presence="accessory"/>
</model>

```

__lt__ (*other*)

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* (`macsypy.report.Hit` object) – the hit to compare to the current object

Returns True if self is < other, False otherwise

__str__ ()

Returns Useful information on the Hit: regarding Hmmer statistics, and sequence information

Return type str

__weakref__

list of weak references to the object (if defined)

get_position ()

Returns the position of the hit (rank in the input dataset file)

Return type integer

class `macsypy.hit.HitWeight` (*itself: float = 1, exchangeable: float = 0.8, mandatory: float = 1, accessory: float = 0.5, neutral: float = 0, loner_multi_system: float = 0.7*)

The weight to compute the cluster and system score see user documentation macsyfinder functioning for further details by default

- `itself = 1`
- `exchangeable = 0.8`
- `mandatory = 1`
- `accessory = 0.5`
- `neutral = 0`
- `loner_multi_system = 0.7`

__weakref__

list of weak references to the object (if defined)

class `macsypy.hit.ValidHit` (*hit, gene_ref, gene_status*)

Encapsulates a `macsypy.report.Hit` This class stores a Hit 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

__eq__ (*other*)

Return self==value.

__gt__ (*other*)

Return self>value.

__init__ (*hit, gene_ref, gene_status*)

Parameters

- **hit** (`macsypy.hit.Hit` object) – a match between a hmm profile and a replicon

- **gene_ref** (*macsypy.gene.ModelGene* object) – The ModelGene link to this hit
The ModelGene have the same name than the CoreGene But one hit can be link to several ModelGene (several Model) To know for what gene this hit play role use the *macsypy.gene.ModelGene.alternate_of()*

