MacSyFinder

Release 2.0

Sophie Abby, Bertrand Néron

Aug 05, 2024

CONTENTS

Note: A **new version of MacSyFinder (v2)** is available, see *here for an overview of the novelties*. The search engine was changed, 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*.

The search engine of v2 being much different from that of v1, we **strongly suggest** to test whether the results are relevant by simply "translating" the models from v1 to v2, or if the models need to be adapted to correctly function with 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**,
- they 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:

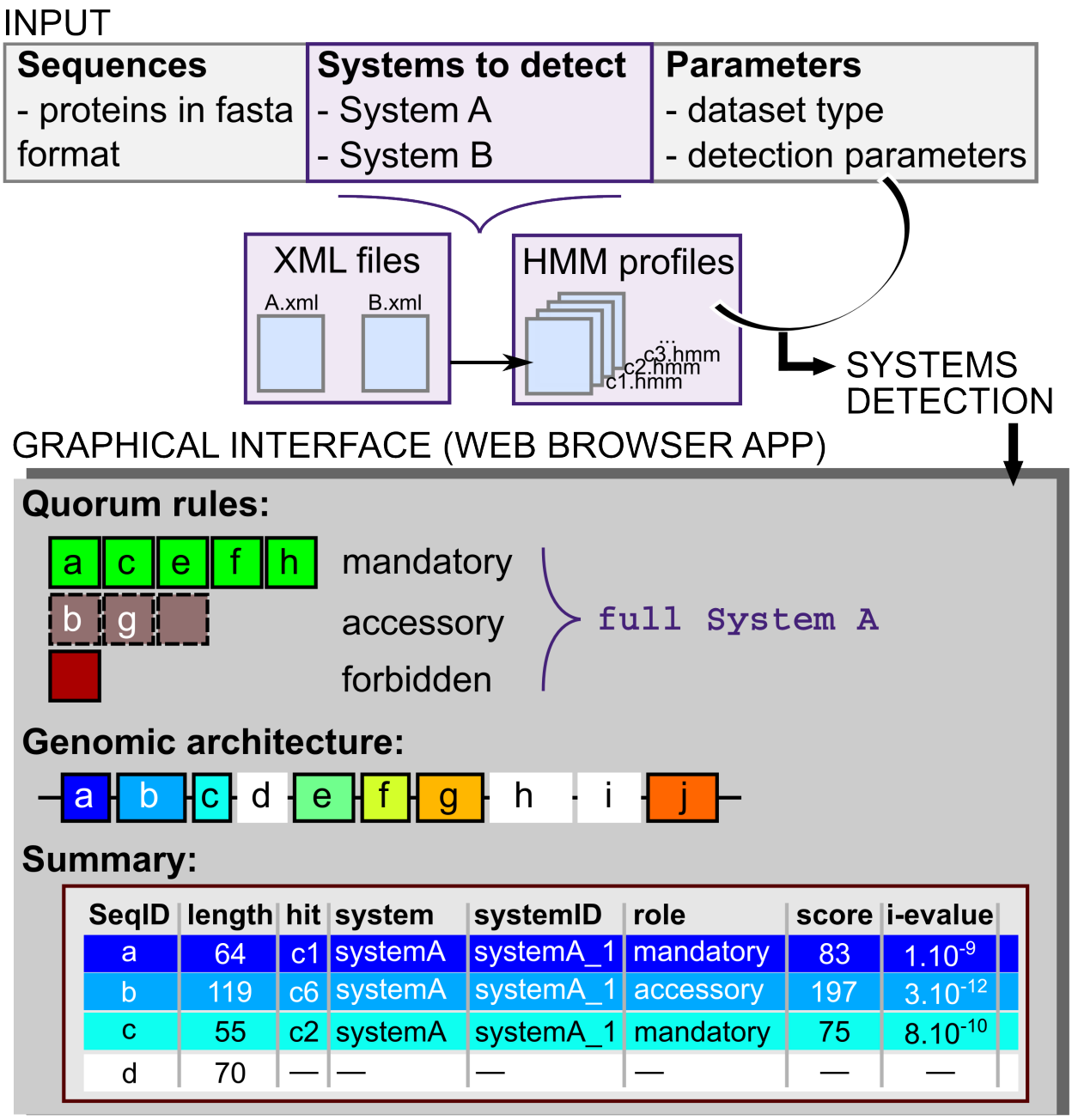
- builds or gather from databanks **HMM protein profiles** for components of interest,
- defines **decision rules** for each system in a dedicated XML grammar (see *Macromolecular models*).

Note: If you use MacSyFinder v2, please cite:

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

If you use MacSyFinder v1, 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



SeqID	length	hit	system	systemID	role	score	i-evalue
a	64	c1	systemA	systemA_1	mandatory	83	1.10^{-9}
b	119	c6	systemA	systemA_1	accessory	197	3.10^{-12}
c	55	c2	systemA	systemA_1	mandatory	75	8.10^{-10}
d	70	—	—	—	—	—	—

CHAPTER

ONE

USER GUIDE

1.1 User Guide

1.1.1 Running MacSyFinder

What's new in MacSyFinder v2?

V 2.1.4

For users

Minor bugs

- when profile name ends with *hmm*, then the profile was not retrieved by msf (https://github.com/gem-pasteur/macsyfinder/issues/69).
- fix omitted parameter *timeout* in *macsyfinder* step in parallel_macsyfinder

For modelers

- The *vers* is now deprecated in model/metadata file. *macsydata* rely only on the tag version to get the version of the model.
- *macsydata* now create a *git* repository for you and add files to it.
- a pre-push git hook is installed by *macsydata init* to prevent to push a tag (publish) a buggy nodel.

For developers

- all the *msf* code (macsypy and tests) pass the ruff linter
- all the code use type hints (required python >3.10) see (https://github.com/gem-pasteur/macsyfinder/blob/master/CONTRIBUTING.md)

V 2.1.3

Features

Support of *gziped* files

MSF can also read *.gz* compressed files both for hmm profiles and sequences in fasta. It will uncompress them on the fly. The compressed files must end with the *.gz* extension. For the *hmmsearch* step You need to have *gunzip* installed on your system for this to work.

search engine

A group of hits that respect the distance constraints but each hit represent the same gene on the model, is **not** considered as a *cluster*.

A group of hits that respect the distance constraints but all hits represent a Neutral gene in model, is **not** considered as a *cluster*.

timeout improvement

If a replicon is skipped due to timeout during best_solution phase. The results corresponding to this replicon are not produced, but a warning indicating that msf skip this replicon appear in outputs.

V 2.1.1

Update MSF citation, fix minor bugs and add add few features

New features

–force option

Force MSF run even the out dir already exists and is not empty. Use this option with caution, MSF will erase everything in out dir before to run. https://github.com/gem-pasteur/macsyfinder/issues/61

Minor bugs

Macsyfinder with python subprocess kill main process on error

If an error occurred during HMM phase, all processes were killed as well the mother process but MSF stoped with an ugly traceback. https://github.com/gem-pasteur/macsyfinder/issues/60

In Gembase format parsing

The genes were not well grouped by contigs for draft genomes.

Cannot join current thread error during unit tests phase

Sometimes the testsuite failed with the following error: *"cannot join current thread"* https://github.com/gem-pasteur/macsyfinder/issues/58

V 2.1

Bug fix

Security patch

Patch macsydata to fix CVE-2007-4559 https://github.com/gem-pasteur/macsyfinder/pull/57

New features

Squash cluster of loners

If a cluster is made up with only loners, then the hits are treated by MSF as loners and not as regular cluster.

New option –timeout

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

2.0

For Version 2, MacSyFinder was carried under Python 3.

New features and search engine

MacSyFinder v2 is a major release. 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*.

> **Warning:** The search engine being different, one might want to check that models carried from v1 to v2 have the expected behaviour.

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.
- a **new component feature** was introduced: "multi-model", that corresponds to components that are allowed to participate in occurrences of systems from different models. *See here* for more.
- 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 readibility. 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 spcific 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*.

Models installation with macsydata provide also some options to help the modeller as

- **macsydata init** to init a new model package.
- **macsydata check** to check the integrity of a model package, before to use/publish it.

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. We also provide container so you can use macsyfinder directly.

MacSyFinder dependencies

Python version >=3.10 is required to run MacSyFinder: https://docs.python.org/3.10/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 six Python library dependencies:

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

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

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

Note: If you want to contribute to the *MacSyFinder* code, check the guide lines (CONTRIBUTING) and specific procedure for *developer installation*.

MacSyFinder Installation procedure

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

Archive overview

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

Installation steps:

Make sure every required dependency/software is present.

By default 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.

```
python3 -m pip install 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

```
python3 -m pip install --user macsyfinder
```

installation in a virtualenv

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

To delete it, remove the directory

```
rm -R macsyfinder
```

From Conda/Mamba

From version 2.0, MacSyFinder is packaged for Conda/Mamba

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

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

From container

With Docker

The docker image is available on Docker Hub (https://hub.docker.com/repository/docker/gempasteur/macsyfinder) The computations are performed under msf user in /home/msf inside the container. So You have to mount a directory from the host in the container to exchange data (inputs data, and results) from the host and the container. The shared directory must be writable by the *msf* user or overwrite the user in the container by your id (see example below)

Furthermore the models are no longer packaged along macsyfinder. So you have to install them by yourself. For that we provide a command line tool macsydata which is inspired by pip.

```
macsydata search PACKNAME
macsydata install PACKNAME== or >=, or ... VERSION
```

To work with Docker you have to install models in a directory which will be mounted in the image at run time

```
mkdir shared_dir
cd shared_dir
```

install desired models in my_models directory

```
docker run -v ${PWD}/:/home/msf -u $(id -u ${USER}):$(id -g ${USER})  gempasteur/
↪macsyfinder:<tag> macsydata install --target /home/msf/my_models <MODELS_PACK>
```

run msf against all models contains in <MODELS_PACK>

```
docker run -v ${PWD}/:/home/msf -u $(id -u ${USER}):$(id -g ${USER})  gempasteur/
↪macsyfinder:<tag> macsyfinder --db-type unordered_replicon --models-dir=/home/msf/my_
↪models/ --models  <MODELS_PACK>  all --sequence-db my_genome.fasta -w 12
```

With Apptainer (formely Singularity)

As the docker image is registered in docker hub you can also use it directly with Apptainer (https://apptainer.org/). Unlike docker you have not to worry about shared directory, your HOME and /tmp are automatically shared.

```
# install desired models in my_models directory
apptainer run -H ${HOME} docker://gempasteur/macsyfinder:<tag> macsydata install --
↪target my_models <MODELS_PACK>

# run msf against all models contains in <MODELS_PACK>
apptainer run -H ${HOME} docker://gempasteur/macsyfinder:<tag> macsyfinder --db-type␣
↪unordered_replicon --models-dir=my_models --models <MODELS_PACK> all --sequence-db my_
↪genome.fasta -w 12
```

If you intend to run *apptainer* from host which cannot access internet (cluster node for instance), you have to

1. download the image locally
2. transfert the image file on the right file system
3. and then use it.

```
apptainer build msf-<tag>.simg docker://gempasteur/macsyfinder:<tag>
cp msf-<tag>.simg <cluster_file_system>
apptainer run -H ${HOME} msf-<tag>.simg macsyfinder --db-type unordered_replicon --
↪models-dir=my_models --models <MODELS_PACK> all --sequence-db my_genome.fasta -w 12
```

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 (https://github.com/macsy-models) or locally installed. The general syntax for *macsydata* is:

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

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

```
macsydata available
```

To search for models on *macsy-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

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

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

To install it in any directory:

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

To know how to cite a model package:

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

To show the model definition:

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

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

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

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

```
macsydata definition TXSS+/bacterial
```

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

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

above macsydata with required options. Below I add optioanl but recommended options.

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

To list all *macsydata* subcommands:

```
macsydata --help
```

To list all available options for a subcommand:

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

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

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

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 a FASTA file (for further details see *Input dataset*). In the best case scenario, they would be ordered as the corresponding genes are ordered along the replicons.
3. You need to install, or make available to MacSyFinder the models to search in your input genome data. Please refer to *Macromolecular models* to create your own package of models. Otherwise, macsy-models contributed by the community are available here: https://github.com/macsy-models and can be retrieved and installed using the *macsydata* command, installed as part of the MacSyFinder suite.
4. Command lines:
   - Type: `macsyfinder -h`

     To see all the 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 the macsy-model of interest, you can simply follow the following steps:

- Install the macsy-models of interest from the Macsy Models repository:

  ```
  macsydata install some-public-models
  ```

- On a "unordered" genome dataset:

  ```
  macsyfinder --db-type unordered --sequence-db unordered_genome.fasta --models
  model_family all
  ```

  will search for systems corresponding to all the models of *model_family* modeled in .xml files shipped with the *"some-public-models"* macsy-model package, without taking into account the gene order.

- On a completely assembled genome (where the gene order is known):

  ```
  macsyfinder --db-type ordered_replicon --sequence-db mygenome.fasta --models-dir
  my-models --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 local macsy-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 macsy-model packages. NB: 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*.

An example data set

We provide here an example dataset comprising a replicon and the output files expected with MacSyFinder, release 2.0 when running the TXSScan macsy-models. The genomic dataset consists in the complete sequence of chromosome I from *Vibrio cholerae* O1 biovar El Tor str. N16961 (published here: https://pubmed.ncbi.nlm.nih.gov/10952301/).

The chromosome to annotate is presented as a multi-FASTA file of the proteins ordered as the genes encoding them. An annotation of the protein secretion systems and appendages was run on the genome, using the macsyfinder set of models ("macsy-model") TXSScan, V1.1.1 in the case of these examples. There are two output files offered, the one expected with the "ordered" genome mode of annotation, and the other with the "unordered" mode of genome annotation. The following command lines were used to obtain the output files:

1. The genome is downloaded from here. It will serve as an input file in the next command-line examples.

2. The TXSScan models for annotation of secretion systems are installed. The command line is the following:

```
macsydata install TXSScan # Installs the latest version of TXSScan
```

3. MacSyFinder is run on the genome, here using 8 workers for the HMM search ("-w 8" option):

   - In "ordered" mode:

```
macsyfinder --sequence-db VICH001.B.00001.C001.fasta -o macsyfinder_TXSScan_VICH001_ordered
--models TXSScan all --db-type ordered_replicon -w 8 # specified output folder: mac-
syfinder_TXSScan_VICH001_ordered
```

- In "unordered" mode:

```
macsyfinder --sequence-db VICH001.B.00001.C001.fasta -o macsyfinder_TXSScan_VICH001_unordered
--models TXSScan all --db-type unordered -w 8 # specified output folder: mac-
syfinder_TXSScan_VICH001_unordered
```

The documentation on the generated output files can be consulted *here*. See also our FAQ: *What search mode to be used?*

Note: A more comprehensive example of genome datasets with dedicated command lines and expected output files can be found here.

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 fasta file can be compressed in *gzip* format see note below)

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

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

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 CRIPRcas/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
                      (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:

```
--hmmer 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.
```

(continues on next page)

(continued from previous page)

```
                      To applied the --e-value-search to all profiles use the --no-cut-
↪ga option.
                      (default: 0.1)
--no-cut-ga           By default the MSF try to applied a threshold per profile by using␣
↪the
                      hmmer -cut-ga option. This is possible only if the GA bit score is␣
↪present in the profile otherwise
                      MF switch to use the --e-value-search (-E in hmmsearch).
                      If this option is set the --e-value-search option is used for all␣
↪profiles regardless the presence of
                      the a GA bit score in the profiles.
                      (default: False)
--cut-ga              By default the MSF try to applied a threshold per profile by using␣
↪the
                      hmmer -cut-ga option. This is possible only if the GA bit score is␣
↪present in the profile otherwise
                      MSF switch to use the --e-value-search (-E in hmmsearch).
                      But the modeler can override this default behavior to do not use␣
↪cut_ga but --e-value-search instead (-E in hmmsearch).
                      The user can reestablish the general MSF behavior, be sure the␣
↪profiles contain the GA bit score.
                      (default: True)

--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
```

(continues on next page)

(continued from previous page)

```
                      (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.
--force               force to run even the out dir already exists and is not empty.
                      Use this option with caution, MSF will erase everything in out dir␣
↪before to run.
--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 threads available will be used).
                       (default: 1)
 -v, --verbosity       Increases the verbosity level. There are 4 levels:
                       Error messages (default), Warning (-v), Info (-vv) and Debug.(-
```

(continues on next page)

(continued from previous page)

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

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.

A macsyfinder utility is provided to generate macsyfinder config file: *macsyconfig*

macsyconfig is a conversation menu which guide you and generate a file *macsyfinder.conf* in ini format. Once generated put this file in specific locations (see below) to be take in account by MacSyFinder.

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

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

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

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

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

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

In 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*)
  - *out_of_cluster* (default= *0.7*)
- **directories**
  - *res_search_dir* = (default= *./datatest/res_search*)
  - *res_search_suffix* = (default= *.search_hmm.out*)
  - *system_models_dir* = (default= *./models*)
  - *res_extract_suffix* = (default= *.res_hmm_extract*)
  - *index_dir* = (default= beside the sequence_db)
- **general**

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

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

  - *log_file* = (default = 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
```

(continues on next page)

(continued from previous page)

```
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 whre masyfinder can find models.

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

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

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 superseed models found in previous location). By default look for following directories: */share/macsyfinder/models*, or */usr/sharemacsyfinder/models* and *$HOME/.macsyfinder/models* and *system_models_dir* uses these directories if they exists.

In-house input files

Gembase format

In order to allow the users to run 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
MKLKYLSCMILASLAMGAFAATAADNNSAIYFNTTQPVNDLQGGLAAEVK
FAQSQILSAHPKEGESQQHLTSLRKSLLLVRLVKADDKTPVQVEARDAND
KILGTLTLSPPSSLPDTVYHLDGVPADGIDFTPQNGTKKIINTVAEVNKL
SDASGSSIKSYLANNALVEIQTANGRWIRDMYLPQGAELEGKMVRFVSYA
GYNSTVFYGDRKVTLSVGNTLLFKYVNGQWFRSGELENNRIAYAQHTWSA
ELPAHWIVPGLNLVIKQGNLSGSLNDINVGAPGELLLHTIDIGMLTTPRG
RFDFAKDKEAHREYFQTIPVSRMIVNNYAPLHLKEVMLPTGTLLTDADPG
>PlasmidA_0002 YP_003225073.1 | type II secretion protein EtpC
MLFFLSSRRDRNLFIKDIALKMLTPNWVLCVILLIAGYQLVSVIRHFWLT
PATSASDLSHVSVSETAVTDEHTEENFVFTLFGTASPPLSEGKVQKTTSS
LSDDLLSGGDLDVRGILYSSVTEHSVAIFAHNNRQFSLGIGEKVPGYDAT
ISAIFSDHIVINYQGKNASLPLRYDNPAKRNAQDDNNLIVGPVTTQANFR
VKNIFDIMSLSPVTVNNTLSGYRLSPGKASSLFYNAGLHDNDLAVLLNGS
ELRDTRQAKQIMKQLTELKEIKITVERDGQLYDAFIAVGEN
....
>ChromosomeA_0001 YP_003573410.1 | adhesin-like protein
MKKLFLFAALLMTGFAFYSCEDVVDNPAQDPAQSWNYSVSVKFADFDFNG
AVDENSVPYTYKAPTTLYVLNEENTLMGTITTDAAPAIGDYGTYAGTLTG
SIGNNLIITTKIGNDLTKQDGTLKSAIENGIVQTAEVPIKIYNANSGTLT
TASAKMDNTAAIAYTSLGYIKGGDKILFVEGNQTFEWTVNEEFDPYTSTD
LYIALPMNTDPETEYTISSDSKDGYTRGGTFKLADYPTLAAGKVSNYIGG
IPFIQTGVDLTKWDAYMRTDPNNTWYMNNINNGWPATFSQEVEDGKSFIV
TQSGPTLDSLNVVVGGVTGKEVNVTLNNIRLGKDRSINIGDKHGWVEYDG
THDIYGWGAKANVTLIGENECETLYIQCPATKKGEGTLNYKNLSIDSYGS
>ChromosomeA_0020 YP_003573411.1 | hypothetical protein
MKRIVLITLVSILTTFQAIAQVANGFYRVQNNASSRYITLRDNAVGTVDY
SSTNVDLSNIVTWSGFDKVKSNPASIIYVEQHDSKYDLKVQGTGIYAITG
GRTYLELRPKDSGYILAVTYNGMEGRLYDSEEDVDGEGYVKRSGNSAYQY
WSFIPVDTENNYIGLQPTVQVGDNYYGTLYASYPFKAASSGIKFYYVDAI
....
>NC_001548_0015 YP_003225080.1 | type II secretion protein EtpJ  (translation)
MSQQRVKGFTLLEMLLALAVFAALSISAFQVLQSGIRAHELSQDKVRRLA
ELQRGGSQIERDLMQMIPRHSRGSEGLLLAAPHLLKSDDWGISFTRNSWL
NPAGMLPRPELQWVGYRLRQQKLERLSYFYVDHPSGIAPDVRVVLEGVHA
FRLRFFVNGTWQARWDSTSILPQAVEVTLVMDDFAELTRLFLVSKETAE
```

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

Warning: This *gembase* format is old and not compliant with the *gembase* format produced by PanACoTA. The support of the new *gembase* format is in the road map.

Topology files

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

For example:

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

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

Output format

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

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

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

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

```
# macsyfinder 20200217.dev
# models: TFF-SF_final-0.1
# 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
```

(continues on next page)

(continued from previous page)

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

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

accessory genes:
        - T4P_pilA: 1 (T4P_pilA)
        - T4P_pilV: 1 (T4P_pilV)
        - T4P_pilY: 0 ()
        - T4P_pilW: 1 (T4P_pilW)
        - T4P_pilX: 1 (T4P_pilX)
        - T4P_fimT: 1 (T4P_fimT)
        - T4P_pilM: 1 (T4P_pilM)
        - T4P_pilP: 1 (T4P_pilP)
        - T4P_pilU: 1 (T4P_pilU)
        - MSH_mshM: 1 (MSH_mshM)

neutral genes:
```

all_systems.tsv

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

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

This file can be easily parsed using the Python pandas library.

```
import pandas as pd

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

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

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH␣
↪T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
replicon        hit_id        gene_name        hit_pos        model_fqn        sys_
↪id        sys_loci        locus_num        sys_wholeness        sys_score        sys_
↪occ        hit_gene_ref        hit_status        hit_seq_len        hit_i_
↪eval        hit_score        hit_profile_cov        hit_seq_cov        hit_begin_
↪match        hit_end_match        counterpart        used_in
GCF_000005845        GCF_000005845_000970        T4P_pilC        97        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        1        0.556        7.
↪260        1        T4P_pilC        mandatory        400        2.2e-105        353.
↪100        0.991        0.830        62        393
GCF_000005845        GCF_000005845_000980        T4P_pilB        98        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        1        0.556        7.
↪260        1        T4P_pilB        mandatory        461        8.9e-152        506.
↪100        0.948        0.850        62        453
GCF_000005845        GCF_000005845_000990        T4P_pilA        99        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        1        0.556        7.
↪260        1        T4P_pilA        accessory        146        1.1e-19        71.
↪200        0.859        0.473        5        73
GCF_000005845        GCF_000005845_025680        T4P_pilW        2568        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        2        0.556        7.
↪260        1        T4P_pilW        accessory        187        3.3e-08        34.
↪500        0.625        0.401        6        80
GCF_000005845        GCF_000005845_025690        T4P_fimT        2569        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        2        0.556        7.
↪260        1        T4P_fimT        accessory        156        2.5e-06        28.
↪500        0.939        0.397        5        66
GCF_000005845        GCF_000005845_030590        T4P_pilQ        3059        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        3        0.556        7.
↪260        1        T4P_pilQ        mandatory        412        5.9e-51        173.
↪100        0.919        0.408        244        411
GCF_000005845        GCF_000005845_030620        T4P_pilN        3062        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        3        0.556        7.
↪260        1        T4P_pilN        mandatory        179        3.8e-09        37.
↪500        0.986        0.765        5        141
GCF_000005845        GCF_000005845_030630        T4P_pilM        3063        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        3        0.556        7.
↪260        1        T4P_pilM        accessory        259        1.1e-09        39.
↪300        0.988        0.598        8        162
GCF_000005845        GCF_000005845_026740        T4P_pilT        2674        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        -1        0.556        7.
↪260        1        T4P_pilT        mandatory        326        1.1e-117        393.
↪600        0.944        0.979        3        321
GCF_000005845        GCF_000005845_026930        T2SS_gspO        2693        TFF-SF/
↪T4P        GCF_000005845_T4P_14        3        -2        0.556        7.
↪260        1        T4P_pilD        mandatory        269        1.3e-87        294.
↪000        1.000        0.859        30        260        GCF_000005845_
↪030080        GCF_000005845_T2SS_2
```