```
hit.gene_ref.alternate_of()
```

- **gene_status** (*macsypy.gene.GeneStatus* object) –

__lt__ (*other*)

Return self<value.

__weakref__

list of weak references to the object (if defined)

property loner

Returns True if the hit represent a *loner* *macsypy.Gene.ModelGene*, False otherwise.

property multi_system

Returns True if the hit represent a *multi_systems* *macsypy.Gene.ModelGene*, False otherwise.

macsypy.hit.get_best_hits (*hits*, *key*='score')

If several hits match the same protein, keep only the best match based either on

- score
- i_evalue
- profile_coverage

Parameters

- **hits** ([*macsypy.hit.Hit* 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 [*macsypy.hit.Hit* object, ...]

cluster

A cluster is an ordered set of hits related to a model which satisfy the model distance constraints.

cluster

class *macsypy.cluster.Cluster* (*hits*, *model*, *hit_weights*)

Handle hits relative to a model which collocates

__contains__ (*v_hit*)

Parameters *v_hit* (*macsypy.hit.ValidHit* object) – The hit to test

Returns True if the hit is in the cluster hits, False otherwise

__init__ (*hits*, *model*, *hit_weights*)

Parameters

- **hits** ([*macsypy.hit.Hit* | *macsypy.hit.ValidHit*, ...]) – the hits constituting this cluster
- **model** (*macsypy.model.Model*) – the model associated to this cluster

__str__ ()

Returns a string representation of this cluster

__weakref__

list of weak references to the object (if defined)

_check_replicon_consistency ()

Raise *MacsypyError* if all hits of a cluster are NOT related to the same replicon

fulfilled_function (*gene*)

Parameters **gene** (*macsypy.gene.Gene* object) – The gene which must be tested.

Returns True if the cluster contains one hit which fulfill the function corresponding to the gene (the gene himself or an exchangeable)

property loner

Returns True if this cluster is made of only one hit representing a loner gene

merge (*cluster*, *before=False*)

merge the cluster in this one. (do it in place)

Parameters

- **cluster** (*macsypy.cluster.Cluster* object) –
- **before** (*bool*) – If False the hits of the cluster will be add at the end of this one, Otherwise the cluster hits will be inserted before the hits of this one.

Returns None

Raises *MacsypyError* – if the two clusters have not the same model

macsypy.cluster.build_clusters (*hits*, *rep_info*, *model*, *hit_weights*)

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 *macsypy.hit.Hit* objects) – list of filtered hits
- **rep_info** (*macsypy.Indexes.RepliconInfo* object) – the replicon to analyse
- **model** (*macsypy.model.Model* object) – the model to study

Returns list of clusters

Return type List of *Cluster* objects

macsypy.cluster.filter_loners (*cluster*, *loners*)

filter loners to remove those which are already in the cluster

Parameters

- **cluster** (*macsypy.cluster.Cluster* object) – The cluster
- **loners** (*list of cluster [Cluster, ..]*) – the clusters constituted by one loner to filter

Returns list of loners which are not already in the cluster

Return type [*Cluster*, ..]

`macsypy.cluster.get_loners(hits, model, hit_weights)`

Create a list of Clusters each cluster is build with one hit matching a loner

Parameters

- **hits** – The list of hits to filter
- **model** (*macsypy.model.Model* object) – the model which will used to build the clusters

Returns The list of cluster which each element is build at least with one loner

Return type [*Cluster*, ..]

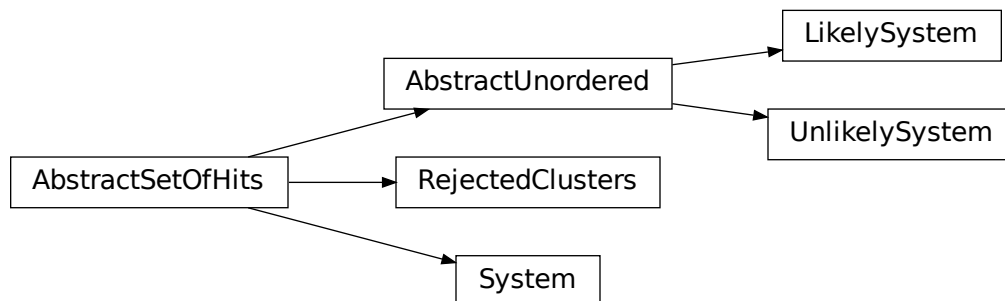
system

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 (*macsypy.hit.ValidHit*) and then check if these valid hit satisfy the quorum constraints defined in the model. According this it instantiate a *macsypy.system.System* or *macsypy.system.RejectedClusters* for ordered replicons or *macsypy.system.LikelySystem* or *macsypy.system.UnlikelySystem* for unordered replicons

below the inheritance diagram:



Warning: The abstract class *macsypy.system.AbstractSetOfHits* is controlled by the metaclass *macsypy.system.MetaSetOfHits* which inject on the fly several private attributes and public properties (see more in *macsypy.system.MetaSetOfHits* documentation)

System

class `macsypy.system.AbstractSetOfHits` (*model*, *replicon_name*)

Is the mother class of System, RejectedCluster, LikelySystems UnlikelySystem, ...

__init__ (*model*, *replicon_name*)

Initialize self. See help(type(self)) for accurate signature.

__weakref__

list of weak references to the object (if defined)

count ()

fill structures one for supported status mandatory, accessory, ... each structure count how many hit for each gene of the model mandatory_occ = { gene_name : [ValidHit, ...] :return: None

property position

Returns The position of the first and last hit, excluded the hit coding for loners. If the system is composed only by loners, used loners to compute position

Return type tuple (start: int, end:int)

property replicon_name

Returns The name of the replicon

Return type str

property wholeness

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 sepcified in model definition (xml) or on the command line

Return type float

class `macsypy.system.AbstractUnordered` (*model*, *mandatory_hits*, *accessory_hits*, *neutral_hits*, *forbidden_hits*)

Technical abstract class to factorize code share between LikelySystem and UnlikelySystem

__init__ (*model*, *mandatory_hits*, *accessory_hits*, *neutral_hits*, *forbidden_hits*)

Initialize self. See help(type(self)) for accurate signature.

property accessory_hits

Returns The list of accesory hits

Return type list of `macsypy.hit.ValidHit` objects

property allowed_hits

Returns The list of allowed (mandatory, accessory, neutral) hits

Return type list of `macsypy.hit.ValidHit` objects

property forbidden_hits

Returns The list of forbidden hits

Return type list of `macsypy.hit.ValidHit` objects

property hits

Returns The list of all hits sorted by their position

Return type list of `macsypy.hit.ValidHit` objects