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

```
GCF_000005845   GCF_000005845_026740    T4P_pilT        2674    TFF-SF/T4P      GCF_
↪000005845_T4P_25    3       -1      0.556   7.800
GCF_000005845   GCF_000005845_026930    T2SS_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 ton encode several systems from the set of components found in this one occurrence.
- **hit_gene_ref** - the gene in the model whose this hit plays the role of
- **hit_status** - the status of the component in the assigned system's definition
- **hit_seq_len** - the length of the protein sequence matched by this hit
- **hit_i_eval** - Hmmer statistics, the independent-evalue
- **hit_score** - Hmmer score
- **hit_profile_cov** - the percentage of the profile covered by the alignment with the sequence

- **hit_seq_cov** - the percentage of the sequence covered by the alignment with the profile
- **hit_begin_match** - the position in the sequence where the profile match begins
- **hit_end_match** - the position in the sequence where the profile match ends
- **counterpart** - the hit id of some other hit which are equivalent. Only loners and multi-systems hits have counterparts
- **used_in** - whether the hit could be used in another system's occurrence

---

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

Example of *best_solution.tsv files*

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH␣
↪T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
replicon        hit_id        gene_name        hit_pos        model_fqn        sys_
↪id        sys_loci        locus_num        sys_wholeness        sys_score        sys_
↪occ        hit_gene_ref        hit_status        hit_seq_len        hit_i_
↪eval        hit_score        hit_profile_cov        hit_seq_cov        hit_begin_
↪match        hit_end_match        counterpart        used_in
GCF_000005845        GCF_000005845_000970        T4P_pilC        97        TFF-SF/
↪T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilC        mandatory        400        2.2e-105        353.
↪100        0.991        0.830        62        393
GCF_000005845        GCF_000005845_000980        T4P_pilB        98        TFF-SF/
↪T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilB        mandatory        461        8.9e-152        506.
↪100        0.948        0.850        62        453
GCF_000005845        GCF_000005845_000990        T4P_pilA        99        TFF-SF/
↪T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilA        accessory        146        1.1e-19        71.
↪200        0.859        0.473        5        73
GCF_000005845        GCF_000005845_026740        T4P_pilT        2674        TFF-SF/
↪T4P        GCF_000005845_T4P_9        1        -1        0.278        3.
↪760        1        T4P_pilT        mandatory        326        1.1e-117        393.
↪600        0.944        0.979        3        321
GCF_000005845        GCF_000005845_026930        T2SS_gspO        2693        TFF-SF/
↪T4P        GCF_000005845_T4P_9        1        -2        0.278        3.
↪760        1        T4P_pilD        mandatory        269        1.3e-87        294.
↪000        1.000        0.859        30        260        GCF_000005845_
↪030080        GCF_000005845_T2SS_2

GCF_000005845        GCF_000005845_025680        T4P_pilW        2568        TFF-SF/
↪T4P        GCF_000005845_T4P_13        2        1        0.389        4.
↪760        1        T4P_pilW        accessory        187        3.3e-08        34.
↪500        0.625        0.401        6        80
GCF_000005845        GCF_000005845_025690        T4P_fimT        2569        TFF-SF/
```

(continues on next page)

(continued from previous page)

```
↪T4P        GCF_000005845_T4P_13        2        1        0.389        4.
↪760        1        T4P_fimT        accessory        156        2.5e-06        28.
↪500        0.939        0.397        5        66
GCF_000005845        GCF_000005845_030590        T4P_pilQ        3059        TFF-SF/
↪T4P        GCF_000005845_T4P_13        2        2        0.389        4.
↪760        1        T4P_pilQ        mandatory        412        5.9e-51        173.
↪100        0.919        0.408        244        411
GCF_000005845        GCF_000005845_030620        T4P_pilN        3062        TFF-SF/
↪T4P        GCF_000005845_T4P_13        2        2        0.389        4.
↪760        1        T4P_pilN        mandatory        179        3.8e-09        37.
↪500        0.986        0.765        5        141
```

Example of *all_best_solutions.tsv files*

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH␣
↪T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
# Systems found:
sol_id        replicon        hit_id        gene_name        hit_pos        model_
↪fqn        sys_id        sys_loci        locus_num        sys_wholeness        sys_
↪score        sys_occ        hit_gene_ref        hit_status        hit_seq_
↪len        hit_i_eval        hit_score        hit_profile_cov        hit_seq_
↪cov        hit_begin_match        hit_end_match        counterpart        used_in
1        GCF_000005845        GCF_000005845_000970        T4P_pilC        97        TFF-
↪SF/T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilC        mandatory        400        2.2e-105        353.
↪100        0.991        0.830        62        393
1        GCF_000005845        GCF_000005845_000980        T4P_pilB        98        TFF-
↪SF/T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilB        mandatory        461        8.9e-152        506.
↪100        0.948        0.850        62        453
1        GCF_000005845        GCF_000005845_000990        T4P_pilA        99        TFF-
↪SF/T4P        GCF_000005845_T4P_9        1        1        0.278        3.
↪760        1        T4P_pilA        accessory        146        1.1e-19        71.
↪200        0.859        0.473        5        73
1        GCF_000005845        GCF_000005845_026740        T4P_
↪pilT        2674        TFF-SF/T4P        GCF_000005845_T4P_9        1        -
↪1        0.278        3.760        1        T4P_
↪pilT        mandatory        326        1.1e-117        393.600        0.944        0.
↪979        3        321
1        GCF_000005845        GCF_000005845_026930        T2SS_
↪gspO        2693        TFF-SF/T4P        GCF_000005845_T4P_9        1        -
↪2        0.278        3.760        1        T4P_
↪pilD        mandatory        269        1.3e-87        294.000        1.000        0.
↪859        30        260        GCF_000005845_030080        GCF_000005845_T2SS_2

1        GCF_000005845        GCF_000005845_025680        T4P_
↪pilW        2568        TFF-SF/T4P        GCF_000005845_T4P_
↪13        2        1        0.389        4.760        1        T4P_
↪pilW        accessory        187        3.3e-08        34.500        0.625        0.
```

(continues on next page)

(continued from previous page)

```
↪401        6        80
1        GCF_000005845        GCF_000005845_025690        T4P_
↪fimT        2569        TFF-SF/T4P        GCF_000005845_T4P_
↪13        2        1        0.389        4.760        1        T4P_
↪fimT        accessory        156        2.5e-06        28.500        0.939        0.
↪397        5        66
1        GCF_000005845        GCF_000005845_030590        T4P_
↪pilQ        3059        TFF-SF/T4P        GCF_000005845_T4P_
↪13        2        2        0.389        4.760        1        T4P_
↪pilQ        mandatory        412        5.9e-51        173.100        0.919        0.
↪408        244        411
1        GCF_000005845        GCF_000005845_030620        T4P_
↪pilN        3062        TFF-SF/T4P        GCF_000005845_T4P_
↪13        2        2        0.389        4.760        1        T4P_
↪pilN        mandatory        179        3.8e-09        37.500        0.986        0.
↪765        5        141
1        GCF_000005845        GCF_000005845_030630        T4P_
↪pilM        3063        TFF-SF/T4P        GCF_000005845_T4P_
↪13        2        2        0.389        4.760        1        T4P_
↪pilM        accessory        259        1.1e-09        39.300        0.988        0.
↪598        8        162
1        GCF_000005845        GCF_000005845_026740        T4P_
↪pilT        2674        TFF-SF/T4P        GCF_000005845_T4P_13        2        -
↪1        0.389        4.760        1        T4P_
↪pilT        mandatory        326        1.1e-117        393.600        0.944        0.
↪979        3        321
1        GCF_000005845        GCF_000005845_026930        T2SS_
↪gspO        2693        TFF-SF/T4P        GCF_000005845_T4P_13        2        -
↪2        0.389        4.760        1        T4P_
↪pilD        mandatory        269        1.3e-87        294.000        1.000        0.
↪859        30        260        GCF_000005845_030080        GCF_000005845_T2SS_2

1        GCF_000005845        GCF_000005845_029970        T2SS_
↪gspC        2997        TFF-SF/T2SS        GCF_000005845_T2SS_
↪1        1        1        0.857        9.000        1        T2SS_
↪gspC        mandatory        271        2.3e-19        70.400        0.897        0.
↪358        47        143
1        GCF_000005845        GCF_000005845_030050        T2SS_
↪gspK        3005        TFF-SF/T2SS        GCF_000005845_T2SS_
↪1        1        1        0.857        9.000        1        T2SS_
↪gspK        accessory        327        1e-16        61.500        1.000        0.
↪180        6        64
1        GCF_000005845        GCF_000005845_030060        T2SS_
↪gspL        3006        TFF-SF/T2SS        GCF_000005845_T2SS_
↪1        1        1        0.857        9.000        1        T2SS_
↪gspL        accessory        387        1.5e-37        129.300        1.000        0.
↪351        6        141
1        GCF_000005845        GCF_000005845_030070        T2SS_
↪gspM        3007        TFF-SF/T2SS        GCF_000005845_T2SS_
↪1        1        1        0.857        9.000        1        T2SS_
↪gspM        accessory        153        2.8e-29        102.900        0.985        0.
↪804        13        135
```

(continues on next page)

(continued from previous page)

```
1        GCF_000005845        GCF_000005845_030080        T2SS_
↪gspO        3008        TFF-SF/T2SS        GCF_000005845_T2SS_
↪1        1        1        0.857        9.000        1        T2SS_
↪gspO        mandatory        225        4e-65        220.400        0.978        0.
↪840        26        214

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

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

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

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

```
1       GCF_000005845   GCF_000005845_030080    T2SS_gspO       3008    TFF-SF/T2SS     ␣
↪GCF_000005845_T2SS_1    1       1       0.857   9.000   1       T2SS_gspO       ␣
↪mandatory       225     4e-65   220.400 0.978   0.840   26      214

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

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

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

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

best_solution_summary.tsv

This file is a concise view of which systems have been found in your replicons and how many per replicon. It is based on **best_solution.tsv**. The first two lines are comments that indicate the version of 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 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type=gembase --models-dir=tests/data//models/ --models TFF-SF Archaeal-T4P ComM MSH␣
↪T2SS T4bP T4P Tad --relative-path --sequence-db tests/data/base/gembase.fasta -w 12
replicon        TFF-SF/MSH        TFF-SF/T2SS        TFF-SF/T4P        TFF-SF/
↪T4bP        TFF-SF/Tad        TFF-SF/Archaeal-T4P        TFF-SF/ComM
GCF_000005845        0        1        2        0        0        0        0
GCF_000006725        0        1        2        0        0        0        0
GCF_000006745        1        1        2        1        0        0        0
GCF_000006765        0        3        1        0        1        0        0
GCF_000006845        0        0        1        0        0        0        0
GCF_000006905        0        1        0        0        1        0        0
GCF_000006925        0        0        1        0        0        0        0
GCF_000006945        0        0        2        0        0        0        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
```

best_solution_loners.tsv

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

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/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
# Loners found:
replicon        model_fqn        function        gene_name        hit_
↪id        hit_pos        hit_status        hit_seq_len        hit_i_
↪eval        hit_score        hit_profile_cov        hit_seq_cov        hit_
↪begin_match        hit_end_match
GCF_000005845        TFF-SF/T4P        T4P_pilT        T4P_pilT        GCF_
↪000005845_026740        2674        mandatory        326        1.100e-
↪117        393.600        0.944        0.979        3        321
GCF_000005845        TFF-SF/T4P        T4P_pilD        T2SS_gspO        GCF_
↪000005845_026930        2693        mandatory        269        1.300e-
↪87        294.000        1.000        0.859        30        260
GCF_000005845        TFF-SF/T4P        T4P_pilD        T2SS_gspO        GCF_
↪000005845_030080        3008        mandatory        225        4.000e-
↪65        220.400        0.978        0.840        26        214
GCF_000006725        TFF-SF/T4P        T4P_pilT        T4P_pilT        GCF_
↪000006725_000270        4269        mandatory        344        1.800e-
↪172        573.700        0.994        0.985        2        340
GCF_000006725        TFF-SF/T4P        T4P_pilA        T4P_pilA        GCF_
↪000006725_003680        4610        accessory        187        9.000e-
↪10        39.500        0.667        0.278        6        57
GCF_000006725        TFF-SF/T2SS        T2SS_gspO        T4P_pilD        GCF_
```

(continues on next page)

(continued from previous page)

```
↪000006725_014570        5699        mandatory        287        7.400e-
↪77        258.600        1.000        0.836        28        267
GCF_000006725        TFF-SF/T2SS        T2SS_gspE        T2SS_gspE        GCF_
↪000006725_018700        6112        mandatory        566        1.800e-
↪171        571.000        0.936        0.701        165        561
GCF_000006725        TFF-SF/T4P        T4P_pilA        T4P_pilA        GCF_
↪000006725_022640        6506        accessory        178        2.000e-
↪10        41.600        0.603        0.264        5        51
GCF_000006745        TFF-SF/T2SS        T2SS_gspO        T4P_pilD        GCF_
↪000006745_021980        8766        mandatory        291        3.100e-
↪88        295.800        1.000        0.832        28        269
GCF_000006765        TFF-SF/T2SS        T2SS_gspO        T4P_pilD        GCF_
↪000006765_044730        14545        mandatory        290        1.100e-
↪88        297.200        1.000        0.828        31        270
GCF_000006925        TFF-SF/T4P        T4P_pilT        T4P_pilT        GCF_
↪000006925_026070        23874        mandatory        341        6.600e-
↪118        394.300        0.950        0.941        18        338
GCF_000006945        TFF-SF/T4P        T4P_pilT        T4P_pilT        GCF_
↪000006945_030160        28596        mandatory        326        3.400e-
↪113        378.800        0.933        0.966        3        317
GCF_000006945        TFF-SF/T4P        T4P_pilD        T2SS_gspO        GCF_
↪000006945_033450        28925        mandatory        155        2.900e-
↪35        122.700        0.588        0.871        9        143
```

best_solution_multisystems.tsv

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

```
# macsyfinder 20220121.dev
# models : functional-0.0b2
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --db-
↪type ordered_replicon --replicon-topology linear --models-dir tests/data/
↪models/ -m functional T12SS-multisystem --relative-path --sequence-db tests/
↪data/base/test_13.fasta -w 15
# Multisystems found:
replicon        model_fqn        function        gene_name        hit_
↪id        hit_pos        hit_status        hit_seq_len        hit_i_
↪eval        hit_score        hit_profile_cov        hit_seq_cov        hit_
↪begin_match        hit_end_match
UserReplicon        functional/T12SS-multisystem        T1SS_omf        T1SS_
↪omf        VICH001.B.00001.C001_
↪01360        20        mandatory        484        3.200e-28        90.
↪000        0.985        0.820        80        476
UserReplicon        functional/T12SS-multisystem        T1SS_omf        T1SS_
↪omf        VICH001.B.00001.C001_
↪01506        35        mandatory        419        9.100e-35        111.
↪500        0.998        0.912        25        406
```

rejected_candidates.txt

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

The header is composed of the 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.

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

```
# macsyfinder 20200511.dev
# models : TFF-SF-0.1b
# macsyfinder --sequence-db data/base/GCF_000006745.fasta --models TFF-SF all --models-
↪dir data/models/ --db-type gembase -w 4
# Rejected candidates:

Cluster:
    - model: T4P
    - hits: (GCF_000005845_025680, T4P_pilW, 2568), (GCF_000005845_025690, T4P_fimT,␣
↪2569)
Cluster:
    - model: T4P
    - hits: (GCF_000005845_026930, T2SS_gspO, 2693)
Cluster:
    - model: T4P
    - hits: (GCF_000005845_030080, T2SS_gspO, 3008)
This candidate has been rejected because:
The quorum of mandatory genes required (4) is not reached: 1
The quorum of genes required (5) is not reached: 3
================================================================
Cluster:
    - model: Archaeal-T4P
    - hits: (GCF_000005845_019260, Archaeal-T4P_arCOG00589, 1926), (GCF_000005845_019310,
↪ Archaeal-T4P_arCOG02900, 1931)
This candidate has been rejected because:
The quorum of mandatory genes required (3) is not reached: 0
The quorum of genes required (3) is not reached: 2
================================================================
```

rejected_candidates.tsv

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

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

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

- the macsyfinder version

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

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

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

Note: A rejected candidate can be constituted of

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

Example of *rejected_candidates.tsv*

```
# macsyfinder 20220805.dev
# models : TFF-SF-None
# /home/bneron/Projects/GEM/MacSyFinder/MacSyFinder/py39/bin/macsyfinder --sequence-db␣
↪data/base/GCF_000006745.fasta --models TFF-SF all --models-dir data/models/ --db-type␣
↪gembase -w 15
# Rejected candidates found:
candidate_id        replicon        model_fqn        cluster_id        hit_id        hit_
↪pos        gene_name        function        reasons
GCF_000006745_Archaeal-T4P_1        GCF_000006745        TFF-SF/Archaeal-
↪T4P        c3        GCF_000006745_018740        1874        Archaeal-T4P_
↪arCOG00589        Archaeal-T4P_arCOG00589        The quorum of mandatory genes␣
↪required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1
GCF_000006745_Archaeal-T4P_1        GCF_000006745        TFF-SF/Archaeal-
↪T4P        c3        GCF_000006745_018800        1880        Archaeal-T4P_
↪arCOG00589        Archaeal-T4P_arCOG00589        The quorum of mandatory genes␣
↪required (3) is not reached: 0/The quorum of genes required (3) is not reached: 1

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

(continues on next page)

(continued from previous page)

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

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

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

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

GCF_000006745_T2SS_7        GCF_000006745        TFF-SF/T2SS        c25        GCF_
↪000006745_021980        2198        T4P_pilD        T2SS_gspO        The quorum of␣
↪mandatory genes required (4) is not reached: 1/The quorum of genes required (6) is not␣
↪reached: 1

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

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

(continues on next page)

(continued from previous page)

```
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12        GCF_000006745        TFF-SF/T4P        c35        GCF_
↪000006745_007850        785        T4P_pilX        T4P_pilX        The quorum of␣
↪mandatory genes required (4) is not reached: 2
GCF_000006745_T4P_12        GCF_000006745        TFF-SF/T4P        c35        GCF_
↪000006745_007860        786        T4P_pilV        T4P_pilV        The quorum of␣
↪mandatory genes required (4) is not reached: 2
```

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

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

for instance

```
# macsyfinder 20230113.dev
# models : TXSScan-1.1.1
# macsyfinder --sequence-db tests/data/base/gembase.fasta --db-type gembase --models␣
↪TXSScan -w 15 --timeout 1s
#
# WARNING: The replicon 'GCF_000006765' has been SKIPPED. Cannot be solved before␣
↪timeout.
#
replicon  hit_id ...
```

Output files for the "unordered replicon" search mode

Systems detection results

As for ordered replicons, several output files are provided.

- *all_systems.txt* - This file contains the description of candidate systems found.
- *all_systems.tsv* - The same information as in *all_systems.txt* but in the tabulated tsv format.
- *uncomplete_systems.txt* - This file contains occurrences for systems that did not complete models' definitions and that were therefore not kept as candidate systems.

Note: In this *unordered* search mode, there is no notion of order or distance of the components along the replicon. The clustering step is skipped by 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
# models : TFF-SF-0.1b
# 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

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)
```

(continues on next page)

(continued from previous page)

```
    - 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
# models : TFF-SF-0.1b
# 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_
↪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       286     2.8e-
↪82 272.300 1.000   0.829   28      264
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
```

(continues on next page)

(continued from previous page)

```
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 list models that probably do not have not 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
# models : TFF-SF-0.1b
# 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.

============================================================
```

Hmmer results' output files

Raw Hmmer outputs are provided, as long with processed tabular outputs that include hits filtered as specified by the user. For instance, the Hmmer 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_eval␣
↪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**.

For big data people

Parallelization

The time limiting part are HMMER (search genes). If you want to deal with a large data

- a collection of file containing replicons (each file must contains one replicon)
- or a gembase file (with a lot of replicons, from ten to more than thousand)

we provide a workflow to parallelize the execution by the data. This mean that

1. We split the data input into chunks containing one replicon each (for *gembase* input file).
2. Then execute MacSyFinder in parallel on each replicon (the number of parallel tasks can be limited)
3. Then aggregate the results in one global summary.

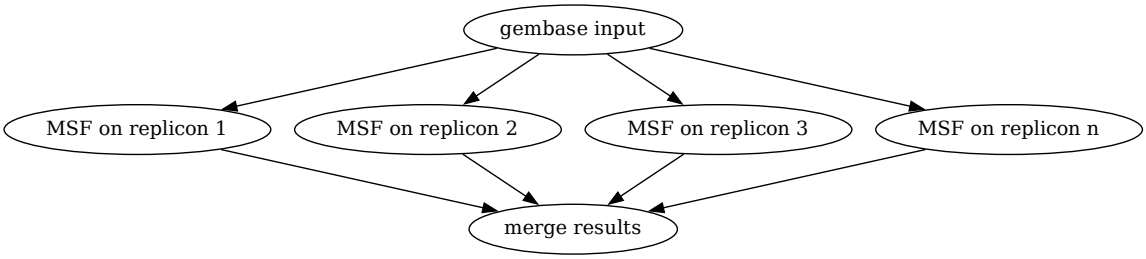


Fig. 1: Diagram of the parallel_macsyfinder workflow on gembase input

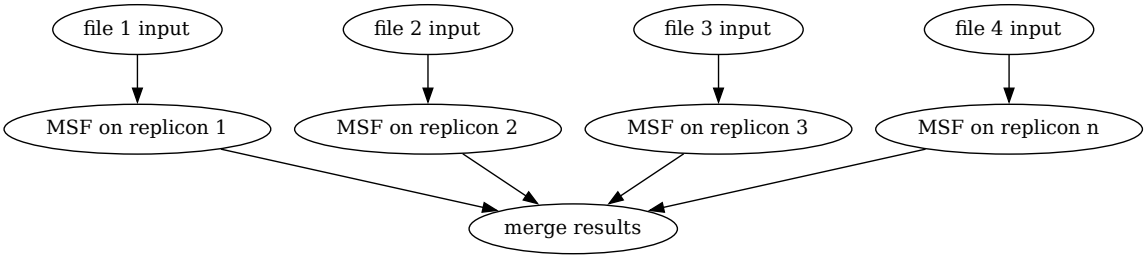


Fig. 2: Diagram of the parallel_macsyfinder workflow on ordered or unordered replicon

The workflow use the nextflow framework and can be run on a single machine or a cluster.

First, you have to install nextflow first, and *macsyfinder*. Then we provide 2 files (you need to download them from the MacSyFinder github repo.)

- *parallel_macsyfinder.nf* which is the workflow itself in nextflow syntax
- *nextflow.config* which is a configuration file to execute the workflow.

The workflow file should not be modified. Whereas the profile **must** be adapted to the **local** architecture.

The file *nextflow.config* provide five profiles:

- a standard profile for local use (single machine).
- an apptainer profile using docker image with apptainer executor (on a single machine).
- a docker profile using docker image with docker executor (on a single machine).
- a cluster profile.
- a cluster profile using apptainer container system with the docker image.

How to get parallel_macsyfinder

The release contains the workflow *parallel_macsyfinder.nf* and the *nextflow.config* at the top level of the archive But if you use pip to install MacSyFinder you have not easily access to them. But they can be downloaded or executed directly by using nextflow.

to download it

```
nextflow pull gem-pasteur/macsyfinder
```

to get the latest version or use *-r* option to specify a version

```
nextflow pull -r release_2.0 gem-pasteur/macsyfinder
```

to see what you download

```
nextflow view macsyfinder
```

to execute it directly on a local host with macsyfinder already installed and with with models installed too:

```
nextflow run gem-pasteur/macsyfinder -profile standard --models "TFF-SF all" --db-type␣
↪gembase --sequence-db <path/to/my/gembase.fasta>
```

or:

```
nextflow run -r release_2.0 gem-pasteur/macsyfinder -profile standard --models "TFF-SF␣
↪all" --db-type gembas --sequence-db <path/to/my/gembase.fasta>
```

or for ordered replicon

```
nextflow run gem-pasteur/macsyfinder -profile cluster_apptainer --models "TFF-SF all" --
↪db-type ordered_replicon --sequence-db '<path/to/replicons/*.fasta>' --outdir <my_
↪results>
nextflow run gem-pasteur/macsyfinder -profile cluster_apptainer --models "TFF-SF all" --
↪db-type ordered_replicon --sequence-db 'file1.fasta,file2.fasta,file3.fst' --outdir
↪<my_results>
```

or if you download the macsyfinder repository, or the the workflow with it's configuration file:

```
nextflow run  parallel_macsyfinder.nf -profile standard --models "TFF-SF all" --db-type␣
↪ordered_replicon --sequence-db 'data/base/split/GCF_*.fasta' --outdir GCF
```

Note:

- For *gembase* data the workflow expected one file with several replicons.
- For *ordered_replicon* or *unordered* the workflow expected several files with one replicon per file.

Warning: See the double quotes surrounding the models value *–models "TFF-SF all"* with out quoting macsyfinder will not received the right argument.

Warning: See the (double) quotes surrounding the models value *–sequence-db '<path/to/replicons/*.fasta>'* with out quoting parallel_macsyfinder will not received all files.

Warning: When you analyzed ordered or unordered replicons (*–db-type* set to *ordered_replicon* or *unordered*) the *–out-dir* option is **REQUIRED**.

standard profile

This profile is used if you want to parallelize MacSyFinder on your machine. You can specify the number of tasks in parallel by setting the *queueSize* value You can also fix the number of cpu used by each task (macsyfinder –worker option see *macsyfinder options*) by setting the *params.worker* parameter in *nextflow.config*

```
standard {
    executor {
        name = 'local'
        queueSize = 4
    }
    process {
        errorStrategy = 'ignore'
        withName: macsyfinder {
            cpus = params.worker
        }
    }
}
```

Almost options available in non parallel version are also available for the parallel one. except: * `--db-type` which is set to *gembase* (only data type supported for the parallelized macsyfinder version). * `--out-dir` which is not available.

A typical command line will be:

```
./parallel_macsyfinder.nf -profile standard --models "TFF-SF all" --sequence-db <path/to/
↪my/gembase.fasta>
```

Note: The options starting with one dash are for nextflow workflow engine, whereas the options starting by two dashes are for macsyfinder workflow.

If you execute this line, 2 kinds of directories will be created.

- One named *work* containing lot of subdirectories this for all jobs launch by nextflow.
- Directories named *merged_macsyfinder_results_XXX* where XXX is the name of the gembase file. This directory contain the final results as in non parallel version.

standard_apptainer or standard_docker profile

If you have not installed *macsyfinder* but you use it through a container docker or https://apptainer.org/ (former *singularity*) We provide profiles for these situations. With the command line below nextflow will download parallel_macsyfinder from github and download the macsyfinder image from the docker-hub (https://hub.docker.com/r/gempasteur/macsyfinder) (and apptainer convert the image on the right format on the fly) so you haven't to install anything except nextflow and apptainer or docker.