property mandatory_hits
Returns The list of mandatory hits
Return type list of *macsypy.hit.ValidHit* objects

property neutral_hits
Returns The list of neutral hits
Return type list of *macsypy.hit.ValidHit* objects

class *macsypy.system.ClusterSystemTracker* (*systems*)
track in which system is implied each cluster

__init__ (*systems*)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

class *macsypy.system.HitSystemTracker* (*systems*)
track in which system is implied each hit

__init__ (*systems*)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

class *macsypy.system.LikelySystem* (*model, mandatory_hits, accessory_hits, neutral_hits, forbidden_hits*)
”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__ ()
Returns a string representation of this LikelySystem

class *macsypy.system.MatchMaker* (*model*)
Is an abstract class for (Ordered/Unordered)MatchMaker the *match* class method must be implemented in concrete classes

__init__ (*model*)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

_create_exchangeable_map (*genes*)
create a map between an exchangeable (formly homolog or analog) gene name and it’s gene reference

Parameters *genes* (list of *macsypy.gene.ModelGene* objects) – The genes to get the exchangeable genes

Return type a dict with keys are the exchangeable gene_name and the value the reference gene name

present_genes ()
Returns the lists of genes name in model which are present replicon (included exchangeable)

Return type 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*)

sort *macsypy.hit.Hit* according to the status of the gene the hit code for.

Parameters *hits* – list of *macsypy.hit.Hit* object

Returns the valid hits according to their status

Return type

a tuple of 4 lists

- *macsypy.hit.ValidHit* for MANDATORY genes
- *macsypy.hit.ValidHit* for ACCESSORY genes
- *macsypy.hit.ValidHit* for NEUTRAL genes
- *macsypy.hit.ValidHit* for FORBIDDEN genes

class *macsypy.system.MetaSetOfHits*

This metaclass controls 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 ()

Create a property which allows access to the gene corresponding to the cat of the model

Parameters *cat* (*str*) – the type of gene category to which we create the getter

Returns unbound method

class *macsypy.system.OrderedMatchMaker* (*model, redundancy_penalty*)

check if a set of hits match the quorum for ordered replicons (*ordered_replicon* or *gembase*)

__init__ (*model, redundancy_penalty*)

Initialize self. See *help(type(self))* for accurate signature.

match (*clusters*)

Check a set of clusters fill model constraints. If yes create a *macsypy.system.System* otherwise create a *macsypy.cluster.RejectedClusters*.

Parameters **clusters** (list of *macsypy.cluster.Cluster* objects) – The list of cluster to check if fit the model

Returns either a *System* or a *RejectedClusters*

Return type *macsypy.system.System* or *macsypy.cluster.RejectedClusters* object

class *macsypy.system.RejectedClusters* (*model, clusters, reasons*)

Handle a set of clusters which has been rejected during the *macsypy.system.match()* step This clusters (can be one) does not fill the requirements or contains forbidden genes.

__init__ (*model, clusters, reasons*)

Parameters

- **model** (*macsypy.model.Model* object) –
- **clusters** (list of *macsypy.cluster.Cluster* objects) – list of clusters. These Clusters should be created with *macsypy.cluster.Cluster* of *macsypy.hit.ValidHit* objects
- **reason** (*list of string*) – the reason why these clusters have been rejected

__str__ ()

Returns a string representation of this *RejectedCluster*

property hits

Returns The list of all hits that compose this system

Return type [*macsypy.hit.ValidHits*, ...]

class *macsypy.system.System* (*model, clusters, redundancy_penalty=1.5*)

Modelize as system. a system is an occurrence of a given model on a replicon.

__init__ (*model, clusters, redundancy_penalty=1.5*)

Parameters

- **model** (*macsypy.model.Model* object) – The model which has ben used to build this system
- **clusters** (list of *macsypy.cluster.Cluster* objects) – The list of cluster that form this system

property hits

Returns The list of all hits that compose this system

Return type [*macsypy.hit.ValidHits*, ...]

is_compatible (*other*)

Parameters **other** (*macsypy.system.System* object) – 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 *macsypy.hit.Hit* 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

Returns The number of loci of this system (loners are not considered)

Return type int ≥ 0

property loci_num

Returns the number of the corresponding locus for each cluster the cluster made of only one hit representing a loner is not considered as a loci so these clusters have the locus_num = 0

Return type list of int

property multi_loci

Returns True if the systems is encoded in multiple loci. False otherwise

Return type bool

occurrence ()

sometimes several systems collocates so they form only one cluster so macsyfinder 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

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

class `macsypy.system.UnlikelySystem` (*model, mandatory_hits, accessory_hits, neutral_hits, forbidden_hits, reasons*)

Handle components that not fill the quorum requirements defined in model.

__init__ (*model, mandatory_hits, accessory_hits, neutral_hits, forbidden_hits, reasons*)

Parameters

- **model** (*macsypy.model.Model* object) – The model which has ben used to build this system
- **allowed_hits** (list of *macsypy.hit.ValidHit* objects) – The list of hits that form this potential system this hits status must be MANDATORY, ACCESSORY or NEUTRAL
- **forbidden_hits** (list of *macsypy.hit.ValidHit* objects) – The list of hits that are forbidden
- **reasons** (*List of str*) – the reasons why this set of hits has been rejected

__str__ ()

Returns a string representation of this UnlikelySystem

property reasons

Returns The reasons why it probably not a system

Return type list of string