```
standard_apptainer {
    executor {
        name = 'local'
        queueSize = 4
    }
    process {
        errorStrategy = 'ignore'
        container = 'docker://gempasteur/macsyfinder:latest'
        withName: macsyfinder {
            cpus = params.worker
        }
    }
    singularity {
        enabled = true
   }
}
```

```
standard_docker {
    executor {
        name = 'local'
        queueSize = 4
    }
    process {
        errorStrategy = 'ignore'
        container = 'macsyfinder'
        withName: macsyfinder {
            cpus = params.worker
        }
    }
    docker {
        enabled = true
        runOptions = '--user $(id -u):$(id -g)'
   }
}
```

The execution is similar than for installed macsyfinder

```
./parallel_macsyfinder.nf -profile standard_apptainer --models "TFF-SF all" --sequence-
↪db <path/to/my/gembase.fasta>
```

or

```
./parallel_macsyfinder.nf -profile standard_docker --models "TFF-SF all" --sequence-db
↪<path/to/my/gembase.fasta>
```

cluster profile

The cluster profile is intended to work on a cluster managed by SLURM. If your cluster is managed by an other drm replace executor name by the right value (see nextflow supported cluster)

You can also manage

- The number of tasks in parallel with the *executor.queueSize* parameter (here 500). If you remove this line, the system will send in parallel as many jobs as there are replicons in your data set.
- The queue (or partition in *Slurm* teminology) with *process.queue* parameter (here *common,dedicated*)
- and some options specific to your cluster management systems with *process.clusterOptions* parameter

```
cluster {
    executor {
        name = 'slurm'
        queueSize = 500
    }

    process {
        errorStrategy = 'ignore'
        queue = 'common,dedicated'
        clusterOptions = '--qos=fast'
        withName: macsyfinder {
            cpus = params.worker
        }
    }
}
```

To run the parallel version on cluster, for instance on a cluster managed by slurm, I can launch the main nextflow process in one slot. The parallelization and the submission on the other slots is made by nextflow itself. Below a command line to run parallel_macsyfinder and use 3 cpus per macsyfinder task, each macsyfinder task can be executed on different machine, each macsyfinder task claim 2 cpus/cores (cpu in *nextflow* terminology/ cores for hardware) to speed up the genes search.

```
sbatch --qos fast -p common nextflow run parallel_macsyfinder.nf -profile cluster --
↪models "TFF-SF all" --sequence-db <path/to/my/gembase.fasta> --worker 3
```

The results will be the same as describe in local execution.

cluster_apptainer profiles

You can also use the macsyfinder apptainer image on a cluster, for this use the profile *cluster_apptainer*.

```
sbatch --qos fast -p common nextflow run  gem-pasteur/macsyfinder -profile cluster_
↪apttainer --models "TFF-SF all" --sequence-db <path/to/my/gembase.fasta>
```

In the case of your cluster cannot reach the world wide web. you have to download the singularity image

```
apptainer pull --name macsyfinder.simg docker://gempasteur/macsyfinder
```

Then move the image on your cluster modify the nextflow.config to point on the location of the image, and adapt the cluster options (executor, queue, ...) to your architecture

```
 cluster_apptainer {
    executor {
        name = 'slurm'
        queueSize = 500
    }

    process {
        errorStrategy = 'ignore'
        container = '/path/to/macsyfinder.simg'
        queue = 'common,dedicated'
        clusterOptions = '--qos=fast'
        withName: macsyfinder {
            cpus = params.worker
        }
    }
    singularity {
        enabled = true
        runOptions = '-H $HOME -B /pasteur'
        autoMounts = false
   }
}
```

then run it

```
sbatch --qos fast -p common nextflow run  ./parallel_macsyfinder.nf -profile cluster_
↪apptainer --models "TFF-SF all" --sequence-db <path/to/my/gembase.fasta>
```

If you want to have more details about the jobs execution you can add some options to generate report:

Execution report

To enable the creation of this report add the `-with-report` command line option when launching the pipeline execution. For example:

```
nextflow run  ./parallel_macsyfinder.nf -profile standard -with-report [file name] --
↪models "TFF-SF all" --sequence-db <path/to/my/gembase.fasta>
```

It creates an HTML execution report: a single document which includes many useful metrics about a workflow execution. For further details see https://www.nextflow.io/docs/latest/tracing.html#execution-report

Trace report

In order to create the execution trace file add the `-with-trace` command line option when launching the pipeline execution. For example:

```
nextflow run  ./parallel_macsyfinder.nf -profile standard -with-trace --models "TFF-SF␣
↪all" --sequence-db <path/to/my/gembase.fasta>
```

It creates an HTML timeline for all processes executed in your pipeline. For further details see https://www.nextflow.io/docs/latest/tracing.html#timeline-report

Timeline report

To enable the creation of the timeline report add the `-with-timeline` command line option when launching the pipeline execution. For example:

```
nextflow run  ./parallel_macsyfinder.nf -profile standard -with-timeline [file name] --
↪models "TFF-SF all" --sequence-db <path/to/my/gembase.fasta> ...
```

It creates an execution tracing file that contains some useful information about each process executed in your pipeline script, including: submission time, start time, completion time, cpu and memory used. For further details see https://www.nextflow.io/docs/latest/tracing.html#trace-report

Warning: When you run parallelize version of macsyfinder the hhm score for each genes can be different than in non parallel version. As hmmsearch use the size of the sequence database to compute the score.

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 definition` to display one or a set of model defintion
- `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 install models in specific directory with the *–target* option.

> macsydata install –target <my_models>

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

> macsyfinder –db-type ordered_replicon –models-dir=my_models –models TFF-SF all –sequence-db my_genome.fasta

The path must point at a directory that contains macsy-model packages as described *above*.

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

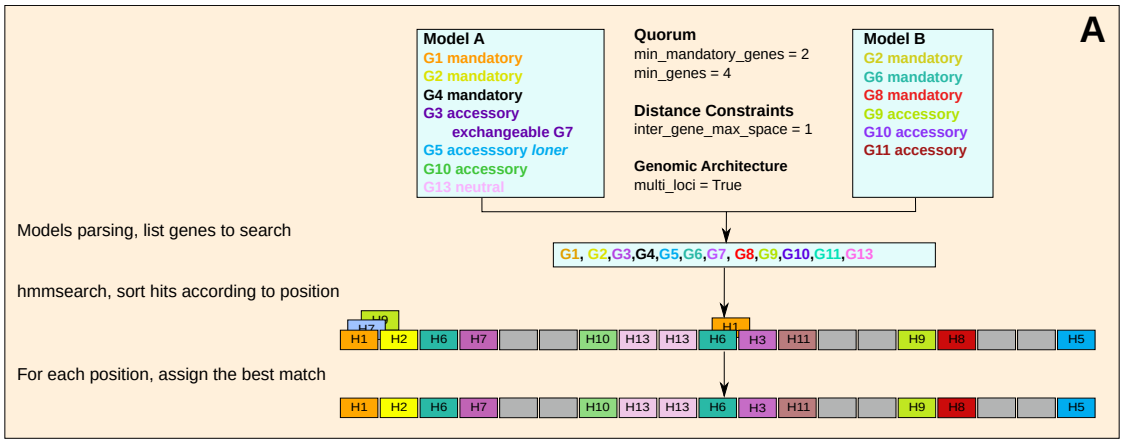
Initially, MacSyFinder **searches for the components** of the *System(s)* to detect by sequence similarity search.

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).

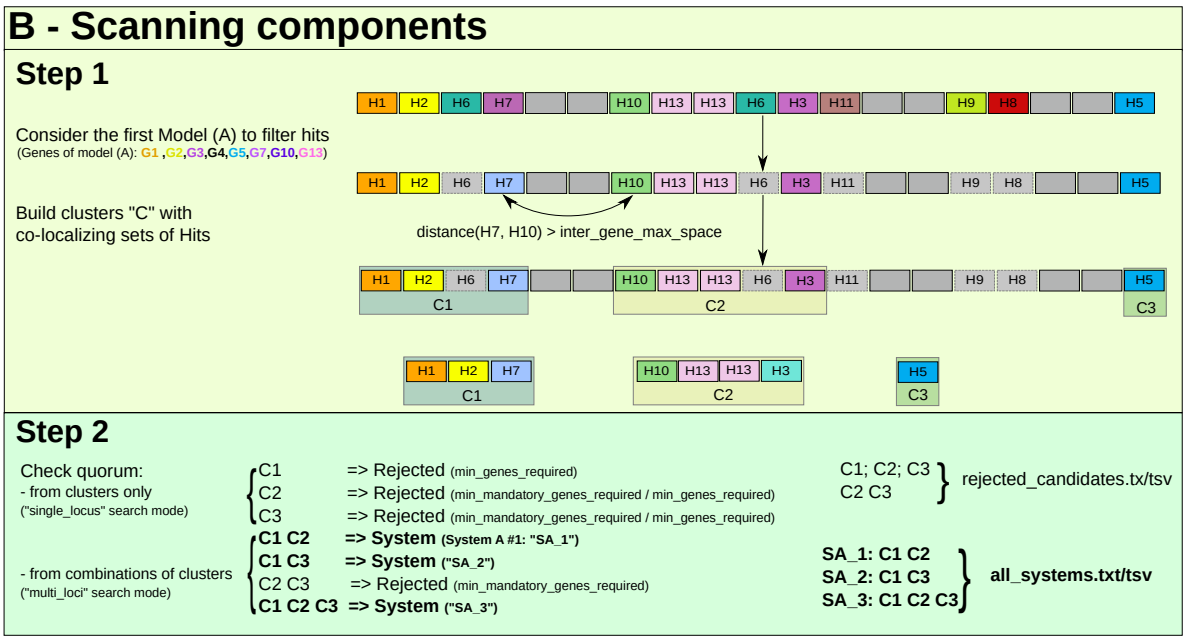
In the case of **ordered datasets** (*ordered_replicon* or *gembase* search mode), the hits are filtered to keep only hits related to the system's model we are looking for. These hits are used to build **clusters of co-localized genes** as defined *in the macsy-model files*. These clusters are then screened 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_candidates.txt/tsv* file.

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

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

For *unordered* datasets:

For each model being searched:

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

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

For unordered datasets, the **search so ends**, and 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_candidates.txt* and *rejected_candidates.tsv* output files.

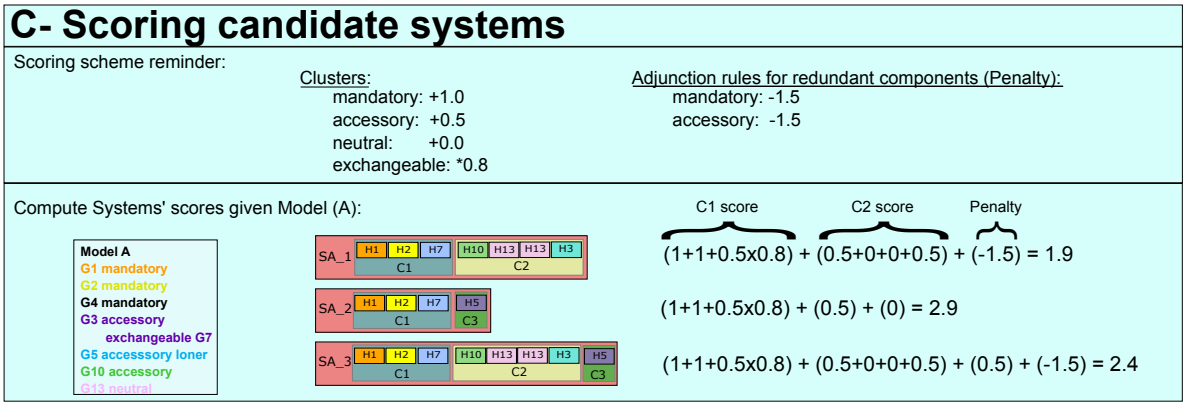
- We introduce a **scoring scheme for candidate Systems**, to easily separate combinations of clusters that are readily more similar to a system's model than others.

  The assumptions behind this scoring scheme are the following:

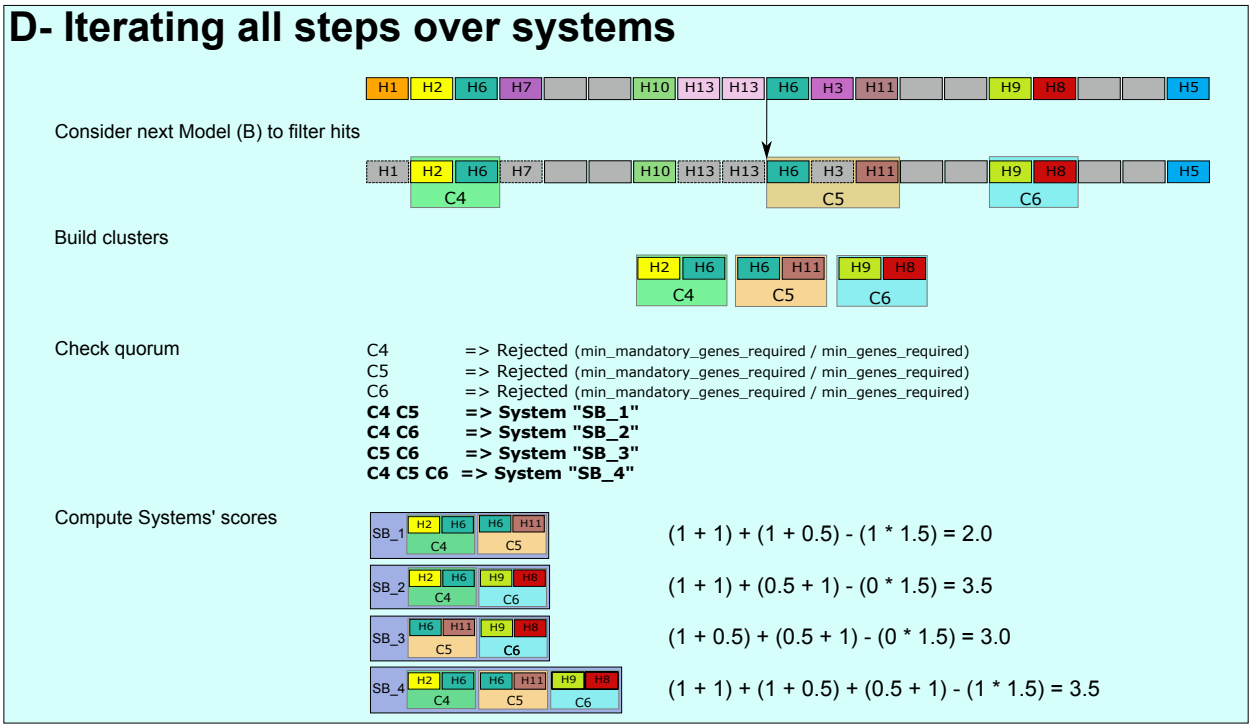
  - We set a score for the different types of genes/components when defining a **cluster's score**. Here are the default values, but these *can be changed*:
    - +1.0 is added when a *mandatory* gene is present
    - +0.5 is added when an *accessory* gene is present
    - +0.0 is added when a *neutral* gene is present
    - *0.8 (a factor of 0.8) is applied to the above-scores when the function is fulfilled by an *exchangeable* gene
    - *0.7 (a factor of 0.7) is applied to the above-scores if the gene is a *loner* and *multi system* component.
  - When combinations of clusters are explored in order to fulfill macsy-models' requirements and build candidate systems ("multi_loci" mode, several clusters can make a complete *System*), we sum the score of clusters to assign a *System*'s score.
  - In addition, we want to **favor concise sets of clusters** to fulfill a *System*'s model. We thus **penalize the adjunction of a cluster** to a candidate *System* when this cluster does not bring any new components to the *System*'s quorum, or when it brings **redundant components**. Thus:
    - -1.5 is added when a **redundant** mandatory gene is added when adjuncting the cluster to a candidate *System*
    - -1.5 is added when a **redundant** accessory gene is added when adjuncting the cluster to a candidate *System*
    - for the components that are *loner* and *multi system*, the score of the loner component is added only if the function is not fulfilled in the other clusters. In this case, even if there are several occurrences of the component, it is counted only once (and no penalty is applied).

– Only candidate sets of clusters that fulfill a macsy-model and that are thus designated candidate *Systems*, obtain a **System's score**

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



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



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.

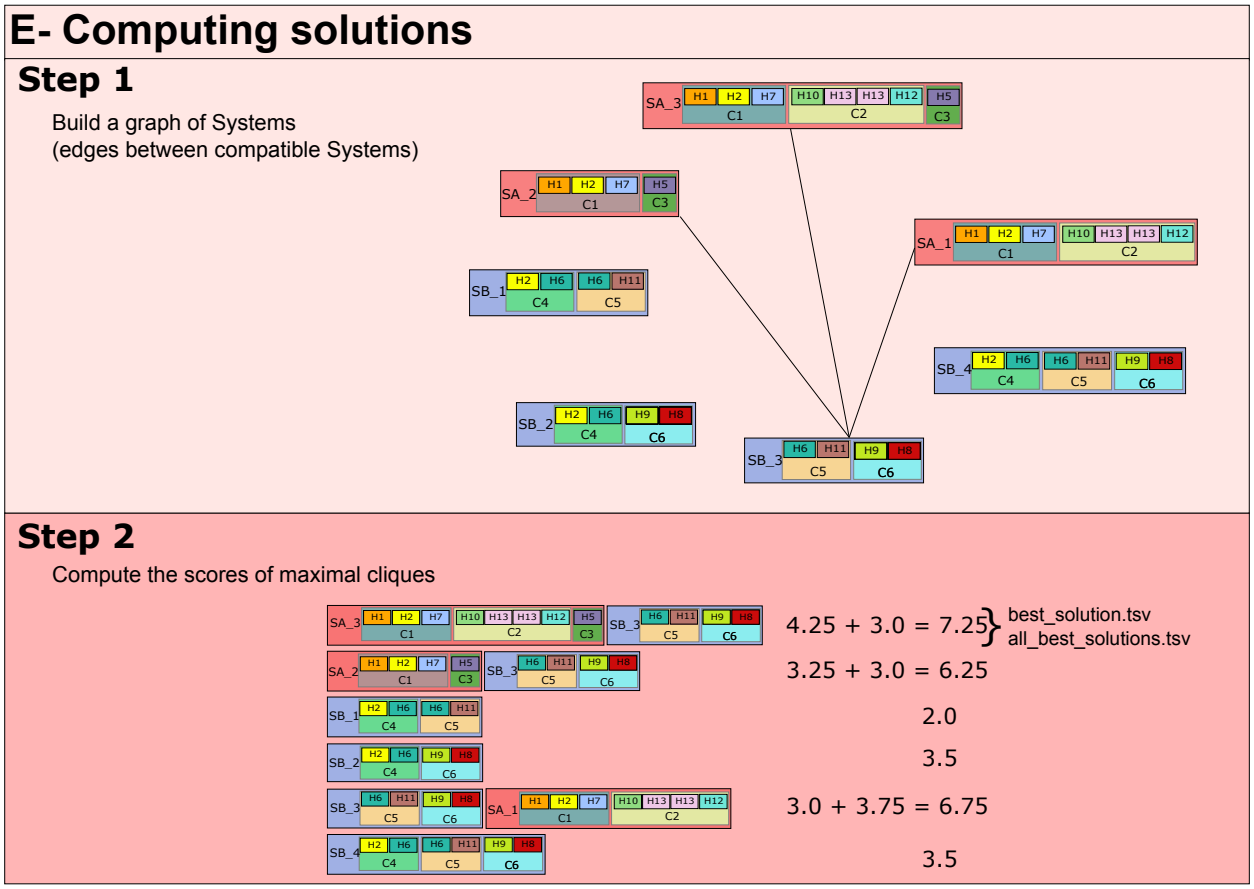
E. Computing possible Solutions, defining the best one (ordered mode)

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 (TXSScan: T1SS, T2SS, T5SS, T6SS, T9SS, Tad, T4P).
- Denise et al. 2019, *PLoS Biology*, for the description of type IV-filament super-family models (TFF-SF: T2SS, T4aP, T4bP, Com, Tad, archaeal T4P).
- Rendueles et al. 2017, *PLoS Pathogens*, for the CapsuleFinder set of models.
- Couvin, Bernheim et al. 2018, *Nucleic Acids Research*, for the updated version of the set of Cas systems' models, CasFinder.

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 environement 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 deal with fragmented genomes (MAGs, SAGs, draft genomes)?

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

1. If your genome is at least partially assembled and contigs are not too short, you might "feel lucky" and first consider to run MacSyFinder with the *ordered_replicon* mode. It could be particularly efficient if you are investigating systems encoded by compact loci (Cas systems, some secretion systems…), as they might be encoded by a single contig.

2. On top of the *ordered_replicon* mode, you might add the option "multi-loci" to the systems to annotate (if not already the case), in order to maximize the chance to annotate an entire system, even if encoded across several contigs.

3. The *unordered* mode can be used in complement of the two above options, e.g. to retrieve some of the missing components. It will enable to assess the genetic potential and possible presence of a system, independently of the quality of assembly of the genome. It might also be the only reasonable option if the genome is too fragmented and/or too incomplete.

Note:

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

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

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

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

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

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
```

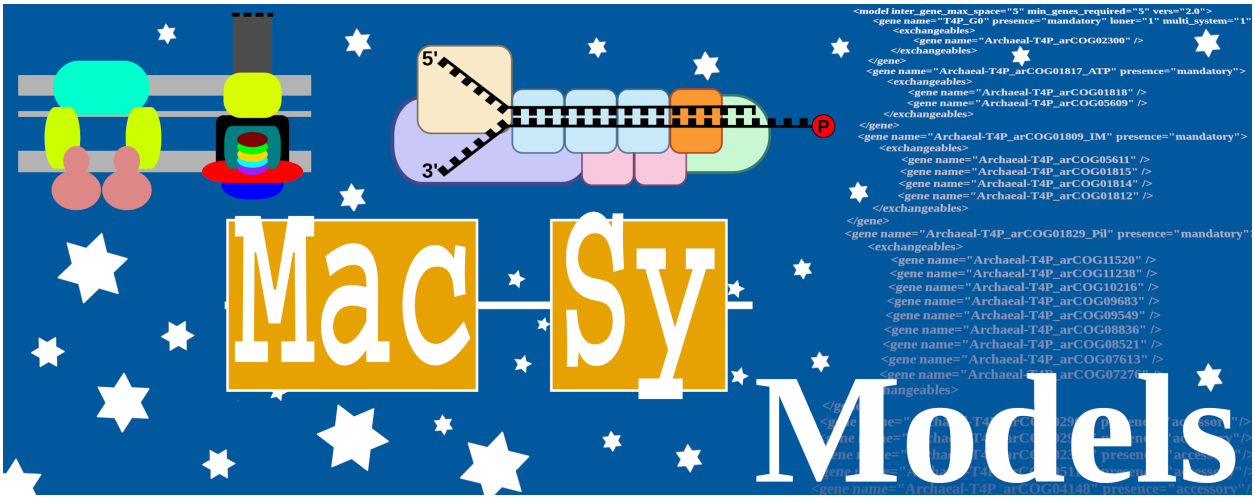
- **system**: the *macsyfinder.conf* file either in /etc/macsyfinder/ or in ${VIRTUAL_ENV}/etc/macsyfinder/ in case of a *virtualenv* this configuration affects only the MacSyFinder version installed in this virtualenv
- **home**: the *~/.macsyfinder/macsyfinder.conf* file
- **model**: the *model_conf.xml* 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

CHAPTER

TWO

MODELLER GUIDE

2.1 Modeller Guide



2.1.1 Modelling Systems with MacSyFinder

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. We also provide container so you can use macsyfinder directly.

MacSyFinder Installation procedure

To develop new models and share them, MacSyFinder requires *git* and the python librarie *GitPython*

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

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

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

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

From Conda/Mamba

From version 2.0 and above, MacSyFinder is packaged for Conda/Mamba The Conda/Mamba package include modeler dependencies

From container

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

Models installation with *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 (https://github.com/macsy-models) or locally installed. The general syntax for *macsydata* is:

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

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

```
macsydata available
```

To search for models on *macsy-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

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

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

To install it in any directory:

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

To know how to cite a model package:

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

To show the model definition:

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

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

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

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

```
macsydata definition TXSS+/bacterial
```

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

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

above macsydata with required options. Below I add optioanl but recommended options.