```
class macsypy.system.UnorderedMatchMaker(model)
```

```
    match (hits)
```

Parameters **hits** –

Returns

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 (`macsypy.HMMReport.Hit`) are built.

HMMReport API reference

```
class macsypy.report.HMMReport(gene, hmmer_output, cfg)
```

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, hmmer_output, cfg)
```

Parameters

- **gene** (`macsypy.gene.CoreGene` object) – the gene corresponding to the profile search reported here
- **hmmer_output** (*string*) – The path to the raw Hmmer output file
- **cfg** (`macsypy.config.Config` object) – the configuration object

```
    __str__ ()
```

Returns string representation of this report

Return type str

```
    __weakref__
```

list of weak references to the object (if defined)

```
    _build_my_db (hmm_output)
```

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

Return type dict

```
    _fill_my_db (macsyfinder_idx, db)
```

Fill the dictionary with information on the matched sequences

Parameters

- **macsyfinder_idx** (*string*) – the path the macsyfinder index corresponding to the dataset
- **db** (*dict*) – the database containing all sequence id of the hits.

abstract _get_replicon_name (*hit_id*)

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*)

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*, *gene_profile_lg*, *seq_lg*, *coverage_threshold*, *replicon_name*, *position_hit*,
i_value_sel, *b_grp*)

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** (*str*) – the sequence identifier
- **gene_profile_lg** (*int*) – the length of the profile matched
- **coverage_threshold** (*float*) – the minimal coverage of the profile to be reached in the Hmmer alignment for hit selection.
- **replicon_name** (*str*) – the identifier of the replicon
- **position_hit** (*int*) – the rank of the sequence matched in the input dataset file
- **i_value_sel** (*float*) – the maximal i-value (independent value) for hit selection
- **b_grp** (*list of list of strings*) – the Hmmer output lines to deal with (grouped by hit)

Paramint *seq_lg* the length of the sequence

Returns a sequence of hits

Return type list of `macsypy.report.Hit` objects

_parse_hmm_header (*h_grp*)

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 ()

Return the best hit among multiple hits

extract ()

Parse the output file of hmmer compute from an unordered genes base and produced a new synthetic report file.

save_extract ()

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 API reference

class `macsypy.report.GeneralHMMReport` (*gene, hmmer_output, cfg*)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to any type of ‘un-ordered’ datasets.

_get_replicon_name (*hit_id*)

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 `macsypy.report.OrderedHMMReport` (*gene, hmmer_output, cfg*)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to ‘ordered_replicon’ datasets.

_get_replicon_name (*hit_id*)

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 `macsypy.report.GembaseHMMReport` (*gene, hmmer_output, cfg*)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to ‘gembase’ format datasets.

_get_replicon_name (*hit_id*)

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

... **MacSyFinder - Detection of macromolecular systems in protein datasets** using systems modelling and similarity search. Authors: Sophie Abby, Bertrand Néron Copyright © 2014-2021 Institut Pasteur (Paris), and CNRS. See the COPYRIGHT file for details MacsyFinder is distributed under the terms of the GNU General Public License (GPLv3). See the COPYING file for details.

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

`macsypy.search_genes.search_genes` (*genes*, *cfg*)

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 Hit 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** (list of `macsypy.gene.CoreGene` objects) – the genes to search in the input sequence dataset
- **cfg** (`macsypy.config.Config` object) – the configuration object

solution

MacSyFinder 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

`macsypy.solution.find_best_solutions` (*systems*)

Among the systems choose the combination of systems which does not share `macsypy.hit.Hit` and maximize the sum of systems scores

Parameters **systems** (list of `macsypy.system.System` object) – the systems to analyse

Returns the list of list of systems which represent one best solution and the it’s score

Return type tuple of 2 elements the best solution and it’s score (`[[macsypy.system.System, ...], [macsypy.system.System, ...]]`, float score) The inner list represent a best solution

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 clusters

class `macsypy.serialization.SystemSerializer`

handle the different way to serialize a system

__weakref__

list of weak references to the object (if defined)

class `macsypy.serialization.TxtSystemSerializer`

Handle System serialization in text

serialize (*system*, *hit_system_tracker*)

Returns a string representation of system readable by human

class `macsypy.serialization.TsvSystemSerializer`

Handle System serialization in tsv format

serialize (*system*, *hit_system_tracker*)

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\tit_end_match
```

Return type str

class `macsypy.serialization.TsvSolutionSerializer`

Handle Solution (list of Systems) serialization in tsv format

`__init__()`

Constructor

`__weakref__`

list of weak references to the object (if defined)

serialize (*solution*, *sol_id*, *hit_system_tracker*)

Parameters

- **solution** (list of `macsypy.system.System` object) – the solution to serialize
- **hit_system_tracker** (`macsypy.system.HitSystemTracker` object) –

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 `macsypy.serialization.TsvSystemSerializer.serialize()` but with an extra column `sol_id` which is a technical id to identified the different solutions.

Database

The “database” object handles the indexes of the sequence dataset in fasta format, and other useful information on the input dataset.

MacSyFinder 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’`), MacSyFinder 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 (–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, MacSyFinder will use it. Otherwise, it will build it.

The user can force MacSyFinder to rebuild these indexes with the “–idx” option on the command-line.

database**class** `macsypy.database.Indexes` (*cfg*)

Handle the indexes for macsyfinder:

- find the indexes required by macsyfinder to compute some scores, or build them.

__init__ (*cfg*)

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* (`macsypy.config.Config` object) – the configuration

__iter__ ()

Raises `MacsyppyError` – if the indexes are not build

Returns an iterator on the indexes

To use it the index must be build.

__weakref__

list of weak references to the object (if defined)

_build_my_indexes (*index_dir*)

Build macsyfinder 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=False*)

search where to store(build=True) read indexes

Parameters *build* (*boolean*) – if check the index-dir permissions to write

Returns The directory where read or write the indexes

Return type str

Raises `ValueError` – if the directory specify by `-index-dir` option does not exists or if `build = True` index-dir is not writable

build (*force=False*)

Build the indexes from the sequence data set in fasta format,

Parameters *force* (*boolean*) – 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

Return type str

find_my_indexes ()

Returns the file of macsyfinder indexes if it exists in the dataset folder, None otherwise.

Return type string

class `macsypy.database.RepliconDB` (*cfg*)

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*)

Parameters **replicon_name** (*string*) – the name of the replicon

Returns True if replicon_name is in the repliconDB, false otherwise.

Return type boolean

__getitem__ (*replicon_name*)

Parameters **replicon_name** (*string*) – the name of the replicon to get information on

Returns the RepliconInfo for the provided replicon_name

Return type *RepliconInfo* object

Raise KeyError if replicon_name is not in repliconDB

__init__ (*cfg*)

Parameters **cfg** (*macsypy.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 (if defined)

_fill_gembase_min_max (*topology, default_topology*)

For each replicon_name of a gembase dataset, it fills the internal dictionary with a namedtuple Replicon-Info

Parameters

- **topology** (*dict*) – the topologies for each replicon (parsed from the file specified with the option -topology-file)
- **default_topology** (*string*) – the topology provided by the config.replicon_topology

_fill_ordered_min_max (*default_topology=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 ()

Fill the internal dictionary with min and max positions for each replicon_name of the sequence_db

get (*replicon_name, default=None*)

Parameters

- **replicon_name** (*string*) – the name of the replicon to get informations
- **default** (*any*) – 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.

Return type *RepliconInfo* object

items ()

Returns a copy of the RepliconDB as a list of (replicon_name, RepliconInfo) pairs

iteritems ()

Returns an iterator over the RepliconDB as a list (replicon_name, RepliconInfo) pairs

replicon_infos()

Returns a copy of the RepliconDB as list of replicons info

Return type RepliconInfo instance

replicon_names()

Returns a copy of the RepliconDB as a list of replicon_names

class macsypy.database.RepliconInfo(*topology, min, max, genes*)

__getnewargs__()

Return self as a plain tuple. Used by copy and pickle.

static __new__ (*_cls, topology, min, max, genes*)

Create new instance of RepliconInfo(topology, min, max, genes)

__repr__()

Return a nicely formatted representation string

_asdict()

Return a new OrderedDict which maps field names to their values.

classmethod __make (*iterable*)

Make a new RepliconInfo object from a sequence or iterable

_replace (***kwargs*)

Return a new RepliconInfo object replacing specified fields with new values

property genes

Alias for field number 3

property max

Alias for field number 2

property min

Alias for field number 1

property topology

Alias for field number 0

macsypy.database.fasta_iter(*fasta_file*)

Parameters **fasta_file** (*file object*) – 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)

Return type iterator

errors

The errors specific to macsyfinder and macsydata

error

exception `macsypy.error.MacsyDataLimitError`

Raised when the maximum number of github api call is reached

exception `macsypy.error.MacsydataError`

Raised when error is encounter during model package handling

exception `macsypy.error.MacsypyError`

The base class for MacSyFinder specific exceptions.

__weakref__

list of weak references to the object (if defined)

exception `macsypy.error.ModelInconsistencyError`

Raised when a definition model is not consistent.

exception `macsypy.error.OptionError`

Raised when command line option is not set properly

exception `macsypy.error.SystemDetectionError`

Raised when the detection of systems from Hits encountered a problem.

utils

Here some useful functions in the rest of macsyfinder code

utils API reference

`macsypy.utils.get_def_to_detect(models, model_registry)`

Parameters

- **models** (*list of tuple with the following structure:*
`[('model_1', ('def1', 'def2', ...)), ('model_2', ('def1', ...)), ...]`) – the list of models to detect as returned by `config.models`.
- **model_registry** (`macsypy.registries.ModelRegistry` object.) – the models registry for this run.

Returns the definitions to parse

Return type list of `macsypy.registries.DefinitionLocation` objects

Raises **ValueError** – if a model name provided in `models` is not in `model_registry`.

`macsypy.utils.get_replicon_names(genome_path)`

parse gembase file and the list of replicon identifiers

Parameters **genome_path** (*str*) – The path to a file containing sequence in **gembase** format

Returns the list of replicon identifiers

Return type list of str

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 *macsydata*, which is the entry point to manipulate models package.

ModelIndex

class `macsypy.package.AbstractModelIndex` (*cache: str = ""*)

This the base class for ModelIndex. This class cannot be implemented, it must be subclassed

__init__ (*cache: str = ""*)

static **__new__** (*cls, *args, **kwargs*)

Create and return a new object. See help(type) for accurate signature.

__weakref__

list of weak references to the object (if defined)

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

class `macsypy.package.LocalModelIndex` (*cache=None*)

It allow to manage installation from a local package (tarball)

__init__ (*cache=None*) → *None*

class `macsypy.package.RemoteModelIndex` (*org: str = 'macsy-models', cache=None*)

This class allow to interact with ModelIndex on github

__init__ (*org: str = 'macsy-models', cache=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*) → *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 exists
If dest is None, the macsyfinder 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 :return: The list of package names.

remote_exists () → bool

check if the remote exists and is an organization :return: True if the Remote url point to a github Organization, False otherwise

Package

class macsypy.package.**Package** (*path: str*)

This class Modelize a package of Models a package is a directory with the name of the models family it must contains 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 TODO

__init__ (*path: str*) → None

Parameters **path** (*str*) – The of the package root directory

__weakref__

list of weak references to the object (if defined)

_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_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()` → Optional[str]
find the README file

Returns The path to the README file or None if there is no file.

`_load_metadata()` → Dict

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`

Returns The parsed metadata as a dict

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

m

- `macsypy.cluster`, 97
- `macsypy.config`, 73
- `macsypy.database`, 110
- `macsypy.definition_parser`, 82
- `macsypy.error`, 113
- `macsypy.hit`, 94
- `macsypy.model_conf_parser`, 77
- `macsypy.package`, 114
- `macsypy.registries`, 78
- `macsypy.search_genes`, 108
- `macsypy.serialization`, 108
- `macsypy.solution`, 108
- `macsypy.system`, 100
- `macsypy.utils`, 113

Symbols

`__call__()` (*macsypy.system.MetaSetOfHits* method), 102
`__contains__()` (*macsypy.cluster.Cluster* method), 97
`__contains__()` (*macsypy.database.RepliconDB* method), 110
`__contains__()` (*macsypy.gene.GeneBank* method), 89
`__contains__()` (*macsypy.model.ModelBank* method), 85
`__eq__()` (*macsypy.hit.Hit* method), 94
`__eq__()` (*macsypy.hit.ValidHit* method), 96
`__eq__()` (*macsypy.model.Model* method), 86
`__eq__()` (*macsypy.registries.DefinitionLocation* method), 78
`__eq__()` (*macsypy.registries.ModelLocation* method), 79
`__getitem__()` (*macsypy.database.RepliconDB* method), 111
`__getitem__()` (*macsypy.gene.GeneBank* method), 89
`__getitem__()` (*macsypy.model.ModelBank* method), 85
`__getitem__()` (*macsypy.registries.ModelRegistry* method), 80
`__getnewargs__()` (*macsypy.database.RepliconInfo* method), 112
`__gt__()` (*macsypy.hit.Hit* method), 94
`__gt__()` (*macsypy.hit.ValidHit* method), 96
`__gt__()` (*macsypy.model.Model* method), 86
`__gt__()` (*macsypy.registries.DefinitionLocation* method), 78
`__gt__()` (*macsypy.registries.ModelLocation* method), 79
`__hash__()` (*macsypy.gene.CoreGene* method), 90
`__hash__()` (*macsypy.gene.ModelGene* method), 90
`__hash__()` (*macsypy.hit.Hit* method), 94
`__hash__()` (*macsypy.model.Model* method), 86
`__hash__()` (*macsypy.registries.DefinitionLocation* method), 78
`__init__()` (*macsypy.cluster.Cluster* method), 97
`__init__()` (*macsypy.config.Config* method), 73
`__init__()` (*macsypy.config.MacsyDefaults* method), 77
`__init__()` (*macsypy.database.Indexes* method), 110
`__init__()` (*macsypy.database.RepliconDB* method), 111
`__init__()` (*macsypy.definition_parser.DefinitionParser* method), 82
`__init__()` (*macsypy.gene.CoreGene* method), 90
`__init__()` (*macsypy.gene.Exchangeable* method), 92
`__init__()` (*macsypy.gene.GeneBank* method), 89
`__init__()` (*macsypy.gene.ModelGene* method), 90
`__init__()` (*macsypy.hit.Hit* method), 94
`__init__()` (*macsypy.hit.ValidHit* method), 96
`__init__()` (*macsypy.model.Model* method), 86
`__init__()` (*macsypy.model.ModelBank* method), 85
`__init__()` (*macsypy.model_conf_parser.ModelConfParser* method), 77
`__init__()` (*macsypy.package.AbstractModelIndex* method), 114
`__init__()` (*macsypy.package.LocalModelIndex* method), 114
`__init__()` (*macsypy.package.Package* method), 115
`__init__()` (*macsypy.package.RemoteModelIndex* method), 114
`__init__()` (*macsypy.profile.Profile* method), 93
`__init__()` (*macsypy.profile.ProfileFactory* method), 93
`__init__()` (*macsypy.registries.DefinitionLocation* method), 78
`__init__()` (*macsypy.registries.ModelLocation* method), 79
`__init__()` (*macsypy.registries.ModelRegistry* method), 80
`__init__()` (*macsypy.report.HMMReport* method), 105
`__init__()` (*macsypy.serialization.TsvSolutionSerializer* method), 109
`__init__()` (*macsypy.system.AbstractSetOfHits* method), 100
`__init__()` (*macsypy.system.AbstractUnordered* method), 100