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

To list all *macsydata* subcommands:

```
macsydata --help
```

To list all available options for a subcommand:

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

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

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

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

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

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

Structure of a macsy-model package

A macsy-model package follows the following structure:

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

If the package contains sub-families:

```
family_name
    |_______ metadata.yml
    |_______ LICENSE
    |_______ README.md
    |_______ model_conf.xml
    |_______ definitions
    |             |________ subfamilyA
    |             |             |________ model_1.xml
    |             |             |________ model_2.xml
    |             |
    |             |________ subfamilyB
    |             |             |________ model_3.xml
    |             |             |________ model_4.xml
    |             |
    |             :
    |
    |_______ profiles
                  |________ geneA.hmm
                  |________ geneB.hmm
                  |________ geneC.hmm.gz
```

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

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

- the data package directory with the right structure.
- a template of *metadata.yaml* .
- a template of *README.md* file.
- a generic *model_conf.xml* file.
- a LICENSE file if *–license* option is set.
- a COPYRIGHT file if *–holders* option is set.

- a directory *definitions* with an example of model definition (model_example.xml to remove before publishing).
- a directory *profiles* where to put the hmm profiles corresponding to the models genes.

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

README.md

A description of the package: what kind of systems the package models, how to use it etc… in markdown format. The Readme is displayed to the user on the macsy-models repository on Github. It is also displayed when the user runs *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 CreativeCommons *This file is optional, but highly recommended.*

Metadata file

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

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

```
---
maintainer:
  name: The name of the person who maintains/to contact for further information.␣
↪(required)
  email: The email of the maintainer (required)
short_desc: A one line description of the package (can e.g. be used for *macsydata*␣
↪searches) (required)
vers: The package version (DEPRECATED)
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).
cite:
   - |
     Abby Sophie S., Cury Jean, Guglielmini Julien, Néron Bertrand, Touchon Marie, Rocha␣
```

(continues on next page)

(continued from previous page)

```
↪Eduardo P. C. (2016).
    Identification of protein secretion systems in bacterial genomes.
    In Scientific Reports, 6, pp. 23080.
    http://dx.doi.org/10.1038/srep23080
doc: https://github.com/macsy-models/TXSS
license: CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0/)
copyright: 2014-2022, Institut Pasteur, CNRS
```

Note:

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

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

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

Model configuration

The modeler has the possibility to specify some options that are specific to their package, different than the 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_evalue_sel (*float* default = 0.001)
- profile_coverage (*float* default = 0.5)
- cut_ga (*bool* default = True)

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

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

- **system**: the *macsyfinder.conf* file either in /etc/macsyfinder/ or in ${VIRTUAL_ENV}/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>
        <out_of_cluster>0.7</out_of_cluster>
    </weights>
    <filtering>
        <e_value_search>0.1</e_value_search>
        <i_evalue_sel>0.01</i_evalue_sel>
        <coverage_profile>0.5</coverage_profile>
        <cut_ga>True</cut_ga>
    </filtering>
</model_config>
```

Details about the scoring method can be obtained here.

Macromolecular models

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

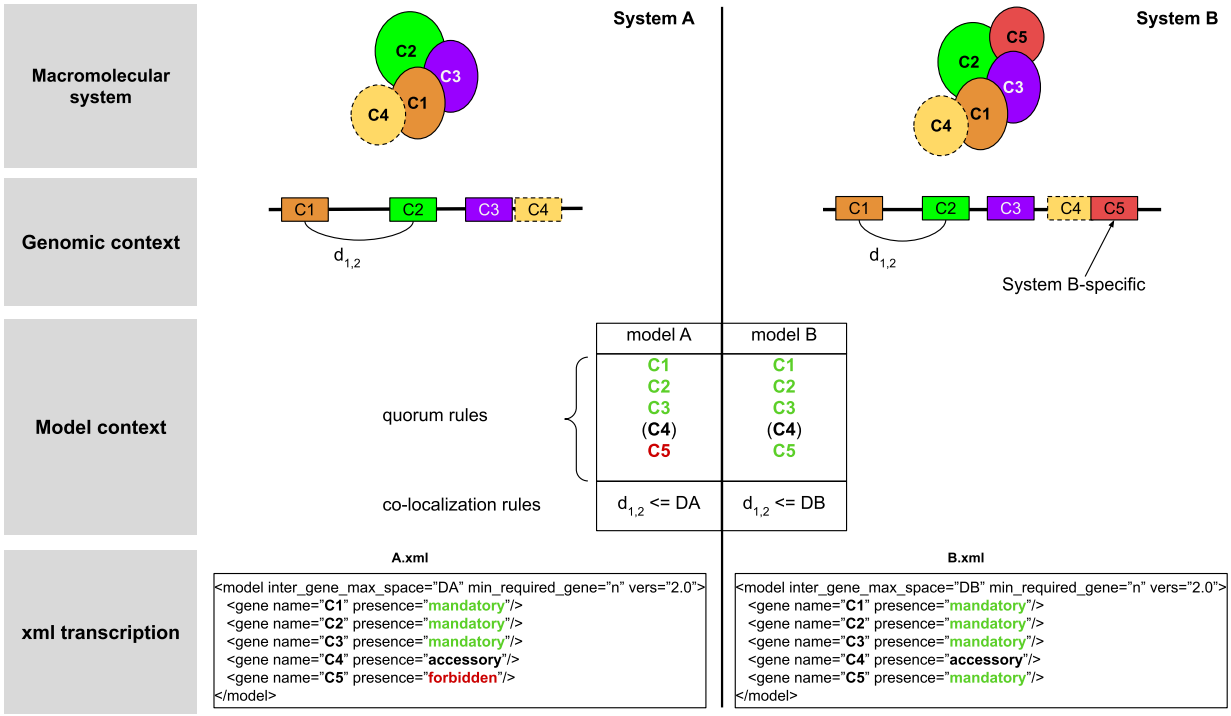
Beyond its list of Components, a MacSyFinder's model of a System is defined by the genomic organization of its components. This genomic organization can be defined in several ways:

- 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 wholeness is false.
  - The model contains one or more element(s) "gene" that correspond(s) to the genetic components of the macromolecular system.
- The element "gene" has several mandatory attributes:
  - **name**: a *string* representing the name of the component/gene which must match that of a profile enclosed in the profile directory of the macsy-model package (see *below*).
  - **presence**: a *string* representing the status of the gene's presence in the system. It can take four values among "mandatory", "accessory", "neutral", "forbidden" (see above).

The element "gene" may have other attributes:

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

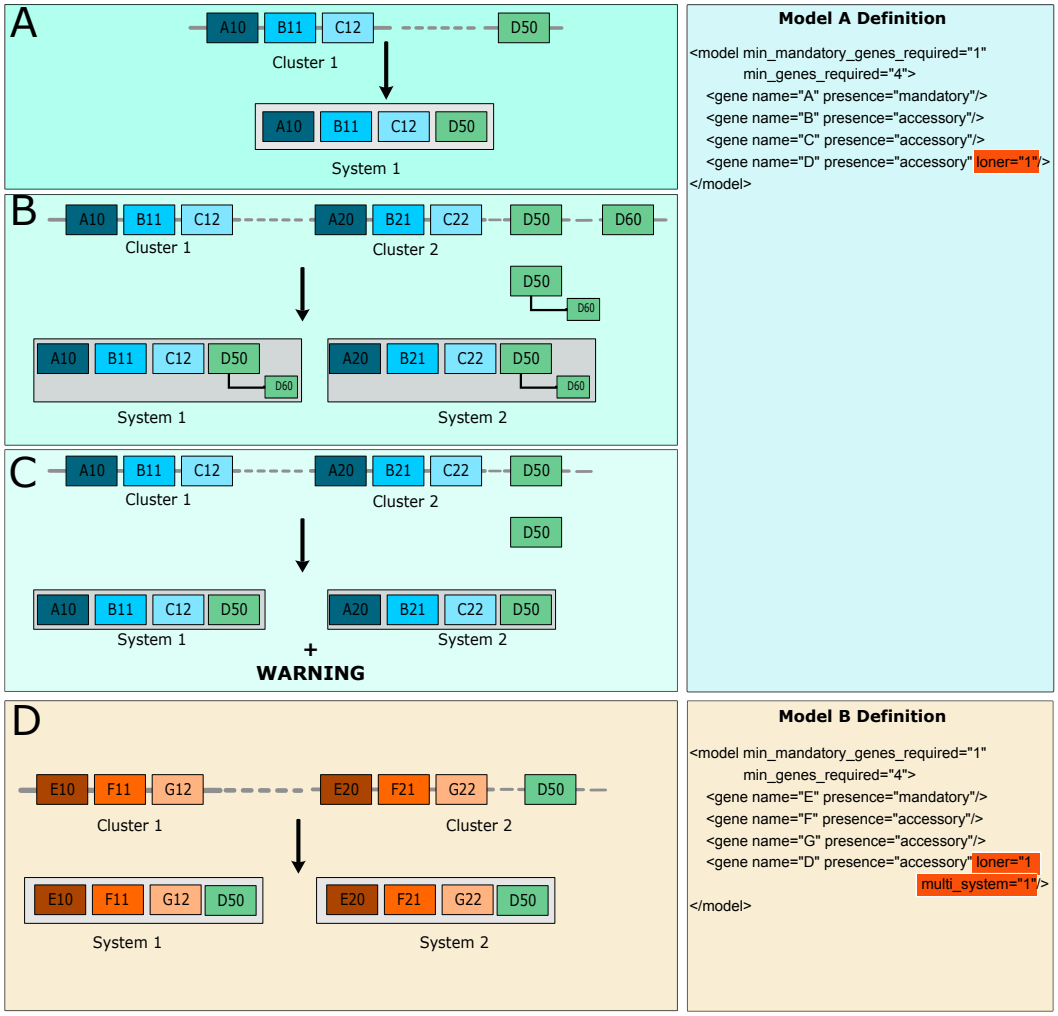


Fig. 1: How *loner* works.

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

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

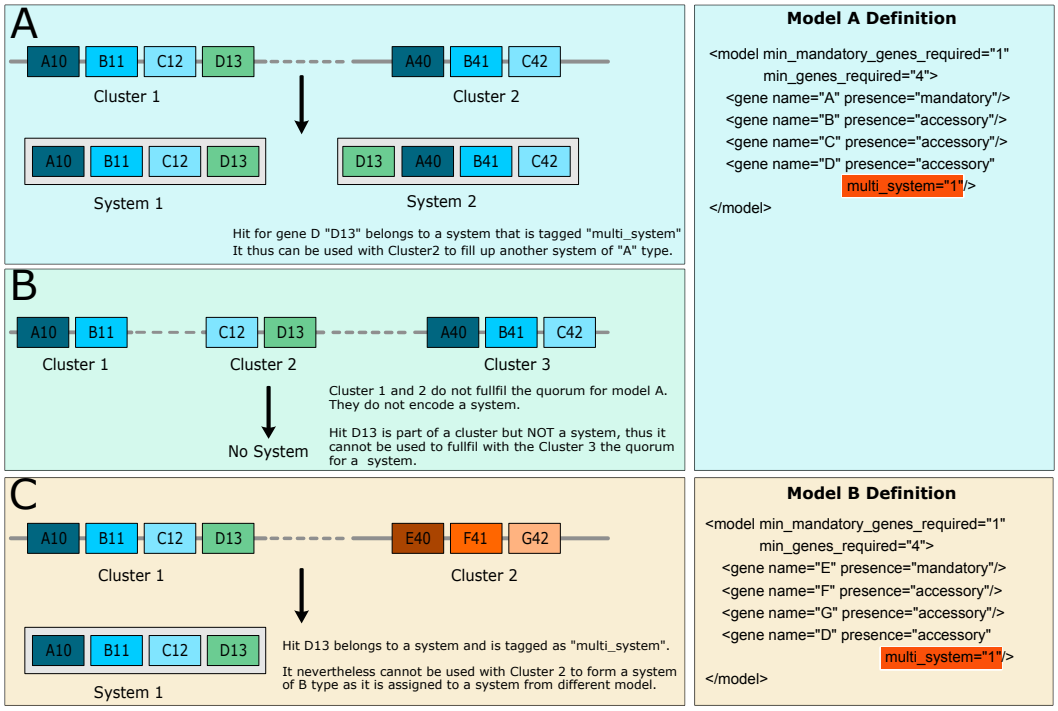


Fig. 2: How *multi_system* works.
A) The hit encoding for gene D in position 13 belongs to the system 1 (encoding model A). So it is used to fill up some other cluster, for instance cluster 2, which lacks this functionality. The cluster 2 then also fulfil the requirement of a system. **B**) The hit encoding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

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

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

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

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

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

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

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

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

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

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

In the snippet code above,

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

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

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

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

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

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

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

```
<model inter_gene_max_space="5" vers="2.0">
  <gene name="gspD" presence="mandatory">
     <exchangeables>
         <gene name="sctC"/>
     </exchangeables>
  </gene>
  <gene name="sctN_FLG" presence="mandatory" loner="1">
```

(continues on next page)

(continued from previous page)

```
    <exchangeables>
        <gene name="gspE"/>
        <gene name="pilT"/>
    </exchangeables>
  </gene>
  <gene name="sctV_FLG" presence="mandatory"/>
  <gene name="flp" presence="accessory"/>
</model>
```

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

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

Warning:

- a gene is identified by its name.
- this name is case sensitive.
- this name must be unique inside a family of models.
- a HMM profile with a gene-based name must exist in the *profiles* directory of the macsy-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 macsy-model would correspond by default to a HMM profile "GeneA.hmm" enclosed in the macsy-model package. The names are **case-sensitive**. All HMM profiles must be placed in the *profiles* directory of the macsy-model package.

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

Helper Tool

macsyprofile

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

It is ran over a previous macsyfinder 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: macsyprofile [-h] [--coverage-profile COVERAGE_PROFILE]
                    [--i-evalue-sel I_EVALUE_SEL]
                    [--best-hits {score,i_eval,profile_coverage}] [-p PATTERN]
                    [-o OUT] [-f] [-V] [-v] [--mute]
                    previous_run

     *            *               *                    * *       *
*           *               *   *   *  *    **                *
  **     *    *   *  *     *                    *               *
     __  __     *        ____ *       ____         **   __ _ _  *
    |  \/  | __ _  ___/ ___| _   _|  _ \ _ __ ___  / _(_) | ___
    | |\/| |/ _` |/ __\___ \| | | | |_) | '__/ _ \| |_| | |/ _ \
    | |  | | (_| | (__ ___) | |_| |  __/| | | (_) |  _| | |  __/
    |_|  |_|\__,_|\___|____/ \__, |_|   |_|  \___/|_| |_|_|\___|
            *                |___/    *                  *
 *      *   * *     *   **          *   *  *           *
  *      *         *        *    *              *
            *                            *  *           *     *

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_EVALUE_SEL
                        Maximal independent e-value for Hmmer hits to be
                        selected for systems detection. (default: no selection
                        based on i-evalue)
  --best-hits {score,i_eval,profile_coverage}
                        If several hits match the same replicon, same gene.
                        Select only the best one (based on best 'score' or
                        'i_evalue' or 'profile_coverage')
  -p PATTERN, --pattern PATTERN
                        pattern to filter the hmm files to analyse.
```

(continues on next page)

(continued from previous page)

```
  -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
```

(continues on next page)

(continued from previous page)

```
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:

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

A useful example for modellers?

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

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

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

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

Parsing macsyprofile outputs

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-20201202_15-17-46/
hit_id  replicon_name   position_hit    hit_sequence_length     gene_name       i_eval␣
↪score   profile_coverage        sequence_coverage       begin   end
GCF_000006745_021980    GCF_000006745   2198    291     ComM_comC       2.500e-40       ␣
↪136.400 0.942   0.708   62      267
GCF_000006745_007650    GCF_000006745   765     253     ComM_comC       9.600e-31       ␣
↪105.100 0.937   0.798   43      244
...
```

Note: This file can be easily parsed using the Python pandas library.

```
import pandas as pd

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

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

Publishing/sharing models

Writing your own macsy-model package

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

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

- HMM profiles within a *profiles* folder (mandatory)

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

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

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 to take to publish the package
   - warning: it means that the package could be improved.

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

---

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

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

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

```
#!/bin/sh

# An example hook script to verify what is about to be pushed.  Called by "git
# push" after it has checked the remote status, but before anything has been
# pushed.  If this script exits with a non-zero status nothing will be pushed.
```

```
#
# This hook is called with the following parameters:
#
# $1 -- Name of the remote to which the push is being done
# $2 -- URL to which the push is being done
#
# If pushing without using a named remote those arguments will be equal.
#
# Information about the commits which are being pushed is supplied as lines to
# the standard input in the form:
#
#   <local ref> <local oid> <remote ref> <remote oid>
#
# This script check if you push a tag
# if yes check if the tag match to the version decalred in metadata.yml
# if yes it prevents the push until the tag and the version match
#
# This script is widely inspired from https://gist.github.com/farseerfc/
↪0729c08cd7c82b07000f20105f733b17

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

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

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

macsydata check
returncode=$?

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

exit $returncode
```

pre-push .

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

through the *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
```

(continues on next page)

(continued from previous page)

```
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

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 litterature, 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.

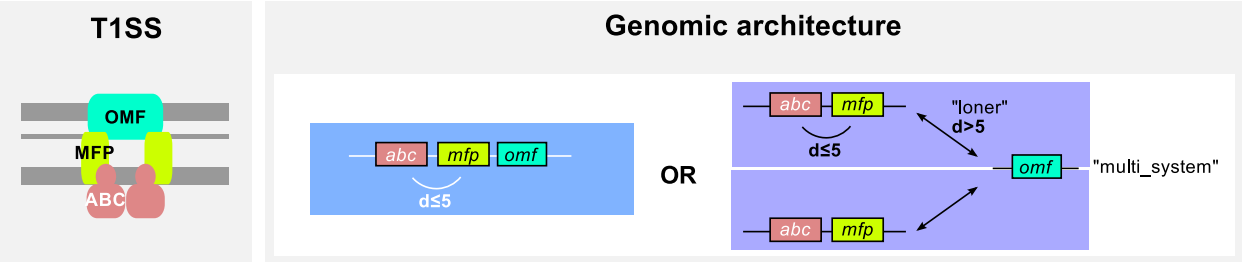
3. Describing their genetic architecture

According to the litteraure, 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.

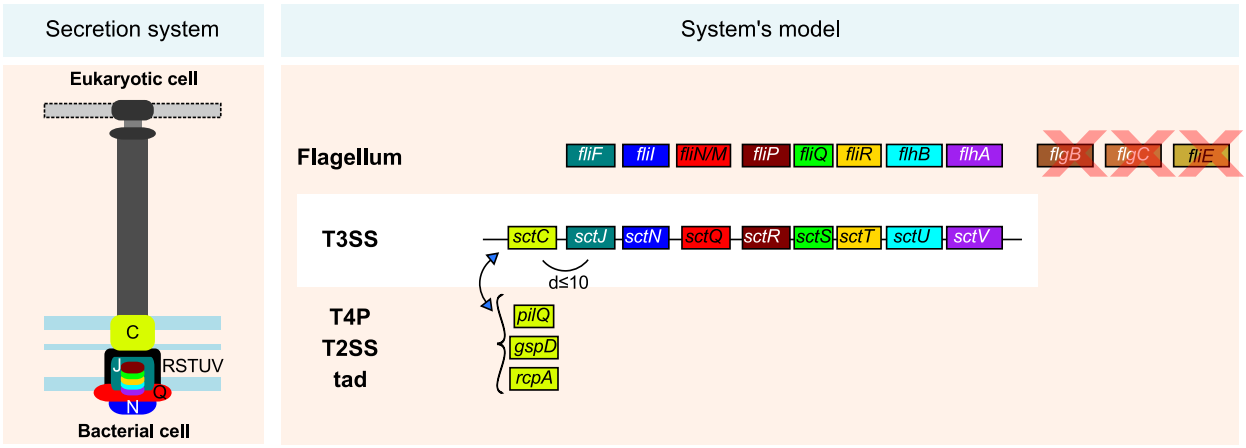
2. Dealing with components with varied evolutionary origins

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 aroound 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>
```

(continues on next page)

(continued from previous page)

```
          <gene name="T2SS_gspD"/>
          <gene name="T4P_pilQ"/>
          <gene name="Tad_rcpA"/>
        </exchangeables>
    </gene>
    <gene name="T3SS_sctJ" presence="mandatory"/>
    <gene name="T3SS_sctN" presence="mandatory"/>
    <gene name="T3SS_sctQ" presence="mandatory"/>
    <gene name="T3SS_sctR" presence="mandatory"/>
    <gene name="T3SS_sctS" presence="mandatory"/>
    <gene name="T3SS_sctT" presence="mandatory"/>
    <gene name="T3SS_sctU" presence="mandatory"/>
    <gene name="T3SS_sctV" presence="mandatory"/>
    <gene name="Flg_fliE" presence="forbidden"/>
    <gene name="Flg_flgB" presence="forbidden"/>
    <gene name="Flg_flgC" presence="forbidden"/>
</model>
```

And the model of the Flagellum:

Flagellum.xml

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

2.1.2 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.

T1SS.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>
```

T1SS.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"/>
```

(continues on next page)

(continued from previous page)

```
                <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>
```

(continues on next page)

(continued from previous page)

```
            <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>
            <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"/>
```

(continues on next page)