```

__init__() (macsypy.system.ClusterSystemTracker
            method), 101
__init__() (macsypy.system.HitSystemTracker
            method), 101
__init__() (macsypy.system.MatchMaker method),
            101
__init__() (macsypy.system.OrderedMatchMaker
            method), 102
__init__() (macsypy.system.RejectedClusters
            method), 103
__init__() (macsypy.system.System method), 103
__init__() (macsypy.system.UnlikelySystem method),
            104
__iter__() (macsypy.database.Indexes method), 110
__iter__() (macsypy.gene.GeneBank method), 89
__iter__() (macsypy.model.ModelBank method), 85
__len__() (macsypy.model.ModelBank method), 85
__len__() (macsypy.profile.Profile method), 93
__lt__() (macsypy.hit.Hit method), 94
__lt__() (macsypy.hit.ValidHit method), 97
__lt__() (macsypy.model.Model method), 86
__lt__() (macsypy.registries.DefinitionLocation
            method), 78
__lt__() (macsypy.registries.ModelLocation method),
            79
__new__() (macsypy.database.RepliconInfo static
            method), 112
__new__() (macsypy.package.AbstractModelIndex
            static method), 114
__repr__() (macsypy.database.RepliconInfo method),
            112
__str__() (macsypy.cluster.Cluster method), 98
__str__() (macsypy.gene.ModelGene method), 90
__str__() (macsypy.hit.Hit method), 96
__str__() (macsypy.model.Model method), 86
__str__() (macsypy.profile.Profile method), 93
__str__() (macsypy.registries.DefinitionLocation
            method), 78
__str__() (macsypy.registries.ModelLocation
            method), 79
__str__() (macsypy.registries.ModelRegistry
            method), 80
__str__() (macsypy.report.HMMReport method), 105
__str__() (macsypy.system.LikelySystem method),
            101
__str__() (macsypy.system.RejectedClusters
            method), 103
__str__() (macsypy.system.UnlikelySystem method),
            104
__weakref__ (macsypy.cluster.Cluster attribute), 98
__weakref__ (macsypy.config.Config attribute), 73
__weakref__ (macsypy.config.MacsyDefaults at-
            tribute), 77
__weakref__ (macsypy.database.Indexes attribute),
            110
__weakref__ (macsypy.database.RepliconDB at-
            tribute), 111
__weakref__ (macsypy.definition_parser.DefinitionParser
            attribute), 83
__weakref__ (macsypy.error.MacsypyError attribute),
            113
__weakref__ (macsypy.gene.CoreGene attribute), 90
__weakref__ (macsypy.gene.GeneBank attribute), 89
__weakref__ (macsypy.gene.ModelGene attribute), 90
__weakref__ (macsypy.hit.Hit attribute), 96
__weakref__ (macsypy.hit.HitWeight attribute), 96
__weakref__ (macsypy.hit.ValidHit attribute), 97
__weakref__ (macsypy.model.Model attribute), 87
__weakref__ (macsypy.model.ModelBank attribute),
            85
__weakref__ (macsypy.model_conf_parser.ModelConfParser
            attribute), 77
__weakref__ (macsypy.package.AbstractModelIndex
            attribute), 114
__weakref__ (macsypy.package.Package attribute),
            115
__weakref__ (macsypy.profile.Profile attribute), 93
__weakref__ (macsypy.profile.ProfileFactory at-
            tribute), 93
__weakref__ (macsypy.registries.DefinitionLocation
            attribute), 78
__weakref__ (macsypy.registries.ModelLocation at-
            tribute), 79
__weakref__ (macsypy.registries.ModelRegistry at-
            tribute), 80
__weakref__ (macsypy.report.HMMReport attribute),
            105
__weakref__ (macsypy.serialization.SystemSerializer
            attribute), 108
__weakref__ (macsypy.serialization.TsvSolutionSerializer
            attribute), 109
__weakref__ (macsypy.system.AbstractSetOfHits at-
            tribute), 100
__weakref__ (macsypy.system.ClusterSystemTracker
            attribute), 101
__weakref__ (macsypy.system.HitSystemTracker at-
            tribute), 101
__weakref__ (macsypy.system.MatchMaker at-
            tribute), 101
_asdict() (macsypy.database.RepliconInfo method),
            112
_build_my_db() (macsypy.report.HMMReport
            method), 105
_build_my_indexes() (macsypy.database.Indexes
            method), 110
_check_metadata() (macsypy.package.Package
            method), 115
_check_model_conf() (macsypy.package.Package

```

method), 115
 _check_model_consistency() (macsypy.package.Package method), 115
 _check_replicon_consistency() (macsypy.cluster.Cluster method), 98
 _check_structure() (macsypy.package.Package method), 115
 _check_syntax() (macsypy.definition_parser.DefinitionParser method), 83
 _config_file_2_dict() (macsypy.config.Config method), 73
 _create_exchangeable_map() (macsypy.system.MatchMaker method), 101
 _create_model() (macsypy.definition_parser.DefinitionParser method), 83
 _fill_gembase_min_max() (macsypy.database.RepliconDB method), 111
 _fill_gene_bank() (macsypy.definition_parser.DefinitionParser method), 83
 _fill_my_db() (macsypy.report.HMMReport method), 105
 _fill_ordered_min_max() (macsypy.database.RepliconDB method), 111
 _fill_topology() (macsypy.database.RepliconDB method), 111
 _find_readme() (macsypy.package.Package method), 115
 _get_model_conf_node() (macsypy.model_conf_parser.ModelConfParser method), 77
 _get_model_node() (macsypy.definition_parser.DefinitionParser method), 83
 _get_replicon_name() (macsypy.report.GembaseHMMReport method), 107
 _get_replicon_name() (macsypy.report.GeneralHMMReport method), 107
 _get_replicon_name() (macsypy.report.HMMReport method), 105
 _get_replicon_name() (macsypy.report.OrderedHMMReport method), 107
 _hit_start() (macsypy.report.HMMReport method), 106
 _index_dir() (macsypy.database.Indexes method), 110
 _load_metadata() (macsypy.package.Package method), 116
 _make() (macsypy.database.RepliconInfo class method), 112
 _parse_exchangeable() (macsypy.definition_parser.DefinitionParser method), 83
 _parse_genes() (macsypy.definition_parser.DefinitionParser method), 84
 _parse_hmm_body() (macsypy.report.HMMReport method), 106
 _parse_hmm_header() (macsypy.report.HMMReport method), 106
 _parse_section() (macsypy.model_conf_parser.ModelConfParser method), 77
 _profile_features() (macsypy.profile.Profile method), 93
 _replace() (macsypy.database.RepliconInfo method), 112