(continued from previous page)

```
        </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.

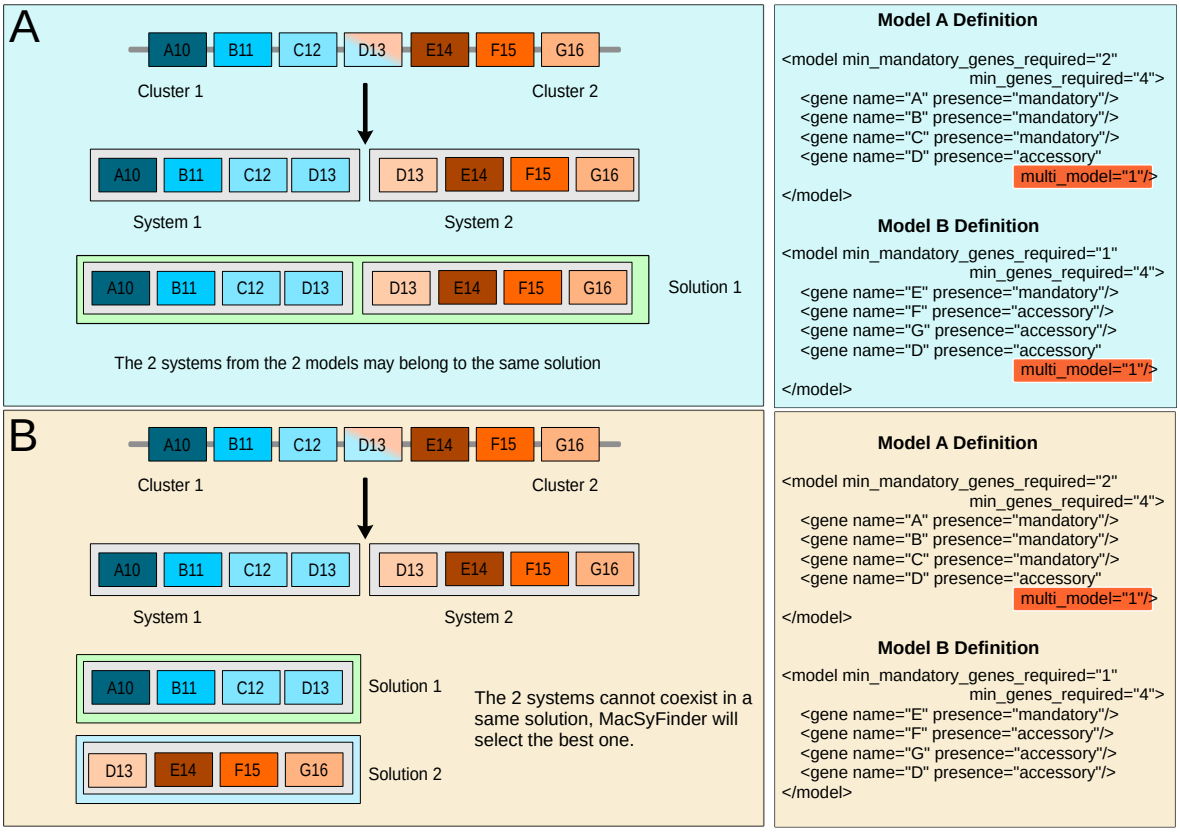


Fig. 3: How *multi_model* works.
The hit encoding for gene D in position 13 is part of 2 systems: one for Model A, one for Model B. **A**) In both model definitions the gene D is tagged as multi_model. So the 2 systems can coexist in a same solution (they are "compatible"). **B**) The gene D is tagged as multi_model **only** in model A definition. The 2 systems are not compatible. So *msf* build 2 solutions and choose the best one. It has to be noted that this behaviour would actually be the same if gene D was not declared multi_model in either definitions.

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Developer Guide

3.1.1 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. We also provide container so you can use macsyfinder directly.

MacSyFinder dependencies

Python version >=3.10 is required to run MacSyFinder: https://docs.python.org/3.10/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 some Python library dependencies:

- colorlog
- colorama
- pyyaml
- packaging
- networkx
- pandas
- GitPython
- sphinx
- sphinx_rtd_theme
- sphinx-autodoc-typehints
- sphinxcontrib-svg2pdfconverter
- coverage
- build
- ruff
- pre-commit

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

MacSyFinder Installation procedure

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.

installation in a virtualenv

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

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

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

Then run *macsyfinder* or *macsydata*.

Note: from 2.1.4 version, *MacSyFinder* has adopted ruff as linter and *pre-commit* to ensure the coding style. please read CONTRIBUTING.md guide lines.

3.1.2 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. 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*

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.

setup.py
The installation recipe.

setup.cfg
The installation recipe.

pyproject.toml
tools to use to build the project.

MacSyFinder architecture overview

An overview of the main classes.

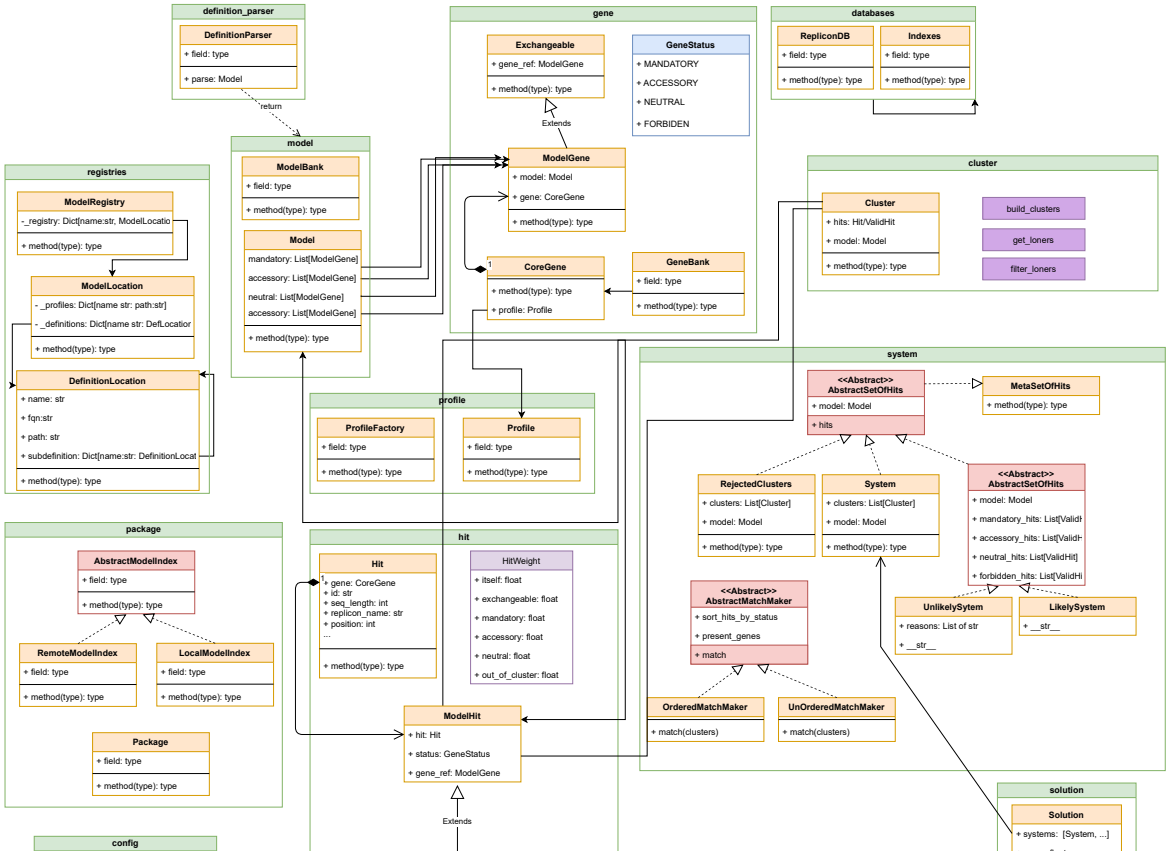


Fig. 1: The macsyfinder classes diagram. The classes are not details. only the main attributes allowing us to understand the interaction are mentioned.

- 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: 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.RejectedCandidate`.

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

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

Among all this potential systems, 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*).

3.1.3 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.

configuration API reference

MacsyDefaults

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*) → None

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

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

__weakref__

list of weak references to the object

Config

class macsypy.config.**Config**(*defaults:* MacsyDefaults, *parsed_args: Namespace*)

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

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

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

__weakref__

list of weak references to the object

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

Parse a configuration file in ini format in dictionnary

Parameters
file – path to the configuration file

Returns
the parsed options

_set_command_line_config(*parsed_args: Namespace*) → None

Parameters
parsed_args – the argument set on the command line

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

set value for 'db_type' option

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

Raises
ValueError – if value is not allowed

_set_default_config() → None

set the value comming from MacsyDefaults

_set_inter_gene_max_space(*value: str*) → None

set value for 'inter_gene_max_space' option

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

Raises
ValueError – if value is not well-formed

_set_log_level(*value: str*) → None

Parameters
value –

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

set value for 'max_nb_genes' option

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

Raises
ValueError – if value is not well-formed

_set_min_genes_required(*value: str*) → None

set value for 'min_genes_required' option

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

Raises
ValueError – if value is not well-formed

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

set value for 'min_mandatory_genes_required' option

Parameters

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

Raises

ValueError – if value is not well-formed

_set_model_config(*model_conf_path: str*) → None

Set the options from the model package model_conf.xml file

Parameters

model_conf_path – The path to the model_conf.xml file

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

Parameters

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

if value come from command_line

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

if value come from config file

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

_set_models_dir(*path: str*) → None

Parameters

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

_set_multi_loci(*value: str*) → None

Parameters

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

_set_no_cut_ga(*value*) → None

Parameters

value –

Returns

Return type

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

set key, value in the general config

Parameters

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

_set_previous_run_config(*prev_config_path: str*) → None

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

Parameters

prev_config_path –

_set_project_config_file(*config_path: str*) → None

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

Parameters
config_path – the path to the configuration file

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

set the default replicon topology

Parameters
value – 'circular' or 'linear'

_set_sequence_db(*path: str*) → None

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

_set_system_models_dir(*value*)

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

Returns

_set_system_wide_config(*config_path: str*) → None

set the options from the system-wide configuration file

Parameters
config_path –

_set_topology_file(*path: str*) → None

test if the path exists and set it in config

Parameters
path – the path to the topology file

_set_user_config_file(*config_path: str*) → None

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

Parameters
config_path – The path to the configuration path

_set_user_wide_config(*config_path: str*) → None

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

Parameters
config_path – The path to the ~/.macsyfinder/macsyfinder.conf

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

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

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

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

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

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

Return type
dict

hmmer_dir() → str

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

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

Parameters
model_fqn – the model fully qualifed name

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

log_level() → int

Returns
the verbosity output level

Return type
int

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

Parameters
model_fqn – the model fully qualifed name

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

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

Parameters
model_fqn – the model fully qualifed name

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

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

Parameters
model_fqn – the model fully qualifed name

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

models_dir() → str | None

Returns
list of models dir path

Return type
list of str

multi_loci(*model_fqn: str*) → bool

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

Returns
True if the model is multi loci, False otherwise

Return type
bool

`out_dir`() → str

Returns
the path to the directory where the results are stored

`save`(*path_or_buf: str | TextIO | None = None*) → None
save itself in a file in ini format.

Note: the undefined options (set to None) are omitted

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

`working_dir`() → str
alias to `config.Config.out_dir()`

NoneConfig

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

__weakref__
list of weak references to the object

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

model_conf_parser API reference

ModelConfParser

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

`__init__`(*path: str*) → None

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

__weakref__
list of weak references to the object

_get_model_conf_node() → ElementTree

Find the root of the document

Returns
the document root of model_conf

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

Parse a node containing configurations options and value

Parameters

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

Returns
dict

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

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

Returns
The specific configuration for a model family

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

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

Parse the node 'filtering' containing the filtering options configuration

Parameters
filtering_node – the node 'filtering'

Returns
the configuration option/value about the filtering

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

Parse the node 'weights' containing the scoring weight configuration

Parameters
weights_node – the node 'weights'

Returns
the configuration option/value about the scores

registries

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

registries API reference

ModelRegistry

class macsypy.registries.**ModelRegistry**

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

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

Parameters
name –

Returns
the model corresponding to name.

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

__init__() → None

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

add(*model_loc:* ModelLocation) → None

Parameters
model_loc – the model location to add to the registry

models() → list[*macsypy.registries.ModelLocation*]

Returns
the list of models

ModelLocation

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

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

__eq__(*other:* ModelLocation) → bool

Return self==value.

__gt__(*other:* ModelLocation) → bool

Return self>value.

__hash__ = None

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

Parameters

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

__lt__(*other:* ModelLocation) → bool

Return self<value.

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

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

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

Parameters

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

Returns

a definition location

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

Store all hmm profiles associated to the model

Parameters

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

Returns

all profiles found in the path

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

Name root_def_name

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

Returns

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

Raises

ValueError – if root_def_name does not match with any definitions

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

Parameters

fqn – the fully qualified name of the definition to retrieve. it's complete path without extension. for instance for a file with path like this: models/TXSS/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.

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

get_definitions() → list[*macsypy.registries.DefinitionLocation*]

Returns
the list of the definitions of this modelLocation. It returns the 1rst level only (not recursive). For recursive explorations see *macsypy.registries.ModelLocation.get_all_definitions()*

get_profile(*name: str*) → str

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

Returns
the absolute path of the hmm profile.

Raise
KeyError if name does not match with any profiles.

get_profiles_names() → list[str]

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

property version: str

Returns
The version of the models

MetaDefLoc

class macsypy.registries.**MetaDefLoc**

DefinitionLocation

class macsypy.registries.**DefinitionLocation**(*name: str | None = None*, *fqn: str | None = None*, *subdefinitions:* macsypy.registries.DefinitionLocation | *None = None*, *path: str | None = None*)

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

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

__eq__(*other:* DefinitionLocation) → bool

Return self==value.

__gt__(*other:* DefinitionLocation) → bool

Return self>value.

__hash__() → int

Return hash(self).

__init__(*name: str | None = None*, *fqn: str | None = None*, *subdefinitions:* macsypy.registries.DefinitionLocation | *None = None*, *path: str | None = None*) → None

__lt__(*other:* DefinitionLocation) → bool

Return self<value.

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

add_subdefinition(*subdefinition:* DefinitionLocation) → None

add new sub category of definitions to this definition

Parameters

subdefinition – the new definition to add as subdefinition.

all() → list[*macsypy.registries.DefinitionLocation*]

Returns

the definition and all recursively all subdefinitions

property family_name: str

Returns

the models family name which is the name of the package

classmethod root_name(*fqn: str*) → str

Parameters

fqn (*str*) – the fully qualified name of a definition

Returns

the root name of this definition (family name)

classmethod split_fqn(*fqn: str*) → list[str]

Parameters

fqn – the fully qualified name of a definition

Returns

each member of the fully qn in list.

split_def_name

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

Parameters

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

Returns

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

join_def_path

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

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

scan_models_dir

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

Parameters

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

Returns
the list of models in models_dir

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

definition_parser

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

Phase 1.

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

Phase 2.

- For each model to search
  - For each Gene defined in this System:
    - link the gene to the model. Create a ModelGene by encapsulating CoreGene from the gene_bank It can exists at each run several ModelGene for one CoreGene
    - If a gene has exhangeables create them (an Exchangeable inherits from ModeleGene) and add them to the current ModelGene

For instance:

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

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

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

With the example above:

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

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

defintion_parser API reference

DefinitionParser

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

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

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

__init__(*cfg:* macsypy.config.Config | macsypy.config.NoneConfig, *model_bank:* ModelBank, *gene_bank:* GeneBank, *model_registry:* ModelRegistry, *profile_factory:* ProfileFactory) → None

Parameters

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

__weakref__

list of weak references to the object

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

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

Parameters

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

Raises

ModelInconsistencyError – if an error is encountered in the document.

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

Parameters

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

Returns

the model corresponding to the definition location.

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

find all gene node and add them to the gene_bank

Parameters

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

_get_model_node(*def_loc:* DefinitionLocation) → ElementTree

Parameters

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

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

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

Parameters

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

Returns
the gene object corresponding to the node

_parse_genes(*model:* Model, *model_node: ElementTree*) → None

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

Parameters

- **model** – the Model currently parsing
- **model_node** – the element ‘model’

check_consistency(*models: list[*macsypy.model.Model*]*) → None

Check the consistency of the co-localization features between the different values given as an input: between XML definitions, configuration file, and command-line options.

Parameters
models – the list of models to check

Raise
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: list[*macsypy.registries.DefinitionLocation*]*) → None

Parse models definition in XML format to build the corresponding Model objects, and add them to the model factory after checking its consistency. To get the model ask it to model_bank

Parameters
models_2_detect – a list of model definition to parse.

model

The model is a formal representation of system. The model is describe in terms of components. There are 4 component classes:

- genes which are mandatory
- genes which are accessory
- genes which are neutral
- genes which are forbiden

Each genes can have Exchangeable. An exchangeable is another gene which can paly the same role in the system. Usualy an analog or homolog. The models describe also distance constraints between genes:

- inter_gene_max_space
- loner
- multi_loci

and quorum constraints

- min_mandatory_genes_required
- min_genes_required

and if a gene can be shared by several systems (several occurrences of the same model)

- multisystem

model API reference

ModelBank

class macsypy.model.**ModelBank**

Store all Models objects.

__contains__(*model:* Model) → bool

Implement the membership test operator

Parameters
model – the model to test
Returns
True if the model is in the Model factory, False otherwise

__getitem__(*fqn: str*) → *Model*

Parameters
fqn – the fully qualified name of the model
Returns
the model corresponding to the fqn.
Raises
KeyError – if the model corresponding to the name does not exist

__init__() → None

__iter__() → Iterator

Returns
an iterator object on the models contained in the bank

__len__() → int

Returns
the number of models stored in the bank

__weakref__

list of weak references to the object

add_model(*model:* Model) → None

Parameters
model – the model to add

Raise
KeyError if a model with the same name is already registered.

Model

class macsypy.model.**Model**(**args*, ***kwargs*)

Handles a macromolecular model.

Contains all its pre-defined characteristics expected to be fulfilled to predict a complete model:

- component list (genes that are mandatory, accessory, neutral, forbidden)
- quorum (number of genes)
- genetic architecture

__eq__(*other:* Model) → bool

Parameters
other – the other model to compare

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

__gt__(*other:* Model) → bool

Parameters
other – the other model to compare

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

__hash__() → int

Returns

__init__(*fqn: str*, *inter_gene_max_space: int*, *min_mandatory_genes_required: int = None*, *min_genes_required: int = None*, *max_nb_genes: int = None*, *multi_loci: bool = False*) → None

Parameters

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

- **min_genes_required** – the quorum of genes to define this model
- **max_nb_genes** – The number of gene to be considered as full system Used to compute the wholeness. If None the mx_nb_genes = mandatory + accessory
- **multi_loci** – if the systems can split in different loci on the genome

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

__lt__(*other:* Model) → bool

Parameters
other – the other model to compare

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

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

property family_name: str

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

filter(*hits: list[*macsypy.hit.CoreHit*]*) → list[*macsypy.hit.ModelHit*]

filter out the hits according to this model and cast them in ModelHit. The filtering is based on the name of CoreGene associated to hit and the name of ModelGene of the model (the name of the ModelGene is the name of the CoreGene embedded in the ModelGene) only the hits related to genes implied in the model are kept.

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

Returns
list of hits

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

genes(*exchangeable: bool = False*) → set[*macsypy.gene.ModelGene*]

Parameters
exchangeable – include exchangeable if True

Returns
all the genes described in the model. with exchangeables if exchangeable is True. otherwise only "first level" genes.

get_gene(*gene_name: str*) → *ModelGene*

Parameters
gene_name – the name of the gene to get

Returns
the gene corresponding to gene_name.

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

property inter_gene_max_space: int

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

property max_nb_genes: int

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

property min_genes_required: int

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

Return type
integer

property min_mandatory_genes_required: int

Returns
get the quorum of mandatory genes required for this model

property multi_loci: bool

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

property name: str

Returns
the short name of this model

gene

The *Gene object* represents genes encoding the protein components of a Model. There is 2 kind of gene The `CoreGene` (*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.defintion_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_stn_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.

gene API reference

GeneBank

class macsypy.gene.**GeneBank**

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

__contains__(*gene:* CoreGene) → bool

Implement the membership test operator

Parameters
gene – the gene to test

Returns
True if the gene is in, False otherwise

Return type
boolean

__getitem__(*key: tuple[str, str]*) → *CoreGene*

Parameters
key – The key to retrieve a gene. The key is composed of the name of models family and the gene name. for instance CRISPR-Cas/cas9_TypeIIB ('CRISPR-Cas' , 'cas9_TypeIIB') or TXSS/T6SS_tssH ('TXSS', 'T6SS_tssH')

Returns
return the Gene corresponding to the key.

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

__init__() → None

__iter__() → Iterator

Return an iterator object on the genes contained in the bank

__weakref__

list of weak references to the object

add_new_gene(*model_location:* ModelLocation, *name: str*, *profile_factory:* ProfileFactory) → None

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

Parameters

- **model_location** – the location where the model family can be found.
- **name** – the name of the gene to add
- **profile_factory** – The Profile factory

genes_fqn() → list[str]

Returns

the fully qualified name for all genes in the bank

Gene

There is two classes to modelize a gene: *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 differents properties, *loner* in one model and not in an other, have different *inter_gene_max_space* … The ModelGene is attached to the model and is composed of a CoreGene.

Note: The `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:* ModelLocation, *name: str*, *profile_factory:* ProfileFactory)

Modeling gene attached to a profile. It can be only one instance with the same name (familly name, gene name)

__hash__() → int

Return hash(self).

__init__(*model_location:* ModelLocation, *name: str*, *profile_factory:* ProfileFactory) → None

__weakref__

list of weak references to the object

property model_family_name: str

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

property name: str

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

property profile: *Profile*

The HMM protein Profile corresponding to this gene

ModelGene

class macsypy.gene.**ModelGene**(*gene: CoreGene*, *model: Model*, *loner: bool = False*, *multi_system: bool = False*, *inter_gene_max_space: int = None*, *multi_model: bool = False*)

Handle Gene described in a Model

__hash__() → int

Return hash(self).

__init__(*gene: CoreGene*, *model: Model*, *loner: bool = False*, *multi_system: bool = False*, *inter_gene_max_space: int = None*, *multi_model: bool = False*)

Handle gene described in a Model

Parameters

- **gene** – a gene link to a profile
- **model** – the model that owns this Gene
- **loner** – True if the Gene can be isolated on the genome (with no contiguous genes), False otherwise.
- **multi_system** – True if this Gene can belong to different occurrences of this System.
- **inter_gene_max_space** – the maximum space between this Gene and another gene of the System.
- **multi_model** – True if this Gene is allowing to appear in several system occurrence from different model.

__str__() → str

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

__weakref__

list of weak references to the object

add_exchangeable(*exchangeable: Exchangeable*)

Add an exchangeable gene to this Gene

Parameters

exchangeable – the exchangeable to add

alternate_of() → *ModelGene*

Returns

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

property **core_gene**: *CoreGene*

Returns

The CoreGene associated to this ModelGene

property **exchangeables**: **list**[*macsypy.gene.ModelGene*]

Returns

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

property inter_gene_max_space: int | None

Returns
The maximum distance allowed between this gene and another gene for them to be considered co-localized. If the value is not set at the Gene level, return None.

is_accessory(*model:* Model) → bool

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

Parameters
model – the query of the test

property is_exchangeable: bool

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

is_forbidden(*model:* Model) → bool

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

Parameters
model – the query of the test

is_mandatory(*model:* Model) → bool

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

Parameters
model – the query of the test

property loner: bool

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

property model: *Model*

Returns
the Model that owns this Gene

property multi_model: bool

Returns

True if this Gene can belong to different occurrences of systems from different
model *macsypy.model.Model*

(and can be used for multiple System assessments), False otherwise.

Return type
boolean.

property multi_system: bool

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

set_status(*status:* GeneStatus) → None

Set the status for this gene

Parameters
status – the status of this gene

property status: *GeneStatus*

Returns
The status of this gene

Exchangeable

class macsypy.gene.**Exchangeable**(*c_gene:* CoreGene, *gene_ref:* ModelGene, *loner: bool | None = None*, *multi_system: bool | None = None*, *multi_model: bool | None = None*, *inter_gene_max_space: int | None = None*)

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

__init__(*c_gene:* CoreGene, *gene_ref:* ModelGene, *loner: bool | None = None*, *multi_system: bool | None = None*, *multi_model: bool | None = None*, *inter_gene_max_space: int | None = None*) → None

Parameters

- **c_gene** – the gene
- **gene_ref** – the gene to which the current can replace it.

add_exchangeable(*exchangeable:* Exchangeable) → None

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

Parameters
exchangeable – the exchangeable gene to add

Raises
MacsypyError –

alternate_of() → *ModelGene*

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

property is_exchangeable: **bool**

Returns
True

property status: *GeneStatus*

Returns
The status of this gene. if the status is not define for this gene itself, return the status of the reference gene.

GeneStatus

class macsypy.gene.**GeneStatus**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

profile API reference

ProfileFactory

class macsypy.profile.**ProfileFactory**(*cfg:* macsypy.config.Config | macsypy.config.NoneConfig)

Build and store all Profile objects. Profiles must not be instanciated 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:* macsypy.config.Config | macsypy.config.NoneConfig) → None

__weakref__

list of weak references to the object

get_profile(*gene:* CoreGene, *model_location:* ModelLocation) → *Profile*

Parameters

- **gene** – the gene associated to this profile
- **model_location** – The where to get the profile

Returns

The profile corresponding to the name. If the profile already exists, return it. Otherwise, build it, store it and return it.

Profile

class macsypy.profile.**Profile**(*gene:* CoreGene, *cfg:* Config, *path: str*)

Handle a HMM protein profile

__init__(*gene:* CoreGene, *cfg:* Config, *path: str*) → None

Parameters

- **gene** – the gene corresponding to this profile
- **cfg** – the configuration
- **path** – the path to the hmm profile.

__len__() → int

Returns
the length of the HMM protein profile

Return type
int

__str__() → str

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

__weakref__

list of weak references to the object

_profile_features() → tuple[int, bool]

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

Returns
the length, presence of ga bit threshold

execute(*cpu: int = 1*) → *macsypy.report.HMMReport* | None

Launch the Hmmer search (hmmsearch executable) with this profile

Parameters
cpu – the number of cpu to use for hmmsearch (must be >= 1)

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

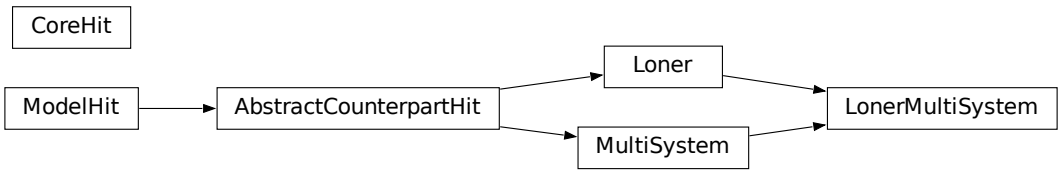
hit

This module implements class relative to hit and some functions to do some computation on hit objects.

macsypy.hit.CoreHit	Modelize a hmm hit on the replicon. There is only one Corehit for a CoreGene.
macsypy.hit.ModelHit	Modelize a hit and its relation to the Model.
macsypy.hit.AbstractCounterpartHit	Parent class of Loner, MultiSystem. It's inherits from ModelHit.
macsypy.hit.Loner	Modelize "true" Loner.
macsypy.hit.MultiSystem	Modelize hit which can be used in several Systems (same model)
macsypy.hit.LonerMultiSystem	Modelize a hit representing a gene Loner and MultiSystem at same time.
macsypy.hit.HitWeight	The weights apply to the hit to compute score
macsypy.hit.get_best_hit_4_func()	Return the best hit for a given function
macsypy.hit.sort_model_hits()	Sort hits
macsypy.hit.compute_best_MSHit()	Choose among svereal multisystem hits the best one
macsypy.hit.get_best_hits()	If several profile hit the same gene return the best hit

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

Below the ingheritance diagram of Hits



And a diagram showing the interaction between CoreGene, ModelGene, Model, Hit, Loner, … interactions

hit API reference

CoreHit

class macsypy.hit.**CoreHit**(*gene:* CoreGene, *hit_id: str*, *hit_seq_length: int*, *replicon_name: str*, *position_hit: int*, *i_eval: float*, *score: float*, *profile_coverage: float*, *sequence_coverage: float*, *begin_match: int*, *end_match: int*)

Handle the hits filtered from the Hmmer search. The hits are instanced by `HMMReport.extract()` method In one run of MacSyFinder, there exists only one CoreHit per gene These hits are independent of any *macsypy.model.Model* instance.

__eq__(*other:* CoreHit) → bool

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

Parameters

other – the hit to compare to the current object

Returns

the result of the comparison

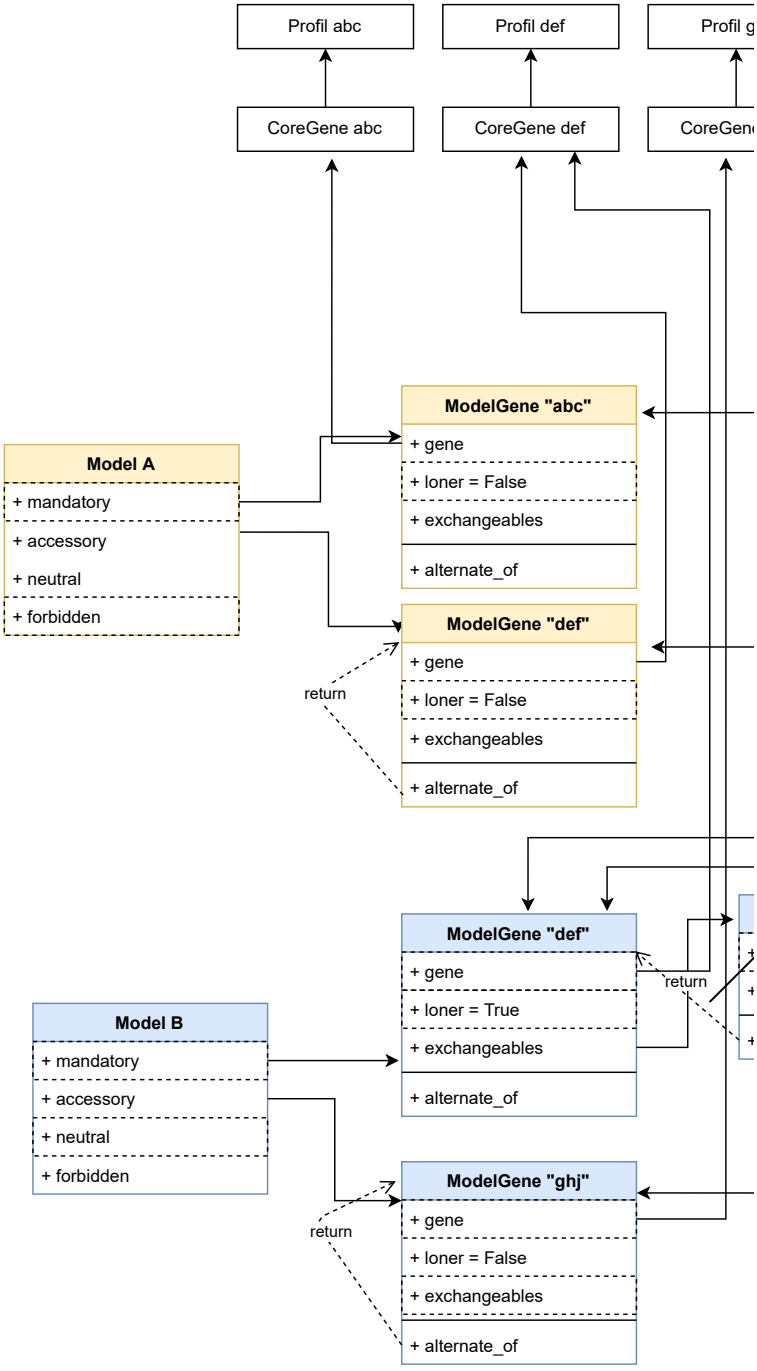


Fig. 2: The diagram above represents the models, genes and hit generated from the definitions below.