 _scan_definitions() (macsypy.registries.ModelLocation method), 79
 _scan_profiles() (macsypy.registries.ModelLocation method), 79
 _set_command_line_config() (macsypy.config.Config method), 74
 _set_db_type() (macsypy.config.Config method), 74
 _set_default_config() (macsypy.config.Config method), 74
 _set_inter_gene_max_space() (macsypy.config.Config method), 74
 _set_max_nb_genes() (macsypy.config.Config method), 74
 _set_min_genes_required() (macsypy.config.Config method), 74
 _set_min_mandatory_genes_required() (macsypy.config.Config method), 74
 _set_model_config() (macsypy.config.Config method), 74
 _set_models() (macsypy.config.Config method), 74
 _set_models_dir() (macsypy.config.Config method), 75
 _set_multi_loci() (macsypy.config.Config method), 75
 _set_options() (macsypy.config.Config method), 75
 _set_previous_run_config() (macsypy.config.Config method), 75
 _set_project_config_file() (macsypy.config.Config method), 75
 _set_replicon_topology() (macsypy.config.Config method), 75
 _set_sequence_db() (macsypy.config.Config method), 75
 _set_system_models_dir() (macsypy.config.Config method), 75
 _set_system_wide_config() (macsypy.config.Config method), 75

syp.py.config.Config method), 75
 _set_topology_file() (*macsypy.config.Config*
method), 75
 _set_user_config_file() (*mac-*
syp.py.config.Config method), 75
 _set_user_wide_config() (*mac-*
syp.py.config.Config method), 75
 _str_2_tuple() (*macsypy.config.Config* method), 75
 _url_json() (*macsypy.package.RemoteModelIndex*
method), 114

A

AbstractModelIndex (class in *macsypy.package*),
 114
AbstractSetOfHits (class in *macsypy.system*), 100
AbstractUnordered (class in *macsypy.system*), 100
 accessory_hits() (*mac-*
syp.py.system.AbstractUnordered property),
 100
 add() (*macsypy.registries.ModelRegistry* method), 80
 add_exchangeable() (*macsypy.gene.Exchangeable*
method), 92
 add_exchangeable() (*macsypy.gene.ModelGene*
method), 90
 add_model() (*macsypy.model.ModelBank* method),
 85
 add_new_gene() (*macsypy.gene.GeneBank* method),
 89
 add_subdefinition() (*mac-*
syp.py.registries.DefinitionLocation method),
 78
 allowed_hits() (*mac-*
syp.py.system.AbstractUnordered property),
 100
 alternate_of() (*macsypy.gene.Exchangeable*
method), 92
 alternate_of() (*macsypy.gene.ModelGene*
method), 90

B

best_hit() (*macsypy.report.HMMReport* method),
 106
 build() (*macsypy.database.Indexes* method), 110
 build_clusters() (in module *macsypy.cluster*), 98

C

check() (*macsypy.package.Package* method), 116
 check_consistency() (*mac-*
syp.py.definition_parser.DefinitionParser
method), 84
 CITATION.yml, 69
 Cluster, 69
 Cluster (class in *macsypy.cluster*), 97

ClusterSystemTracker (class in *macsypy.system*),
 101
Config (class in *macsypy.config*), 73
 CONTRIBUTING, 70
 CONTRIBUTORS, 69
 COPYING, 70
 COPYRIGHT, 70
CoreGene (class in *macsypy.gene*), 90
 count() (*macsypy.system.AbstractSetOfHits* method),
 100

D

DefinitionLocation (class in *macsypy.registries*),
 78
DefinitionParser (class in *mac-*
syp.py.definition_parser), 82
 doc, 69
 download() (*macsypy.package.RemoteModelIndex*
method), 114

E

etc, 69
Exchangeable (class in *macsypy.gene*), 92
 exchangeables() (*macsypy.gene.ModelGene* prop-
 erty), 91
 execute() (*macsypy.profile.Profile* method), 93
 extract() (*macsypy.report.HMMReport* method), 106

F

family_name() (*macsypy.model.Model* property), 87
 fasta_iter() (in module *macsypy.database*), 112
 filter() (*macsypy.model.Model* method), 87
 filter_loners() (in module *macsypy.cluster*), 98
 find_best_solutions() (in module *mac-*
syp.py.solution), 108
 find_my_indexes() (*macsypy.database.Indexes*
method), 110
 forbidden_hits() (*mac-*
syp.py.system.AbstractUnordered property),
 100
 fulfilled_function() (*macsypy.cluster.Cluster*
method), 98

G

GembaseHMMReport (class in *macsypy.report*), 107
GeneBank (class in *macsypy.gene*), 89
GeneralHMMReport (class in *macsypy.report*), 107
 genes() (*macsypy.database.RepliconInfo* property),
 112
 genes() (*macsypy.model.Model* property), 87
 genes_fn() (*macsypy.gene.GeneBank* method), 89
GeneStatus (class in *macsypy.gene*), 92
 get() (*macsypy.database.RepliconDB* method), 111

- get_all_definitions() (macsypy.registries.ModelLocation method), 79
 get_best_hits() (in module macsypy.hit), 97
 get_def_to_detect() (in module macsypy.utils), 113
 get_definition() (macsypy.registries.ModelLocation method), 80
 get_definitions() (macsypy.registries.ModelLocation method), 80
 get_gene() (macsypy.model.Model method), 87
 get_loners() (in module macsypy.cluster), 99
 get_metadata() (macsypy.package.RemoteModelIndex method), 114
 get_position() (macsypy.hit.Hit method), 96
 get_profile() (macsypy.profile.ProfileFactory method), 93
 get_profile() (macsypy.registries.ModelLocation method), 80
 get_profiles_names() (macsypy.registries.ModelLocation method), 80
 get_replicon_names() (in module macsypy.utils), 113
 getter_maker() (macsypy.system.MetaSetOfHits method), 102
- ## H
- help() (macsypy.package.Package method), 116
 Hit (class in macsypy.hit), 94
 hit_weights() (macsypy.config.Config method), 76
 hits() (macsypy.system.AbstractUnordered property), 100
 hits() (macsypy.system.RejectedClusters property), 103
 hits() (macsypy.system.System property), 103
 HitSystemTracker (class in macsypy.system), 101
 HitWeight (class in macsypy.hit), 96
 hmmer_dir() (macsypy.config.Config method), 76
 HMMReport (class in macsypy.report), 105
- ## I
- Indexes (class in macsypy.database), 110
 info() (macsypy.package.Package method), 116
 inter_gene_max_space() (macsypy.config.Config method), 76
 inter_gene_max_space() (macsypy.gene.ModelGene property), 91
 inter_gene_max_space() (macsypy.model.Model property), 87
 is_accessory() (macsypy.gene.ModelGene method), 91
 is_compatible() (macsypy.system.System method), 103
 is_exchangeable() (macsypy.gene.Exchangeable property), 92
 is_exchangeable() (macsypy.gene.ModelGene property), 91
 is_forbidden() (macsypy.gene.ModelGene method), 91
 is_mandatory() (macsypy.gene.ModelGene method), 91
 items() (macsypy.database.RepliconDB method), 111
 iteritems() (macsypy.database.RepliconDB method), 111