```
<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"/>
```

__gt__(*other:* CoreHit) → bool

compare two Hits. If the sequence identifier is the same, do the comparison on the score. Otherwise, do it on alphabetical comparison of the sequence identifier.

Parameters
other – the hit to compare to the current object

Returns
True if self is > other, False otherwise

__hash__() → int

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

__init__(*gene:* CoreGene, *hit_id: str*, *hit_seq_length: int*, *replicon_name: str*, *position_hit: int*, *i_eval: float*, *score: float*, *profile_coverage: float*, *sequence_coverage: float*, *begin_match: int*, *end_match: int*) → None

Parameters

- **gene** – the gene corresponding to this profile
- **hit_id** – the identifier of the hit
- **hit_seq_length** – the length of the hit sequence
- **replicon_name** – the name of the replicon
- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_eval** – the best-domain evalue (i-evalue, "independent evalue")
- **score** – the score of the hit
- **profile_coverage** – percentage of the profile that matches the hit sequence
- **sequence_coverage** – percentage of the hit sequence that matches the profile
- **begin_match** – where the hit with the profile starts in the sequence
- **end_match** – where the hit with the profile ends in the sequence

__lt__(*other:* CoreHit) → bool

Compare two Hits. If the sequence identifier is the same, do the comparison on the score. Otherwise, do it on alphabetical comparison of the sequence identifier.

Parameters
other – the hit to compare to the current object

Returns
True if self is < other, False otherwise

__str__() → str

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

Return type
str

__weakref__

list of weak references to the object

get_position() → int

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

ModelHit

class macsypy.hit.**ModelHit**(*hit:* CoreHit, *gene_ref:* ModelGene, *gene_status:* GeneStatus)

Encapsulates a `macsypy.report.CoreHit` This class stores a CoreHit that has been attributed to a putative system. Thus, it also stores:

- the system,
- the status of the gene in this system, ('mandatory', 'accessory', ...
- the gene in the model for which it's an occurrence

for one gene it can exist several ModelHit instance one for each Model containing this gene

__eq__(*other:* ModelHit) → bool

Return self==value.

__gt__(*other:* ModelHit) → bool

Return self>value.

__hash__() → int

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

__init__(*hit:* CoreHit, *gene_ref:* ModelGene, *gene_status:* GeneStatus) → None

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModeleGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the *macsypy.gene.ModelGene.alternate_of()*

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

- **gene_status** –

__lt__(*other:* ModelHit) → bool

Return self<value.

__str__() → str

Return str(self).

__weakref__

list of weak references to the object

property hit: *CoreHit*

Returns
The CoreHit below this ModelHit

property loner: bool

Returns

True if the hit represent a *loner* `macsypy.Gene.ModelGene`, False otherwise. A True Loner is a hit representing a gene with the attribute loner and which does not include in a cluster.

- a hit representing a loner gene but include in a cluster is not a true loner
- a hit which is not include with other gene in a cluster but does not represent a gene loner is not a True loner (This situation may append when min_genes_required = 1)

property multi_model: bool

Returns

True if the hit represent a *multi_model* `macsypy.Gene.ModelGene`, False otherwise.

property multi_system: bool

Returns

True if the hit represent a *multi_system* `macsypy.Gene.ModelGene`, False otherwise.

AbstractCounterpartHit

class macsypy.hit.**AbstractCounterpartHit**(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: set[*macsypy.hit.ModelHit*]* = *None*)

Abstract Class to handle ModelHit wit equivalent for instance Loner or MultiSystem hit

__init__(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: set[*macsypy.hit.ModelHit*]* = *None*) → None

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModeleGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the *macsypy.gene.ModelGene.alternate_of()*

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

- **gene_status** –

__str__() → str

Return str(self).

property counterpart: set[*macsypy.hit.ModelHit*]

Returns

The set of hits that can play the same role

property loner: bool

Returns

True if the hit represent a *loner* `macsypy.Gene.ModelGene`, False otherwise. A True Loner is a hit representing a gene with the attribute loner and which does not include in a cluster.

- a hit representing a loner gene but include in a cluster is not a true loner

- a hit which is not include with other gene in a cluster but does not represent a gene loner is not a True loner (This situation may append when min_genes_required = 1)

property multi_system: bool

Returns

True if the hit represent a *multi_system* `macsypy.Gene.ModelGene`, False otherwise.

Loner

class macsypy.hit.**Loner**(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*)

Handle hit which encode for a gene tagged as loner and which not clustering with other hit.

__init__(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*) → None

hit that is outside a cluster, the gene_ref is a loner

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** – The ModelGene link to this hit The ModeleGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the *macsypy.gene.ModelGene.alternate_of()*

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

- **gene_status** –
- **counterpart** – the other occurrence of the gene or exchangeable in the replicon

property loner

Returns

True if the hit represent a *loner* `macsypy.Gene.ModelGene`, False otherwise. A True Loner is a hit representing a gene with the attribute loner and which does not include in a cluster.

- a hit representing a loner gene but include in a cluster is not a true loner
- a hit which is not include with other gene in a cluster but does not represent a gene loner is not a True loner (This situation may append when min_genes_required = 1)

MultiSystem

class macsypy.hit.**MultiSystem**(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*)

Handle hit which encode for a gene tagged as loner and which not clustering with other hit.

__init__(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*)

hit that is outside a cluster, the gene_ref is a loner

Parameters

- **hit** – a match between a hmm profile and a replicon

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

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

- **gene_status** –
- **counterpart** – the other occurence of the gene or exchangeable in the replicon

property multi_system: bool

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

LonerMultiSystem

class macsypy.hit.**LonerMultiSystem**(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*)

Handle hit which encode for a gene

- gene tagged as multi-system
- and gene tagged as loner also
- and the hit do not clustering with other hits.

__init__(*hit:* macsypy.hit.CoreHit | macsypy.hit.ModelHit, *gene_ref:* ModelGene = *None*, *gene_status:* GeneStatus = *None*, *counterpart: Iterable[*CoreHit*]* = *None*)

hit that is outside a cluster, the gene_ref is loner and multi_system

Parameters

- **hit** – a match between a hmm profile and a replicon
- **gene_ref** (*macsypy.gene.ModelGene* object) – The ModelGene link to this hit The ModeleGene have the same name as the CoreGene But one hit can be linked to several ModelGene (several Model) To know for what gene this hit play role use the *macsypy.gene.ModelGene.alternate_of()*

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

- **gene_status** (*macsypy.gene.GeneStatus* object) –
- **counterpart** (list of *macsypy.hit.CoreHit*) – the other occurence of the gene or exchangeable in the replicon

HitWeight

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

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

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

__delattr__(*name*)

Implement delattr(self, name).

__eq__(*other*)

Return self==value.

__hash__()

Return hash(self).

__init__(*itself: float = 1*, *exchangeable: float = 0.8*, *mandatory: float = 1*, *accessory: float = 0.5*, *neutral: float = 0*, *out_of_cluster: float = 0.7*) → None

__repr__()

Return repr(self).

__setattr__(*name*, *value*)

Implement setattr(self, name, value).

__weakref__

list of weak references to the object

get_best_hit_4_func

macsypy.hit.**get_best_hit_4_func**(*function: str*, *hits: Iterable[*ModelHit*]*, *key: str = 'score'*) → *ModelHit*

select the best Loner among several ones encoding for same function

- score
- i_evalue
- profile_coverage

Parameters

- **function** – the name of the function fulfill by the hits (all hits must have same function)
- **hits** – the hits to filter.
- **key** – The criterion used to select the best hit 'score', i_evalue', 'profile_coverage'

Returns

the best hit

sort_model_hits

macsypy.hit.**sort_model_hits**(*model_hits: Iterable[*ModelHit*]*) → dict[slice(<class 'str'>, list[macsypy.hit.ModelHit], None)]

Sort *macsypy.hit.ModelHit* per function

Parameters
model_hits – a sequence of *macsypy.hit.ModelHit*

Returns
dict {str function name: [model_hit, ...] }

compute_best_MSHit

macsypy.hit.**compute_best_MSHit**(*ms_registry: dict[slice(<class 'str'>, list[macsypy.hit.MultiSystem | macsypy.hit.LonerMultiSystem], None)]*) → list[*macsypy.hit.MultiSystem* | *macsypy.hit.LonerMultiSystem*]

Parameters
ms_registry –

Returns

get_best_hits

macsypy.hit.**get_best_hits**(*hits: Iterable[*macsypy.hit.CoreHit | macsypy.hit.ModelHit*], key: Literal['score', 'i_eval', 'profile_coverage'] = 'score'*) → list[*macsypy.hit.CoreHit* | *macsypy.hit.ModelHit*]

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

- score
- i_evalue
- profile_coverage

Parameters

- **hits** ([*macsypy.hit.CoreHit* object, ...]) – the hits to filter, all hits must match the same protein.
- **key** (*str*) – The criterion used to select the best hit 'score', i_evalue', 'profile_coverage'

Returns
the list of the best hits

Return type
[*macsypy.hit.CoreHit* object, ...]

cluster

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

cluster API reference

cluster

class macsypy.cluster.**Cluster**(*hits: list[*macsypy.hit.CoreHit | macsypy.hit.ModelHit*]*, *model*, *hit_weights*)

Handle hits relative to a model which collocates

__contains__(*m_hit:* ModelHit) → bool

Parameters
m_hit – The hit to test

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

__init__(*hits: list[*macsypy.hit.CoreHit | macsypy.hit.ModelHit*]*, *model*, *hit_weights*) → None

Parameters

- **hits** – the hits constituting this cluster
- **model** – the model associated to this cluster
- **hit_weights** – the weight of the hit to compute the score

__str__() → str

Returns
a string representation of this cluster

__weakref__
list of weak references to the object

_check_replicon_consistency() → None

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

fulfilled_function(**genes:* macsypy.gene.ModelGene | *str*) → frozenset[str]

Parameters
genes – The genes which must be tested.

Returns
the common functions between genes and this cluster.

property functions: frozenset[str]

Returns

The set of functions encoded by this cluster *function* mean gene name or reference gene name for exchangeables genes for instance

<model vers="2.0">
<gene a presence="mandatory"/> <gene b presence="accessory"/>

<exchangeable>
<gene c />

</exchangeable>

<gene/>

</model>

the functions for a cluster corresponding to this model wil be { 'a' , 'b' }

property hit_weights: *HitWeight*

Returns
the different weight for the hits used to compute the score

property loner: **bool**

Returns

True if this cluster is made of only some hits representing the same gene and this gene is tag as loner False otherwise:

- contains several hits coding for different genes
- contains one hit but gene is not tag as loner (max_gene_required = 1)

merge(*cluster:* Cluster, *before: bool = False*) → None

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

Parameters

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

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

property multi_system: **bool**

Returns

True if this cluster is made of only one hit representing a multi_system gene False otherwise:

- contains several hits
- contains one hit but gene is not tag as loner (max_gene_required = 1)

replace(*old:* ModelHit, *new:* ModelHit) → None

replace hit old in this cluster by new one. (do it in place)

Parameters

- **old** – the hit to replace
- **new** – the new hit

Returns
None

property replicon_name: **str**

Returns
The name of the replicon where this cluster is located

Return type
str

property score: float

Returns
The score for this cluster

build_clusters

macsypy.cluster.**build_clusters**(*hits: list[*macsypy.hit.ModelHit*]*, *rep_info:* RepliconInfo, *model:* Model, *hit_weights:* HitWeight) → tuple[list[macsypy.cluster.Cluster], dict[slice(<class 'str'>, macsypy.hit.Loner | macsypy.hit.LonerMultiSystem, None)]]

From a list of filtered hits, and replicon information (topology, length), build all lists of hits that satisfied the constraints:

- max_gene_inter_space
- loner
- multi_system

If Yes create a cluster. A cluster contains at least two hits separated by less or equal than max_gene_inter_space Except for loner genes which are allowed to be alone in a cluster

Parameters

- **hits** – list of filtered hits
- **rep_info** – the replicon to analyse
- **model** – the model to study
- **hit_weights** – the hit weight needed to compute the cluster score

Returns
list of regular clusters, the special clusters (loners not in cluster and multi systems)

Return type

tuple with 2 elements

- true_clusters which is list of *Cluster* objects
- true_loners: a dict { str function: :class:macsypy.hit.Loner | :class:macsypy.hit.LonerMultiSystem object}

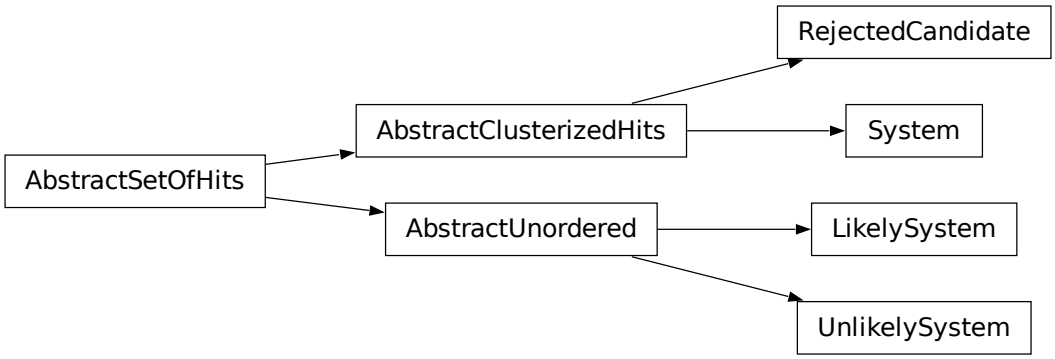
system

This module classes and functions which a given set of hits and a model compute if this set satisfy the model or not

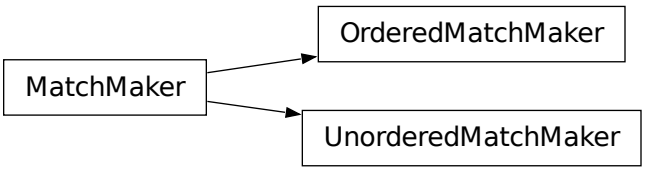
The object which check the compliance of hits to a model is MatchMaker which have 2 sub-classes for ordered and unordered replicons

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 instanciate a *macsypy.system.System* or *macsypy.system.RejectedCandidate* 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 reference api

MatchMaker

class macsypy.system.**MatchMaker**(*model:* Model)

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

__init__(*model:* Model) → None

__weakref__

list of weak references to the object

_create_exchangeable_map(*genes: list[*macsypy.gene.ModelGene*]*) → dict[slice(<class 'str'>, <class 'macsypy.gene.ModelGene'>, None)]

create a map between an exchangeable (formly homolog or analog) gene name and it's gene reference

Parameters
genes – The genes to get the exchangeable genes

Returns
a dict with keys are the exchangeable gene_name and the value the reference gene

present_genes() → tuple[list[str], list[str], list[str], list[str]]

Returns

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

tuple of 4 lists for mandatory, accessory, neutral and forbidden

([str gene_name, ...], [str gene_name], [str gene_name], [str gene_name])

sort_hits_by_status(*hits: Iterable[*ModelHit*]*) → tuple[list[*macsypy.hit.ModelHit*], list[*macsypy.hit.ModelHit*], list[*macsypy.hit.ModelHit*], list[*macsypy.hit.ModelHit*]]

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

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

Returns
the valid hits according their status ([mandatory,], [accessory,], [neutral,], [forbidden])

Raises
MacsypyError – when a gene is not found in the model

OrderedMatchMaker

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*)

match(*clusters: Iterable[*Cluster*]*) → *macsypy.system.System* | *macsypy.system.RejectedCandidate*

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

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

Returns
either a System or a RejectedCandidates

Return type
macsypy.system.System or *macsypy.system.RejectedCandidate* object

UnorderedMatchMaker

class macsypy.system.**UnorderedMatchMaker**(*model:* Model)

match(*hits: Iterable[*ModelHit*]*) → *macsypy.system.LikelySystem* | *macsypy.system.UnlikelySystem*

Parameters
hits – the hits to check

HitSystemTracker

class macsypy.system.**HitSystemTracker**(*systems: list[*macsypy.system.System*]*)

track in which system is implied each hit

__init__(*systems: list[*macsypy.system.System*]*) → None

__weakref__
list of weak references to the object

MetaSetOfHits

class macsypy.system.**MetaSetOfHits**(*name*, *bases*, *namespace*, */*, ***kwargs*)

This metaclass control the AbstractSetOfHits class creation. In this metaclass we inject on the fly several attributes and properties two private attributes and one public property corresponding to each value of _supported_status class attribute defined in the concrete classes. for instance for System class

- **the attributes**
  - self._mandatory
  - self._mandatory_occ
  - self._accessory
  - self._accessory_occ
  - self._neutral
  - self._neutral_occ
- **and the properties**
  - mandatory
  - accessory
  - neutral

are automatically injected

The value for attributes _<*status*>_*occ* are filled by the *count* method which is defined in AbstractSetOfHits

__call__(**args*, ***kwargs*)
Call self as a function.

getter_maker() → Callable
Create a property which allow to access to the gene corresponding of the cat of the model

Parameters
status – the type of gene category to which we create the getter

Returns
unbound method

AbstractSetOfHits

class macsypy.system.**AbstractSetOfHits**(**args*, ***kwargs*)

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

__init__(*model:* Model) → None

__weakref__
list of weak references to the object

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

property position: tuple[int, int]

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

property replicon_name: str

Returns
The name of the replicon

Return type
str

property wholeness: float

Returns
a score indicating the genes ratio of the model which have at least one hit by default full system is mandatory + accessory ('neutral' genes do not count) but for special corner case it can be specified in model definition (xml) or on the command line

AbstractClusterizedHits

class macsypy.system.**AbstractClusterizedHits**(**args*, ***kwargs*)

Modelize SetOfHits that colocalize.

should be inherited

__init__(*model:* Model, *clusters:* macsypy.cluster.Cluster | *list[*macsypy.cluster.Cluster*]*)

fulfilled_function(**genes:* macsypy.gene.ModelGene | *str*) → set[str]

Parameters
genes (*macsypy.gene.ModelGene* object or string representing the gene name) – The genes which must be tested.

Returns
the common functions between genes and this system.

Return type
set of string

System

class macsypy.system.**System**(**args*, ***kwargs*)

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

__init__(*model:* Model, *clusters: list[*macsypy.cluster.Cluster*]*, *redundancy_penalty: float = 1.5*) → None

Parameters

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

__str__() → str

Return str(self).

get_hits_encoding_multisystem() → set[*macsypy.hit.MultiSystem*]

Returns
The hits codding for a gene taged as multi system

get_loners() → set[*macsypy.hit.Loner* | *macsypy.hit.LonerMultiSystem*]

Returns
The True Loners (Loner which not colocalize with another hit) belonging to the systems

get_multisystems() → set[*macsypy.hit.MultiSystem* | *macsypy.hit.LonerMultiSystem*]

Returns
The MultiSystem hit (comming from out system (other cluster or loner) and tag as multisystem)

property hits: list[*macsypy.hit.ModelHit*]

Returns
The list of all hits that compose this system

is_compatible(*other:* System) → bool

Parameters
other – the other systems to test compatibility

Returns

True if other system is compatible with this one. False otherwise. Two systems are compatible if they do not share *macsypy.hit.CoreHit* except hit corresponding to a multi_system gene in the model.

Note: This method is used to compute the best combination of systems.

---

property loci_nb: int

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

Return type
int >= 0

property loci_num: list[int]

Returns
the number of the corresponding locus for each cluster the cluster made of only one Loner are not considered as a loci so these clusters have a negative locus_num

property multi_loci: bool

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

occurrence() → int

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: float

Returns
a score take in account * if a hit match for the gene or it is an exchangeable gene * if a hit is duplicated and already present in the system or the cluster * if a hit match for mandatory/accessory gene of the model

Return type
float

RejectedCandidate

class macsypy.system.RejectedCandidate(*args, **kwargs)

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:* Model, *clusters: list[*macsypy.cluster.Cluster*]*, *reasons: list[str]*) → None

Parameters

- **model** –
- **clusters** – list of clusters. These Clusters should be created with *macsypy.cluster.Cluster* of *macsypy.hit.ModelHit* objects
- **reasons** – the reason why these clusters have been rejected

__str__() → str

Returns
a string representation of this RejectedCandidates

property hits: list[*macsypy.hit.ModelHit*]

Returns
The list of all hits that compose this system

property reasons: list[str]

Returns
The reason why it has been rejected

AbstractUnordered

class macsypy.system.**AbstractUnordered**(**args*, ***kwargs*)

Technical abstract class to factorize code share between LikelySystem and UnlikelySystem

__init__(*model:* Model, *mandatory_hits: list[*macsypy.hit.ModelHit*]*, *accessory_hits: list[*macsypy.hit.ModelHit*]*, *neutral_hits: list[*macsypy.hit.ModelHit*]*, *forbidden_hits: list[*macsypy.hit.ModelHit*]*) → None

Parameters

- **model** – The model which has been used to build this system
- **mandatory_hits** – The list of mandatory hits (encode for a gene tagged as mandatory)
- **accessory_hits** – The list of accessory hits (encode for a gene tagged as accessory)
- **neutral_hits** – The list of neutral hits (encode for a gene tagged as neutral)
- **forbidden_hits** – The list of hits that are forbidden

property accessory_hits: list[*macsypy.hit.ModelHit*]

Returns
The list of accesory hits

property allowed_hits: list[*macsypy.hit.ModelHit*]

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

property forbidden_hits: list[*macsypy.hit.ModelHit*]

Returns
The list of forbidden hits

property hits: list[*macsypy.hit.ModelHit*]

Returns
The list of all hits sorted by their position

property mandatory_hits: list[*macsypy.hit.ModelHit*]

Returns
The list of mandatory hits

property neutral_hits: list[*macsypy.hit.ModelHit*]

Returns
The list of neutral hits

LikelySystem

class macsypy.system.**LikelySystem**(**args*, ***kwargs*)

" Handle components that fill the quorum requirements defined in model. with no idea about genetic organization (gene cluster) so we cannot take in account forbidden genes

__str__() → str

Returns
a string representation of this LikelySystem

UnlikelySystem

class macsypy.system.**UnlikelySystem**(**args*, ***kwargs*)

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

__init__(*model:* Model, *mandatory_hits: list[*macsypy.hit.ModelHit*]*, *accessory_hits: list[*macsypy.hit.ModelHit*]*, *neutral_hits: list[*macsypy.hit.ModelHit*]*, *forbidden_hits: list[*macsypy.hit.ModelHit*]*, *reasons: list[str]*) → None

Parameters

- **model** – The model which has been used to build this system
- **mandatory_hits** – The list of mandatory hits (encode for a gene tagged as mandatory)
- **accessory_hits** – The list of accessory hits (encode for a gene tagged as accessory)
- **neutral_hits** – The list of neutral hits (encode for a gene tagged as neutral)
- **forbidden_hits** – The list of hits that are forbidden
- **reasons** – the reasons why this set of hits has been rejected

__str__() → str

Returns
a string representation of this UnlikelySystem

property **reasons: list[str]**

Returns
The reasons why it probably not a system

Return type
list of string

report

A *"HMMReport"* object represents the results of a Hmmer program search on a dataset with a hidden Markov model protein profile (see *this section*). This object has methods to extract and filter Hmmer raw outputs (see *generated output files*), and then build Hits relevant for system detection. For matches selected with the filtering parameters, *"Hit"* objects (`macsypy.HMMReport.Hit`) are built.

report API reference

HMMReport

class macsypy.report.**HMMReport**(*gene:* CoreGene, *hmmer_output: str*, *cfg:* Config)

Handle the results from the HMM search. Extract a synthetic report from the raw hmmer output, after having applied a hit filtering. This class is an **abstract class**. There are two implementations of this abstract class depending on whether the input sequence dataset is "ordered" ("gembase" or "ordered_replicon" db_type) or not ("unordered" db_type).

__init__(*gene:* CoreGene, *hmmer_output: str*, *cfg:* Config) → None

Parameters

- **gene** – the gene corresponding to the profile search reported here

- **hmmer_output** – The path to the raw Hmmer output file
- **cfg** – the configuration object

__str__() → str

Returns
string representation of this report

__weakref__
list of weak references to the object

_build_my_db(*hmm_output: str*) → dict[slice(<class 'str'>, None, None)]
Build the keys of a dictionary object to store sequence identifiers of hits.

Parameters
hmm_output (*string*) – the path to the hmmsearch output to parse.

Returns
a dictionary containing a key for each sequence id of the hits

_fill_my_db(*db: dict[slice(<class 'str'>, tuple[int, int], None)]*) → None
Fill the dictionary with information on the matched sequences

Parameters
db (*dict*) – the database containing all sequence id of the hits.

abstract _get_replicon_name(*hit_id: str*) → str
This method is used by extract method and must be implemented by concrete class

Parameters
hit_id (*str*) – the id of the current hit extract from hmm output.

Returns
The name of the replicon

_hit_start(*line: str*) → bool

Parameters
line (*string*) – the line to parse

Returns
True if it's the beginning of a new hit in Hmmer raw output files. False otherwise

Return type
boolean.

_parse_hmm_body(*hit_id: str*, *gene_profile_lg: int*, *seq_lg: int*, *coverage_threshold: float*, *replicon_name: str*, *position_hit: int*, *i_evalue_sel: float*, *b_grp: Iterator*) → list[*macsypy.hit.CoreHit*]
Parse the raw Hmmer output to extract the hits, and filter them with threshold criteria selected ("coverage_profile" and "i_evalue_select" command-line parameters)

Parameters

- **hit_id** – the sequence identifier
- **gene_profile_lg** – the length of the profile matched
- **seq_lg** – the length of the sequence
- **coverage_threshold** – the minimal coverage of the profile to be reached in the Hmmer alignment for hit selection.
- **replicon_name** – the identifier of the replicon

- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_evalue_sel** – the maximal i-evalue (independent evalue) for hit selection
- **b_grp** (*list of list of strings*) – the Hmmer output lines to deal with (grouped by hit)

Returns
a sequence of hits

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

_parse_hmm_header(*h_grp: Iterator*) → str

Parameters
h_grp (*sequence of string (<itertools._grouper object at 0x7ff9912e3b50>)*) – the sequence of string return by groupby function representing the header of a hit

Returns
the sequence identifier from a set of lines that corresponds to a single hit

Return type
string

best_hit() → *macsypy.hit.CoreHit* | None

Return the best hit among multiple hits

extract() → None | list[*macsypy.hit.CoreHit*]

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

save_extract() → None

Write the string representation of the extract report in a file. The name of this file is the concatenation of the gene name and of the "res_extract_suffix" from the config object

GeneralHMMReport

class macsypy.report.**GeneralHMMReport**(*gene:* CoreGene, *hmmer_output: str*, *cfg:* Config)

Handle HMM report. Extract a synthetic report from the raw hmmer output. Dedicated to any type of 'unordered' datasets.

_get_replicon_name(*hit_id: str*) → str

This method is used by extract method and must be implemented by concrete class

Parameters
hit_id (*str*) – the id of the current hit extract from hmm output.

Returns
The name of the replicon

OrderedHMMReport

class macsypy.report.**OrderedHMMReport**(*gene:* CoreGene, *hmmer_output: str*, *cfg:* Config)

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

_get_replicon_name(*hit_id: str*) → str

This method is used by extract method and must be implemented by concrete class

Parameters
hit_id (*str*) – the id of the current hit extract from hmm output.

Returns
The name of the replicon

GembaseHMMReport

class macsypy.report.**GembaseHMMReport**(*gene:* CoreGene, *hmmer_output: str*, *cfg:* Config)

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

_get_replicon_name(*hit_id: str*) → str

This method is used by extract method and must be implemented by concrete class

Parameters
hit_id (*str*) – the id of the current hit extract from hmm output.

Returns
The name of the replicon

… MacSyFinder - Detection of macromolecular systems in protein datasets
using systems modelling and similarity search. Authors: Sophie Abby, Bertrand Néron Copyright © 2014-2023 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 hmmsearch* to find the genes constituting the models in the input dataset.

search_genes API reference

search_genes

Manage the hmm step (hmmsearch or recover results from previous run) in parallele

macsypy.search_genes.**search_genes**(*genes: list[*macsypy.gene.ModelGene*]*, *cfg:* Config) → list[*macsypy.report.HMMReport*]

For each gene of the list, use the corresponding profile to perform an Hmmer search, and parse the output to generate a HMMReport that is saved in a file after CoreHit filtering. These tasks are performed in parallel using threads. The number of workers can be limited by worker_nb directive in the config object or in the command-line with the "-w" option.

Parameters

- **genes** – the genes to search in the input sequence dataset
- **cfg** – the configuration object

macsypy.search_genes.**worker_cpu**(*genes_nb: int*, *cfg:* Config) → tuple[int, int]

Compute the optimum number of worker and cpu per worker The number of worker is set by the user (1 by default 0 means all worker available)

we use one worker per gene if number of workers is greater than number of genes then several cpu can be use by hmsearch to speed up the search step

Parameters

- **genes_nb** – the number of genes to search
- **cfg** – The macsyfinder configuration

Returns
the number of worker and cpu_per_worker to use

Return type
tuple (int worker_nb, int cpu_per_worker)

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 API reference

Solution

class macsypy.solution.**Solution**(*systems: list[*macsypy.system.System*]*)

Handle Solution, a solution is a set of compatible Systems

when compare solutions we check the following criteria

1. The number of hits
2. The number of systems
3. The average of wholeness
4. The hits position (is used ti give predictable output for unit tests)

__eq__(*other*) → bool

Return self==value.

__gt__(*other*) → bool

Return self>value.

__hash__ = None

__init__(*systems: list[*macsypy.system.System*]*) → None

Parameters
systems – The list of system that composed this solution

__iter__() → Generator

Solution allow to iterate over the systems

Returns
generator

__lt__(*other*) → bool

Return self<value.

__weakref__

list of weak references to the object

_sorted_systems(*systems: list[*macsypy.system.System*]*) → list[*macsypy.system.System*]

sort the systems following the positions of th hits that composed the systems

Parameters
systems (list of `mcsypy.system.System` objects) – the systems to sort

Returns
a sorted copy of the *systems*

Return type
list of `mcsypy.system.System` objects

property average_wholeness: float

The average of the systems wholeness

property hits_number: int

The sum of the hits of each system in this solution

property hits_positions: list[int]

The list of position of all hits of the solution

property score: float

The score of this solution

property systems: list[*macsypy.system.System*]

“a sorted list of the *systems* that composed the solution

combine_clusters

macsypy.solution.**combine_clusters**(*clusters: list[macsypy.cluster.Cluster], true_loners: dict[slice(<class 'str'>, macsypy.hit.Loner | macsypy.hit.LonerMultiSystem, None)], multi_loci: bool = False*) → list[tuple[*macsypy.cluster.Cluster*]]

generate the combinations of clusters, with loners and multi systems

Parameters

- **clusters** – the clusters to combines
- **true_loners** (dict the name of the function code by hit gene_ref.alternate_of as key and 1 *macsypy.cluster.Cluster* with the best a *macsypy.hit.Loner* or *macsypy.hit.LonerMultiSystem* hit as value) – the multi-systems hits
- **multi_loci** – True if the model is multi_loci false otherwise

Returns
all available combination of clusters

Return type
List of combination. a combination is a tuple of *macsypy.cluster.Cluster* objects

combine_multisystems

macsypy.solution.**combine_multisystems**(*rejected_candidates: list[*macsypy.system.RejectedCandidate*]*, *multi_systems: list[*macsypy.cluster.Cluster*]*)

Parameters

- **rejected_candidates** –
- **multi_systems** – sequence of *macsypy.cluster.Cluster* each cluster must be composed of only one *macsypy.hit.MultiSystem* object

Returns
list of cluster combination with the multisystem

Return type
[(*macsypy.cluster.Cluster* cluster1, cluster2, ...), (*macsypy.cluster.Cluster* cluster3, cluster4, ...)]

find_best_solutions

macsypy.solution.**find_best_solutions**(*systems: list[*macsypy.system.System*]*) → tuple[list[*macsypy.solution.Solution*], float]

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

Parameters
systems – the systems to analyse

Returns
the list of *macsypy.solution.Solution* which represent one best solution then it's score.

Return type
tuple of 2 elements the best solutions and it's score

serialization

This module is a technical module where we can find the different way to serialize the results:

- the Systems found
- The best solutions (best combination of systems)
- The rejected candidates

SystemSerializer

class macsypy.serialization.**SystemSerializer**

handle the different way to serialize a system

__weakref__

list of weak references to the object

TsvSystemSerializer

class macsypy.serialization.**TsvSystemSerializer**

Handle System serialization in tsv format

serialize(*system:* System, *hit_system_tracker:* HitSystemTracker) → str

:param *macsypy.system.System* system: The system to serialize. :param hit_system_tracker: The hit_system_tracker which allow to know for each hit

in which system it is implied.

Returns

a serialisation of this system in tabulated separated value format each line represent a hit and have the following structure:

```
replicon\thit_id\tgene_name\thit_pos\tmodel_fqn\tsys_id\tsys_loci\tlocus_num\
↪tsys_wholeness\tsys_score
\tsys_occ\thit_gene_ref.alternate_of\thit_status\thit_seq_len\thit_i_eval\thit_
↪score\thit_profile_cov
\thit_seq_cov\tit_begin_match\thit_end_match\tcounterpart\tused_in_systems
```

Return type

str

TsvSolutionSerializer

class macsypy.serialization.**TsvSolutionSerializer**

Handle Solution (list of Systems) serialization in tsv format

__weakref__

list of weak references to the object

serialize(*solution:* Solution, *sol_id: int*, *hit_system_tracker:* HitSystemTracker) → str

Parameters

- **solution** – the solution to serialize
- **sol_id** – the solution identifier
- **hit_system_tracker** –

Returns

a serialisation of this solution (a list of systems) in tabulated separated value format each line represent a hit and have the same structure as system serialization *macsypy.serialization.TsvSystemSerializer.serialize()* but with an extra column sol_id which is a technical id to identify the different solutions.

TsvLikelySystemSerializer

class macsypy.serialization.**TsvLikelySystemSerializer**

Handle potential System from unordered replicon serialization in tsv format

serialize(*system:* LikelySystem, *hit_system_tracker:* HitSystemTracker) → str

Parameters

- **system** – The likely system to serialize. Used only for unordered db-type
- **hit_system_tracker** – The hit_system_tracker which allow to know for each hit in which system it is implied.

Returns

a serialisation of this system in tabulated separated value format each line represent a hit and have the following structure:

```
replicon\thit_id\tgene_name\thit_pos\tmodel_fqn\tsys_id\tsys_wholeness
\thit_gene_ref.alternate_of\thit_status\thit_seq_len\thit_i_eval\thit_score\
↪thit_profile_cov
\thit_seq_cov\tit_begin_match\thit_end_match\t$used_in_systems
```

Return type

str

TsvRejectedCandidatesSerializer

class macsypy.serialization.**TsvRejectedCandidatesSerializer**

Serialize Rejected Cluster in tsv format

__weakref__

list of weak references to the object

serialize(*candidates: list[*macsypy.system.RejectedCandidate*]*) → str

Parameters

candidates – list of rejected candidates to serialize

TsvSpecialHitSerializer

class macsypy.serialization.**TsvSpecialHitSerializer**

Serialize special hits: *macsypy.hit.Loner* and *macsypy.hit.MultiSystem* in tsv format

__weakref__

list of weak references to the object

serialize(*best_hits: Union[Iterable[*Loner*], Iterable[*MultiSystem*]]*)

Parameters

best_hits (sequence of *macsypy.hit.Loner* or *macsypy.hit.MultiSystem* objects) – the special hits to serialized

TxtSystemSerializer

class macsypy.serialization.**TxtSystemSerializer**

Handle System serialization in text

serialize(*system:* System, *hit_system_tracker:* HitSystemTracker) → str

Returns
a string representation of system readable by human

TxtLikelySystemSerializer

class macsypy.serialization.**TxtLikelySystemSerializer**

Handle System serialization in text

serialize(*system:* LikelySystem, *hit_system_tracker:* HitSystemTracker)

Parameters

- **system** – The likely system to serialize. Used only for unordered db-type
- **hit_system_tracker** – The hit_system_tracker which allow to know for each hit in which system it is implied.

Returns
a string representation of system readable by human

TxtUnikelySystemSerializer

class macsypy.serialization.**TxtUnikelySystemSerializer**

Handle System serialization in text

serialize(*system:* UnlikelySystem) → str

Parameters
system – The unlikely system to serialize. (used only if db-type is "unordered_replicon")

Returns
a string representation of system readable by human

database

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

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 API reference

Indexes

class macsypy.database.**Indexes**(*cfg:* Config)

Handle the indexes for macsyfinder:

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

__init__(*cfg:* Config) → None

The constructor retrieves the file of indexes in the case they are not present or the user asked for build indexes (–idx) Launch the indexes building.

Parameters
cfg (*macsypy.config.Config* object) – the configuration

__iter__() → Iterator[tuple[str, str, int]]

Raises
MacsypyError – if the indexes are not buid

Returns
an iterator on the indexes

To use it the index must be built.

__weakref__

list of weak references to the object

_build_my_indexes(*index_dir: str*) → str

Build 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: bool = False*) → str

search where to store(build=True) read indexes

Parameters
build – if check the index-dir permissions to write

Returns
The directory where read or write the indexes

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

build(*force: bool = False*) → str

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

Parameters
force – If True, force the index building even if the index files are present in the sequence data set folder

Returns
the path to the index

find_my_indexes() → str | None

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

Return type
string

RepliconInfo

Module to handle sequences and their indexes

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

handle information about a replicon

topology

The type of replicon topology 'linear or 'circular'

min

The position of the last gene of the replicon in the sequence dataset.

max

The position of the last gene of the replicon in the sequence dataset.

genes

A list of genes beloging to the replicon. Each genes is representing by a tuple (str seq_id, int length)

genes

Alias for field number 3

max

Alias for field number 2

min

Alias for field number 1

topology

Alias for field number 0

RepliconDB

class macsypy.database.**RepliconDB**(*cfg:* Config)

Stores information (topology, min, max, [genes]) for all replicons in the sequence_db the Replicon object must be instantiated only for sequence_db of type 'gembase' or 'ordered_replicon'

__contains__(*replicon_name: str*) → bool

Parameters
replicon_name – the name of the replicon

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

__getitem__(*replicon_name: str*) → *RepliconInfo*

Parameters
replicon_name – the name of the replicon to get information on

Returns
the RepliconInfo for the provided replicon_name

Raise
KeyError if replicon_name is not in repliconDB

__init__(*cfg:* Config) → None

Parameters
cfg (*macsypy.config.Config* object) – The configuration object

---

Note: This class can be instanced only if the db_type is 'gembase' or 'ordered_replicon'

__weakref__
list of weak references to the object

_fill_gembase_min_max(*topology: dict[slice(<class 'str'>, typing.Literal['linear', 'ciircular'], None)], default_topology: ~typing.Literal['linear', 'ciircular']*) → None

For each replicon_name of a gembase dataset, it fills the internal dictionary with a namedtuple RepliconInfo

Parameters

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

_fill_ordered_min_max(*default_topology: Optional[Literal['linear', 'ciircular']] = None*) → None

For the replicon_name of the ordered_replicon sequence base, fill the internal dict with RepliconInfo

Parameters
default_topology (*string*) – the topology provided by config.replicon_topology

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

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

get(*replicon_name: str*, *default: Any = None*) → *RepliconInfo*

Parameters

- **replicon_name** – the name of the replicon to get information
- **default** – the value to return if the replicon_name is not in the RepliconDB

Returns
the RepliconInfo for replicon_name if replicon_name is in the repliconDB, else default. If default is not given, it is set to None, so that this method never raises a KeyError.

guess_if_really_gembase() → bool

Count the number of replicon with only on sequence if this number is above a threshold may be it's not gembase. for instance the folowing sequence have id compliant with the gembase id syntax but it's not it only contains one replicon ('ordered replicon')

>1E10S0A0cP00_0010 D GTG TGA 483 2027 Valid dnaA 1545 _PA0001_NP_064721.1_ PA0001 1 483 2027
MSVELWQQCVDLLRDELPSQQFNTWIRPLQVEAEGDELRVYAPNRFVLDW
>0200S001A0c_0P1E0 D ATG TAA 2056 3159 Valid dnaN 1104 _PA0002_NP_064722.1_ PA0002 1 2056 3159
MHFTIQREALLKPLQLVAGVVERRQTLPVLSNVLLVVEGQQLSLTGTDLE
>0000310E00S0c_1PA D ATG TGA 3169 4278 Valid recF 1110 _PA0003_NP_064723.1_ PA0003 1 3169 4278
MSLTRVSVTAVRNLHPVTLSPSPRINILYGDNGSGKTSVLEAIHLLGLAR
>c_01000A0PS00014E D ATG TGA 4275 6695 Valid gyrB 2421 _PA0004_NP_064724.1_ PA0004 1 4275 6695
MSENNTYDSSSIKVLKGLDAVRKRPGMYIGDTDDGTGLHHMVFEVVDNSI
>07700ES100A0cP01_ C ATG TGA 91521 94826 Valid icmF1 3306 _PA0077_NP_248767.1_ PA0077 1 91521 94826
MQSLAEVSAPDAASVAT

Returns
False if most replicon contains only one sequence, True otherwise

items() → list[tuple[str, *macsypy.database.RepliconInfo*]]

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

iteritems() → Iterator[tuple[str, *macsypy.database.RepliconInfo*]]

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

replicon_infos() → list[*macsypy.database.RepliconInfo*]

Returns
a copy of the RepliconDB as list of replicons info

Return type
RepliconInfo instance

replicon_names() → list[str]

Returns
a copy of the RepliconDB as a list of replicon_names

fasta_iter

macsypy.database.**fasta_iter**(*fasta_file: TextIO*) → Iterator[tuple[str, str, int]]

Parameters
fasta_file – the file containing all input sequences in fasta format.

Author
http://biostar.stackexchange.com/users/36/brentp

Returns
for a given fasta file, it returns an iterator which yields tuples (string id, string comment, int sequence length)

errors

The errors specific to macsyfinder and macsydata

error API reference

error

Manage MacSyFinder specific errors

exception macsypy.error.**EmptyFileError**
Raised when fasta file does not contains sequences

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

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.

exception macsypy.error.**Timeout**
Raised when best solution reach the timeout

utils

Here some useful functions in the rest of macsyfinder code

utils API reference

get_def_to_detect

macsypy.utils.**get_def_to_detect**(*models: list[tuple[str, tuple[str]]]*, *model_registry:* ModelRegistry) → tuple[list[*macsypy.registries.DefinitionLocation*], str, str]

Parameters

- **models** (*list of tuple with the following structure: [('model_fqn', ('def1, def2, ...)), ('model_2', ('def1', ...)), ...]*) – the list of models to detect as returned by config.models.

- **model_registry** – the models registry for this run.

Returns
the definitions to parse

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

get_replicon_names

macsypy.utils.**get_replicon_names**(*genome_path*, *db_type*) → list[str]

threads_available

macsypy.utils.**threads_available**() → int

Returns
The maximal number of threads available. It's nice with cluster scheduler or linux. On Mac it uses the number of physical cores

parse_time

macsypy.utils.**parse_time**(*user_time: int | str*) → int

parse user-friendly time and return it in seconds user time supports units as s h m d for sec min hour day or a combination of them 1h10m50s means 1 hour 10 minutes 50 seconds all terms will be converted in seconds and added

Parameters
user_time –

Returns
seconds

Raise
ValueError if user_time is not parseable

package

Allow to handles model package either on localhost or from a remote location. the model packages can be stored in github organization to be downloaded and installed locally. The classes below are used by *macsydata*, which is the entry point to manipulate models package.

package API reference

AbstractModelIndex

class macsypy.package.**AbstractModelIndex**(*cache: str | None = None*)

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

__init__(*cache: str | None = None*) → None

__weakref__

list of weak references to the object

unarchive_package(*path: str*) → str

Unarchive and uncompress a package under *<remote cache>/<organization name>/<package name>/<vers>/<package name>*

Parameters
path (*str*) –

Returns
The path to the package

LocalModelIndex

class macsypy.package.**LocalModelIndex**(*cache: str | None = None*)

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

__init__(*cache: str | None = None*) → None

RemoteModelIndex

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

This class allow to interact with ModelIndex on github

__init__(*org: str = 'macsy-models'*, *cache: str | None = None*) → None

Parameters
org – The name of the organization on github where are stored the models

_url_json(*url: str*) → dict

Get the url, deserialize the data as json

Parameters
url (*str*) – the url to download

Returns
the json corresponding to the response url

download(*pack_name: str*, *vers: str*, *dest: str | None = None*) → str

Download a package from a GitHub repos and save it as <remote cache>/<organization name>/<package name>/<vers>.tar.gz

Parameters

- **pack_name** (*str*) – the name of the package to download
- **vers** (*str*) – the version of the package to download
- **dest** (*str*) – The path to the directory where save the package This directory must exist If dest is None, the 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

Returns

The list of package names.

remote_exists() → bool

check if the remote exists and is an organization

Returns

True if the Remote url point to a GitHub Organization, False otherwise

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 contain at least - a subdirectory definitions - a subdirectory profiles - a file metadata.yml it is also recommended to add a file for licensing and copyright and a README. for further explanation see documentation: modeler guide > package

__init__(*path: str*) → None

Parameters

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

__weakref__

list of weak references to the object

_check_metadata() → tuple[list[str], list[str]]

Check the QA of package metadata_path

Returns

errors and warnings

Return type

tuple of 2 lists ([str error_1, ...], [str warning_1, ...])

_check_model_conf() → tuple[list[str], list[str]]

check if a model configuration file is present in the package (model_conf.xml) if the syntax of this file is good.

Returns

_check_model_consistency() → tuple[list, list]

check if each xml seems well write, each genes have an associated profile, etc.

Returns

_check_structure() → tuple[list[str], list[str]]

Check the QA structure of the package

Returns
errors and warnings

Return type
tuple of 2 lists ([str error_1, ...], [str warning_1, ...])

_find_readme() → str | None

find the README file

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

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

Open the metadata_path file and de-serialize it's content :return:

check() → tuple[list[str], list[str]]

Check the QA of this package

help() → str

return the content of the README file

info() → str

Returns
some information about the package

property metadata: dict[slice(<class 'str'>, <class 'str'>, None)]

Returns
The parsed metadata as a dict

scripts

The are 4 entry points.

- macsyfinder: which is the main scripts
- macsydata: which allow to manage the models
- macsyconfig: an interactive conversational utility to generate macsyfinder configuration file
- macsyprofile: an utility dedicated to modelers which gather information about hmmer output

API reference

macsyfinder

Main entrypoint to macsyfinder

macsypy.scripts.macsyfinder.**_loner_warning**(*systems: list[*macsypy.system.System*]*) → list[str]

Parameters
systems – sequence of systems

Returns
warning for loner which have less occurrences than systems occurrences in which this lone is used except if the loner is also multi system

macsypy.scripts.macsyfinder.**_outfile_header**(*models_fam_name: str*, *models_version: str*, *skipped_replicons: list[str] | None = None*) → str

Returns
The 2 first lines of each result file

macsypy.scripts.macsyfinder.**_search_in_ordered_replicon**(*hits_by_replicon: dict[slice(<class 'str'>, list[macsypy.hit.CoreHit], None)], models_to_detect: list[macsypy.model.Model], config: ~macsypy.config.Config, logger: ~logging.Logger*) → tuple[list[*macsypy.system.System*], list[*macsypy.system.RejectedCandidate*]]

Parameters

- **hits_by_replicon** – the hits sort by replicon and position
- **models_to_detect** – the models to search
- **config** – MSF configuration
- **logger** – the logger

macsypy.scripts.macsyfinder.**_search_in_unordered_replicon**(*hits_by_replicon: dict[slice(<class 'str'>, list[macsypy.hit.CoreHit], None)], models_to_detect: list[macsypy.model.Model], logger: ~logging.Logger*) → tuple[list[*macsypy.system.LikelySystem*], list[*macsypy.system.UnlikelySystem*]]

Parameters

- **hits_by_replicon** – the hits sort by replicon and position
- **models_to_detect** – the models to search
- **logger** – the logger

macsypy.scripts.macsyfinder.**alarm_handler**(*signum: Signals*, *frame*) → None

Handle signal alarm flush loggers :param signum: :param frame: :raise: Timeout

macsypy.scripts.macsyfinder.**get_version_message**() → str

Returns
the long description of the macsyfinder version

macsypy.scripts.macsyfinder.**likely_systems_to_tsv**(*models_fam_name: str*, *models_version: str*, *likely_systems: list[*macsypy.system.LikelySystem*]*, *hit_system_tracker:* HitSystemTracker, *sys_file: IO*) → None

print likely systems occurrences (from unordered replicon) in a file in tabulated separeted value (tsv) format

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **likely_systems** – list of systems found
- **hit_system_tracker** – a filled HitSystemTracker.
- **sys_file** – The file where to write down the systems occurrences

Returns
None

macsypy.scripts.macsyfinder.**likely_systems_to_txt**(*models_fam_name: str*, *models_version: str*, *likely_systems: list[*macsypy.system.LikelySystem*]*, *hit_system_tracker:* HitSystemTracker, *sys_file: IO*)

print likely systems occurrences (from unordered replicon) in a file in text human readable format

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **likely_systems** – list of systems found
- **hit_system_tracker** – a filled HitSystemTracker.
- **sys_file** – file object

Returns
None

macsypy.scripts.macsyfinder.**list_models**(*args: Namespace*) → str

Parameters
args – The command line argument once parsed

Returns
a string representation of all models and submodels installed.

macsypy.scripts.macsyfinder.**loners_to_tsv**(*models_fam_name: str*, *models_version: str*, *systems: list[*macsypy.system.System*]*, *sys_file: IO*)

get loners from valid systems and save them on file

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)

- **models_version** – the version of the models
- **systems** – the systems from which the loners are extract
- **sys_file** – the file where loners are saved

macsypy.scripts.macsyfinder.**main**(*args: list[str] | None = None*, *loglevel: Optional[Union[Literal['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'], int]] = None*)

main entry point to MacSyFinder do some check before to launch `main_search_systems()` which is the real function that perform a search

Parameters

- **args** – the arguments passed on the command line without the program name
- **loglevel** – the output verbosity

macsypy.scripts.macsyfinder.**multisystems_to_tsv**(*models_fam_name: str*, *models_version: str*, *systems: list[*macsypy.system.System*]*, *sys_file: IO*)

get multisystems from valid systems and save them on file

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **systems** – the systems from which the loners are extract
- **sys_file** – the file where multisystems are saved

macsypy.scripts.macsyfinder.**parse_args**(*args: list[str]*) → tuple[argparse.ArgumentParser, argparse.Namespace]

Parameters
args – The arguments provided on the command line

Returns
The arguments parsed

macsypy.scripts.macsyfinder.**rejected_candidates_to_tsv**(*models_fam_name: str*, *models_version: str*, *rejected_candidates: list[*macsypy.system.RejectedCandidate*]*, *cand_file: IO*, *skipped_replicons: list[str] | None = None*)

print rejected clusters in a file

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **rejected_candidates** – list of candidates which does not contitute a system
- **cand_file** – The file where to write down the rejected candidates
- **skipped_replicons** – the replicons name for which msf reach the timeout

Returns
None

macsypy.scripts.macsyfinder.**rejected_candidates_to_txt**(*models_fam_name: str*, *models_version: str*, *rejected_candidates: list[*macsypy.system.RejectedCandidate*]*, *cand_file: IO*, *skipped_replicons: list[str] | None = None*)

print rejected clusters in a file

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **rejected_candidates** – list of candidates which does not contitute a system
- **cand_file** – The file where to write down the rejected candidates
- **skipped_replicons** – the replicons name for which msf reach the timeout

Returns

None

macsypy.scripts.macsyfinder.**search_systems**(*config: Config*, *model_registry: ModelRegistry*, *models_def_to_detect: list[*macsypy.registries.DefinitionLocation*]*, *logger: Logger*) → tuple[list[*macsypy.system.System* | *macsypy.system.LikelySystem*], list[*macsypy.system.RejectedCandidate*] | list[*macsypy.system.UnlikelySystem*]]

Do the job, this function is the orchestrator of all the macsyfinder mechanics at the end several files are produced containing the results

- macsyfinder.conf: The set of variables used to runt this job
- macsyfinder.systems: The list of the potential systems
- **macsyfinder.rejected_cluster: The list of all clusters and clusters combination** which has been rejected and the reason
- macsyfinder.log: the copy of the standard output

Parameters

- **config** (*macsypy.config.Config* object) – The MacSyFinder Configuration
- **model_registry** (*macsypy.registries.ModelRegistry* object) – the registry of all models
- **models_def_to_detect** (list of *macsypy.registries.DefinitionLocation* objects) – the definitions to detect
- **logger** (`colorlog.Logger` object) – The logger use to display information to the user. It must be initialized. see `macsypy.init_logger()`

Returns

the systems and rejected clusters found

Return type

([*macsypy.system.System*, ...], [`macsypy.cluster.RejectedCandidate`, ...])

macsypy.scripts.macsyfinder.**solutions_to_tsv**(*models_fam_name: str*, *models_version: str*, *solutions: list[*macsypy.solution.Solution*]*, *hit_system_tracker:* HitSystemTracker, *sys_file: IO*, *skipped_replicons: list[str] | None = None*) → None

print solution in a file in tabulated format A solution is a set of systems which represents an optimal combination of systems to maximize the score.

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **solutions** – list of systems found
- **hit_system_tracker** – a filled HitSystemTracker.
- **sys_file** – The file where to write down the systems occurrences
- **skipped_replicons** – the replicons name for which msf reach the timeout

Returns

None

macsypy.scripts.macsyfinder.**summary_best_solution**(*models_fam_name: str*, *models_version: str*, *best_solution_path: str*, *sys_file: IO*, *models_fqn: list[str]*, *replicon_names: list[str]*, *skipped_replicons: list[str] | None = None*) → None

do a summary of best_solution in best_solution_path and write it on out_path a summary compute the number of system occurrence for each model and each replicon