- ## J
- join_def_path() (in module macsypy.registries), 81
- ## L
- LikelySystem (class in macsypy.system), 101
 list_package_vers() (macsypy.package.RemoteModelIndex method), 115
 list_packages() (macsypy.package.RemoteModelIndex method), 115
 LocalModelIndex (class in macsypy.package), 114
 loci_nb() (macsypy.system.System property), 104
 loci_num() (macsypy.system.System property), 104
 log_level() (macsypy.config.Config method), 76
 loner() (macsypy.cluster.Cluster property), 98
 loner() (macsypy.gene.ModelGene property), 91
 loner() (macsypy.hit.ValidHit property), 97
- ## M
- MacsydataError, 113
 MacsyDataLimitError, 113
 MacsyDefaults (class in macsypy.config), 77
 macsypy, 69
 macsypy.cluster (module), 97
 macsypy.config (module), 73
 macsypy.database (module), 110
 macsypy.definition_parser (module), 82
 macsypy.error (module), 113
 macsypy.hit (module), 94
 macsypy.model_conf_parser (module), 77
 macsypy.package (module), 114
 macsypy.registries (module), 78
 macsypy.search_genes (module), 108
 macsypy.serialization (module), 108
 macsypy.solution (module), 108
 macsypy.system (module), 100
 macsypy.utils (module), 113
 MacsypyError, 113
 mandatory_hits() (macsypy.system.AbstractUnordered property), 100

`match()` (*macsypy.system.OrderedMatchMaker method*), 102
`match()` (*macsypy.system.UnorderedMatchMaker method*), 105
`MatchMaker` (*class in macsypy.system*), 101
`max()` (*macsypy.database.RepliconInfo property*), 112
`max_nb_genes()` (*macsypy.config.Config method*), 76
`max_nb_genes()` (*macsypy.model.Model property*), 87
`merge()` (*macsypy.cluster.Cluster method*), 98
`metadata()` (*macsypy.package.Package property*), 116
`MetaSetOfHits` (*class in macsypy.system*), 102
`min()` (*macsypy.database.RepliconInfo property*), 112
`min_genes_required()` (*macsypy.config.Config method*), 76
`min_genes_required()` (*macsypy.model.Model property*), 87
`min_mandatory_genes_required()` (*macsypy.config.Config method*), 76
`min_mandatory_genes_required()` (*macsypy.model.Model property*), 87
`Model`, 69
`Model` (*class in macsypy.model*), 86
`Model family`, 69
`model()` (*macsypy.gene.ModelGene property*), 91
`model_family_name()` (*macsypy.gene.CoreGene property*), 90
`ModelBank` (*class in macsypy.model*), 85
`ModelConfParser` (*class in macsypy.model_conf_parser*), 77
`ModelDefinition`, 69
`ModelGene` (*class in macsypy.gene*), 90
`ModelInconsistencyError`, 113
`ModelLocation` (*class in macsypy.registries*), 79
`ModelRegistry` (*class in macsypy.registries*), 80
`models()` (*macsypy.registries.ModelRegistry method*), 81
`models_dir()` (*macsypy.config.Config method*), 76
`multi_loci()` (*macsypy.config.Config method*), 76
`multi_loci()` (*macsypy.model.Model property*), 87
`multi_loci()` (*macsypy.system.System property*), 104
`multi_system()` (*macsypy.gene.ModelGene property*), 91
`multi_system()` (*macsypy.hit.ValidHit property*), 97

N

`name()` (*macsypy.gene.CoreGene property*), 90
`name()` (*macsypy.model.Model property*), 87
`neutral_hits()` (*macsypy.system.AbstractUnordered property*), 101

O

`occurrence()` (*macsypy.system.System method*), 104

`OptionError`, 113
`OrderedHMMReport` (*class in macsypy.report*), 107
`OrderedMatchMaker` (*class in macsypy.system*), 102
`out_dir()` (*macsypy.config.Config method*), 76

P

`Package` (*class in macsypy.package*), 115
`parse()` (*macsypy.definition_parser.DefinitionParser method*), 84
`parse()` (*macsypy.model_conf_parser.ModelConfParser method*), 77
`parse_filtering()` (*macsypy.model_conf_parser.ModelConfParser method*), 78
`parse_weights()` (*macsypy.model_conf_parser.ModelConfParser method*), 78
`position()` (*macsypy.system.AbstractSetOfHits property*), 100
`present_genes()` (*macsypy.system.MatchMaker method*), 101
`Profile` (*class in macsypy.profile*), 93
`profile()` (*macsypy.gene.CoreGene property*), 90
`ProfileFactory` (*class in macsypy.profile*), 93

R

`README.md`, 70
`reasons()` (*macsypy.system.UnlikelySystem property*), 104
`RejectedClusters` (*class in macsypy.system*), 103
`remote_exists()` (*macsypy.package.RemoteModelIndex method*), 115
`RemoteModelIndex` (*class in macsypy.package*), 114
`replicon_infos()` (*macsypy.database.RepliconDB method*), 112
`replicon_name()` (*macsypy.system.AbstractSetOfHits property*), 100
`replicon_names()` (*macsypy.database.RepliconDB method*), 112
`RepliconDB` (*class in macsypy.database*), 110
`RepliconInfo` (*class in macsypy.database*), 112
`requirements.txt`, 70
`requirements_dev.txt`, 70

S

`save()` (*macsypy.config.Config method*), 77
`save_extract()` (*macsypy.report.HMMReport method*), 106
`scan_models_dir()` (*in module macsypy.registries*), 81
`score()` (*macsypy.system.System property*), 104

[search_genes\(\)](#) (in module *macsypy.search_genes*),
[108](#)
[serialize\(\)](#) (*macsypy.serialization.TsvSolutionSerializer*
method), [109](#)
[serialize\(\)](#) (*macsypy.serialization.TsvSystemSerializer*
method), [108](#)
[serialize\(\)](#) (*macsypy.serialization.TxtSystemSerializer*
method), [108](#)
[setup.py](#), [70](#)
[Solution](#), [69](#)
[sort_hits_by_status\(\)](#) (*mac-*
sypy.system.MatchMaker *method*), [101](#)
[split_def_name\(\)](#) (in module *macsypy.registries*),
[81](#)
[System](#), [69](#)
[System](#) (class in *macsypy.system*), [103](#)
[SystemDetectionError](#), [113](#)
[SystemSerializer](#) (class in *macsypy.serialization*),
[108](#)

T

[tests](#), [69](#)
[topology\(\)](#) (*macsypy.database.RepliconInfo* *prop-*
erty), [112](#)
[TsvSolutionSerializer](#) (class in *mac-*
sypy.serialization), [109](#)
[TsvSystemSerializer](#) (class in *mac-*
sypy.serialization), [108](#)
[TxtSystemSerializer](#) (class in *mac-*
sypy.serialization), [108](#)

U

[unarchive_package\(\)](#) (*mac-*
sypy.package.AbstractModelIndex *method*),
[114](#)
[UnlikelySystem](#) (class in *macsypy.system*), [104](#)
[UnorderedMatchMaker](#) (class in *macsypy.system*),
[104](#)
[utils](#), [69](#)

V

[ValidHit](#) (class in *macsypy.hit*), [96](#)

W

[wholeness\(\)](#) (*macsypy.system.AbstractSetOfHits*
property), [100](#)
[working_dir\(\)](#) (*macsypy.config.Config* *method*), [77](#)