```
replicon        model_fqn_1  model_fqn_2  ....
rep_name_1           1            2
rep_name_2           2            0
```

columns are separated by character

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **best_solution_path** (*str*) – the path to the best_solution file in tsv format
- **sys_file** – the file where to save the summary
- **models_fqn** – the fully qualified names of the models
- **replicon_names** – the names of the replicons used
- **skipped_replicons** – the replicons name for which msf reach the timeout

macsypy.scripts.macsyfinder.**systems_to_tsv**(*models_fam_name: str*, *models_version: str*, *systems: list[*macsypy.system.System*]*, *hit_system_tracker:* HitSystemTracker, *sys_file: IO*, *skipped_replicons: list[str] | None = None*) → None

print systems occurrences in a file in tabulated format

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)

- **models_version** – the version of the models
- **systems** – list of systems found
- **hit_system_tracker** – a filled HitSystemTracker.
- **sys_file** – The file where to write down the systems occurrences
- **skipped_replicons** – the replicons name for which msf reach the timeout

Returns
None

macsypy.scripts.macsyfinder.**systems_to_txt**(*models_fam_name: str*, *models_version: str*, *systems: list[*macsypy.system.System*]*, *hit_system_tracker:* HitSystemTracker, *sys_file: IO*, *skipped_replicons: list[str] | None = None*) → None

print systems occurrences in a file in human-readable format

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **systems** – list of systems found
- **hit_system_tracker** – a filled HitSystemTracker.
- **sys_file** – The file where to write down the systems occurrences
- **skipped_replicons** – the replicons name for which msf reach the timeout

Returns
None

macsypy.scripts.macsyfinder.**unlikely_systems_to_txt**(*models_fam_name: str*, *models_version: str*, *unlikely_systems: list[*macsypy.system.UnlikelySystem*]*, *sys_file: IO*)

print hits (from unordered replicon) which probably does not make a system occurrences in a file in human readable format

Parameters

- **models_fam_name** – the family name of the models (Conj, CrisprCAS, ...)
- **models_version** – the version of the models
- **unlikely_systems** – list of `macsypy.system.UnLikelySystem` objects
- **sys_file** – The file where to write down the systems occurrences

Returns
None

macsydata

This is the entrypoint to the macsydata command macsydata allow the user to manage the MacSyFinder models

macsypy.scripts.macsydata.**_find_all_installed_packages**(*models_dir: list[str] | None = None*) → *ModelRegistry*

Returns
all models installed

macsypy.scripts.macsydata.**_find_installed_package**(*pack_name: str*, *models_dir: list[str] | None = None*) → *macsypy.registries.ModelLocation* | None

search if a package names *pack_name* is already installed

Parameters
pack_name – the name of the family model to search

Returns
The model location corresponding to the *pack_name*

macsypy.scripts.macsydata.**_get_remote_available_versions**(*pack_name: str*, *org: str*) → list[str]

Ask the organization org the available version for the package pack_name :param pack_name: the name of the package :param org: The remote organization to query :return: list of available version for the package

macsypy.scripts.macsydata.**_search_in_desc**(*pattern: str*, *remote:* RemoteModelIndex, *packages: list[str]*, *match_case: bool = False*) → tuple[str, str, str]

Parameters

- **pattern** – the substring to search packages descriptions
- **remote** – the uri of the macsy-models index
- **packages** – list of packages to search in
- **match_case** – True if the search is case-sensitive, False otherwise

Returns

macsypy.scripts.macsydata.**_search_in_pack_name**(*pattern: str*, *remote:* RemoteModelIndex, *packages: list[str]*, *match_case: bool = False*) → list[tuple[str, str, dict]]

Parameters

- **pattern** – the substring to search packages names
- **remote** – the uri of the macsy-models index
- **packages** – list of packages to search in
- **match_case** – True if the search is case-sensitive, False otherwise

Returns

macsypy.scripts.macsydata.**build_arg_parser**() → ArgumentParser

Build argument parser.

macsypy.scripts.macsydata.**cmd_name**(*args: Namespace*) → str

Return the name of the command being executed (scriptname + operation).

Example
macsydata uninstall

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_available**(*args: Namespace*) → None
List Models available on macsy-models :param args: the arguments passed on the command line :return: None

macsypy.scripts.macsydata.**do_check**(*args: Namespace*) → None

Parameters
args – the arguments passed on the command line

Return type
None

macsypy.scripts.macsydata.**do_cite**(*args: Namespace*) → None
How to cite an installed model.

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_download**(*args: Namespace*) → str
Download tarball from remote models' repository.

Parameters
args (argparse.Namespace object) – the arguments passed on the command line

macsypy.scripts.macsydata.**do_freeze**(*args: Namespace*) → None
display all models installed with their respective version, in requirement format.

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_help**(*args: Namespace*) → None
Display on stdout the content of readme file if the readme file does not exist display a message to the user see macsypy.package.help()

Parameters
args – the arguments passed on the command line (the package name)

Returns
None

Raises
ValueError – if the package name is not known.

macsypy.scripts.macsydata.**do_info**(*args: Namespace*) → None
Show information about installed model.

Parameters
args – the arguments passed on the command line

Raises
ValueError – if the package is not found locally

macsypy.scripts.macsydata.**do_init_package**(*args: Namespace*) → None
Create a template for data package

- skeleton for metadata.yml
- definitions directory with a skeleton of models.xml
- profiles directory

- skeleton for README.md file
- COPYRIGHT file (if holders option is set)
- LICENSE file (if license option is set)

Parameters
args – The parsed commandline subcommand arguments

Returns
None

macsypy.scripts.macsydata.**do_install**(*args: Namespace*) → None

Install new models in macsyfinder local models repository.

Parameters
args – the arguments passed on the command line

Raises
- **RuntimeError** – if there is problem is installed package
- **ValueError** – if the package and/or version is not found

macsypy.scripts.macsydata.**do_list**(*args: Namespace*) → None

List installed models.

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_search**(*args: Namespace*) → None

Search macsy-models for Model in a remote index. by default search in package name, if option -S is set search also in description by default the search is case-insensitive except if option –match-case is set.

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_show_definition**(*args: Namespace*) → None

display on stdout the definition if only a package or sub-package is specified display all model definitions in the corresponding package or subpackage

for instance

TXSS+/bacterial T6SSii T6SSiii

display models *TXSS+/bacterial/T6SSii* and *TXSS+/bacterial/T6SSiii*

TXSS+/bacterial all or *TXSS+/bacterial*

display all models contains in *TXSS+/bacterial subpackage*

Parameters
args – the arguments passed on the command line

macsypy.scripts.macsydata.**do_uninstall**(*args: Namespace*) → None

Remove models from macsyfinder local models repository.

Parameters
args – the arguments passed on the command line

Raises
ValueError – if the package is not found locally

macsypy.scripts.macsydata.**get_version_message**() → str

Returns
the long description of the macsyfinder version

Return type
str

macsypy.scripts.macsydata.**init_logger**(*level: Union[Literal['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'], int] = 'INFO'*, *out: bool = True*) → Logger

Parameters

- **level** – The logger threshold could be a positive int or string among: 'CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG'
- **out** – if the log message must be displayed

Returns
logger

macsypy.scripts.macsydata.**main**(*args: list[str] = None*) → None

Main entry point.

Parameters
args – the arguments passed on the command line (before parsing)

macsypy.scripts.macsydata.**verbosity_to_log_level**(*verbosity: int*) → int

transform the number of -v option in loglevel :param verbosity: number of -v option on the command line :return: an int corresponding to a logging level

macsyconfig

Entrypoint for macsyconfig command which generate a MacSyFinder config file

class macsypy.scripts.macsyconfig.**ConfigParserWithComments**(*defaults=None*, *dict_type=<class 'dict'>*, *allow_no_value=False*, ***, *delimiters=('=', ':')*, *comment_prefixes=('#', ';')*, *inline_comment_prefixes=None*, *strict=True*, *empty_lines_in_values=True*, *default_section='DEFAULT'*, *interpolation=<object object>*, *converters=<object object>*)

Extend ConfigParser to allow comment in serialization

add_comment(*section: str*, *option: str*, *comment: str*, *comment_nb: int = count(1)*, *add_space_before: bool = False*, *add_space_after: bool = True*) → None

Write a comment in .ini-format (start line with #)

Parameters

- **section** – the name of the section
- **option** – the name of the option
- **comment** – the comment linked to this option
- **comment_nb** – the identifier of the comment by default an integer

- **add_space_before** –
- **add_space_after** –

write(*file: IO*) → None

Write an .ini-format representation of the configuration state.

Parameters

file (*file*) – the file object wher to write the configuration

class macsypy.scripts.macsyconfig.**Theme**(*ERROR: str = '\x1b[1m\x1b[31m'*, *WARN: str = '\x1b[33m'*, *SECTION: str = '\x1b[35m'*, *RESET: str = '\x1b[0m'*, *RETRY: str = '\x1b[33m'*, *QUESTION: str = '\x1b[32m'*, *EMPHASIZE: str = '\x1b[1m'*, *EXPLANATION: str = '\x1b[0m'*, *DEFAULT: str = '\x1b[1m\x1b[32m'*)

Handle color combination to highlight interactive question

__delattr__(*name*)

Implement delattr(self, name).

__eq__(*other*)

Return self==value.

__hash__()

Return hash(self).

__init__(*ERROR: str = '\x1b[1m\x1b[31m'*, *WARN: str = '\x1b[33m'*, *SECTION: str = '\x1b[35m'*, *RESET: str = '\x1b[0m'*, *RETRY: str = '\x1b[33m'*, *QUESTION: str = '\x1b[32m'*, *EMPHASIZE: str = '\x1b[1m'*, *EXPLANATION: str = '\x1b[0m'*, *DEFAULT: str = '\x1b[1m\x1b[32m'*) → None

__repr__()

Return repr(self).

__setattr__(*name*, *value*)

Implement setattr(self, name, value).

__weakref__

list of weak references to the object

macsypy.scripts.macsyconfig.**_validator**(*cast_func: Callable*, *raw: Any*, *default: Any*, *sequence: bool = False*) → Any

Parameters

- **cast_func** – the function which will cast the raw value
- **raw** – the raw value
- **default** – the default value
- **sequence** – True if the value is a sequence, False otherwise

Returns

The cast Value

Raises

MacsypyError – if the raw value cannot be cast

macsypy.scripts.macsyconfig.**ask**(*question: str*, *validator: Callable*, *default: Any = None*, *expected: Any = None*, *explanation: str = ''*, *sequence: bool = False*, *question_color: str | None = None*, *retry: int = 2*)

ask a question on the terminal and return the user response check if the user response is allowed (right type, among allowed values, ...)

Parameters

- **question** – The question to prompt to the user on the terminal
- **validator** – what validator to be used to check the user response
- **default** – the default value
- **expected** – the values allowed (can be a list of value
- **explanation** – some explanation about the option
- **sequence** – True if the parameter accept a sequence of value (comma separated values)
- **question_color** – the color of the question display to the user
- **retry** – The number of time to repeat the question if the response is rejected

Returns
the value casted in right type

macsypy.scripts.macsyconfig.**check_bool**(*raw: str*, *default: bool*, *expected*, *sequence: bool = False*) → bool

Check if value can be cast in str

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_choice**(*raw: str*, *default: str*, *expected: list[str]*, *sequence: bool = False*) → str

Check if value is in list of expected values

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – the allowed values for this option
- **sequence** – True if parameter accept a sequence of value, False otherwise

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_dir**(*raw: str*, *default: str*, *expected*, *sequence: bool = False*) → str

Check if value point to a directory

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_exe**(*raw: str*, *default: str*, *expected*, *sequence: bool = False*) → str

Check if value point to an executable

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_file**(*raw: str*, *default: str*, *expected*, *sequence: bool = False*) → str

Check if value point to a file

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_float**(*raw: str*, *default: float*, *expected*, *sequence: bool = False*) → float

Check if value can be cast in float

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_positive_int**(*raw: str*, *default: int*, *expected*, *sequence: bool = False*) → int

Check if value can be cast in integer >=0

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**check_str**(*raw: str*, *default: str*, *expected*, *sequence: bool = False*) → str

Check if value can be cast in str

Parameters

- **raw** – the value return by the user
- **default** – the default value for the option
- **expected** – not used here to have the same signature for all check_xxx functions

Returns
value

Raises
MacsypyError – if the value cannot be cast in right type

macsypy.scripts.macsyconfig.**epilog**(*path: str*) → str

Returns
the text to the user before to start the configuration

macsypy.scripts.macsyconfig.**main**(*args: list[str] | None = None*) → None

The main entrypoint of the script

Parameters
args – the command line arguments.

macsypy.scripts.macsyconfig.**parse_args**(*args: list[str]*) → Namespace

parse command line

Parameters
args – the command line arguments

Returns

macsypy.scripts.macsyconfig.**prolog**() → str

Returns
the text displayed to the user when the configuration file is generated

macsypy.scripts.macsyconfig.**serialize**(*config:* ConfigParserWithComments, *path: str*) → None

save the configuration on file

Parameters

- **config** – the config to save

- **path** (*str*) – where to store the configuration

macsypy.scripts.macsyconfig.**set_base_options**(*config:* ConfigParserWithComments, *defaults:* MacsyDefaults, *use_defaults: bool = False*) → None

Options for base section

Parameters

- **config** – The config to setup
- **defaults** – the macsyfinder defaults values
- **use_defaults** (*bool*) – If True do not ask any question use the defaults values

macsypy.scripts.macsyconfig.**set_general_options**(*config:* ConfigParserWithComments, *defaults:* MacsyDefaults, *use_defaults: bool = False*) → None

Options for general section

Parameters

- **config** – The config to setup
- **defaults** – the macsyfinder defaults values
- **use_defaults** (*bool*) – If True do not ask any question use the defaults values

macsypy.scripts.macsyconfig.**set_hmmer_options**(*config:* ConfigParserWithComments, *defaults:* MacsyDefaults, *use_defaults: bool = False*) → None

Options for hmmer section

Parameters

- **config** – The config to setup
- **defaults** – the macsyfinder defaults values
- **use_defaults** (*bool*) – If True do not ask any question use the defaults values

macsypy.scripts.macsyconfig.**set_path_options**(*config:* ConfigParserWithComments, *defaults:* MacsyDefaults, *use_defaults: bool = False*) → None

Options for directories section

Parameters

- **config** – The config to setup
- **defaults** – the macsyfinder defaults values
- **use_defaults** (*bool*) – If True do not ask any question use the defaults values

macsypy.scripts.macsyconfig.**set_score_options**(*config:* ConfigParserWithComments, *defaults:* MacsyDefaults, *use_defaults: bool = False*) → None

Options for scoring section

Parameters

- **config** – The config to setup
- **defaults** – the macsyfinder defaults values
- **use_defaults** (*bool*) – If True do not ask any question use the defaults values

macsypy.scripts.macsyconfig.**set_section**(*sec_name: str, options: dict[slice(<class 'str'>, typing.Any, None)], config: ~macsypy.scripts.macsyconfig.ConfigParserWithComments, defaults: ~macsypy.config.MacsyDefaults, use_defaults: bool = False*) → *ConfigParserWithComments*

iter over options of a section ask question for each option and set this option in the config

Parameters

- **sec_name** – the name of the section
- **options** – a dictionnary with the options to set up for this section
- **config** – The config to fill in.
- **defaults** – the macsyfinder defaults values
- **use_defaults** – The user skip this section so use defaults to set in config object

Returns
configuration

macsyprofile

class macsypy.scripts.macsyprofile.**HmmProfile**(*gene_name: str*, *gene_profile_lg: int*, *hmmer_output: str*, *cfg:* Config)

Handle the HMM output files

__init__(*gene_name: str*, *gene_profile_lg: int*, *hmmer_output: str*, *cfg:* Config)

Parameters

- **gene_name** – the name of the gene corresponding to the profile search reported here
- **hmmer_output** – The path to the raw Hmmer output file
- **cfg** – the configuration object

__weakref__
list of weak references to the object

_build_my_db(*hmm_output: str*) → dict[slice(<class 'str'>, None, None)]
Build the keys of a dictionary object to store sequence identifiers of hits.

Parameters
hmm_output – the path to the hmmsearch output to parse.

Returns
a dictionary containing a key for each sequence id of the hits

_fill_my_db(*db: dict[slice(<class 'str'>, tuple[int, int], None)]*) → None
Fill the dictionary with information on the matched sequences

Parameters
db – the database containing all sequence id of the hits.

_hit_start(*line: str*) → bool

Parameters
line – the line to parse

Returns
True if it's the beginning of a new hit in Hmmer raw output files. False otherwise

_parse_hmm_body(*hit_id: str*, *gene_profile_lg: int*, *seq_lg: int*, *coverage_threshold: float*, *replicon_name: str*, *position_hit: int*, *i_evalue_sel: float*, *b_grp: list[list[str]]*) → list[*macsypy.hit.CoreHit*]

Parse the raw Hmmer output to extract the hits, and filter them with threshold criteria selected ("coverage_profile" and "i_evalue_select" command-line parameters)

Parameters

- **hit_id** – the sequence identifier
- **gene_profile_lg** – the length of the profile matched
- **seq_lg** – the length of the sequence
- **coverage_threshold** – the minimal coverage of the profile to be reached in the Hmmer alignment for hit selection.
- **replicon_name** – the identifier of the replicon
- **position_hit** – the rank of the sequence matched in the input dataset file
- **i_evalue_sel** – the maximal i-evalue (independent evalue) for hit selection
- **b_grp** – the Hmmer output lines to deal with (grouped by hit)

Returns
a sequence of hits

_parse_hmm_header(*h_grp: str*) → str

Parameters
h_grp – the sequence of string return by groupby function representing the header of a hit

Returns
the sequence identifier from a set of lines that corresponds to a single hit

parse() → list[*macsypy.scripts.macsyprofile.LightHit*]

parse a hmm output file and extract all hits and do some basic computation (coverage profile)

Returns
The list of extracted hits

class macsypy.scripts.macsyprofile.**LightHit**(*gene_name: str*, *id: str*, *seq_length: int*, *replicon_name: str*, *position: int*, *i_eval: float*, *score: float*, *profile_coverage: float*, *sequence_coverage: float*, *begin_match: int*, *end_match: int*)

Handle hmm hits

__eq__(*other*)

Return self==value.

__hash__ = None

__init__(*gene_name: str*, *id: str*, *seq_length: int*, *replicon_name: str*, *position: int*, *i_eval: float*, *score: float*, *profile_coverage: float*, *sequence_coverage: float*, *begin_match: int*, *end_match: int*) → None

__repr__()

Return repr(self).

__str__() → str
Return str(self).

__weakref__
list of weak references to the object

macsypy.scripts.macsyprofile.**get_gene_name**(*path: str*, *suffix: str*) → str

Parameters

- **path** – The path to the hmm output to analyse
- **suffix** – the suffix of the hmm output file

Returns
the name of the analysed gene

macsypy.scripts.macsyprofile.**get_profile_len**(*path: str*) → int
Parse the HMM profile to extract the length and the presence of GA bit threshold

Parameters
path – The path to the hmm profile used to produce the hmm search output to analyse

Returns
the length, presence of ga bit threshold

macsypy.scripts.macsyprofile.**get_version_message**() → str

Returns
the long description of the macsyfinder version

macsypy.scripts.macsyprofile.**header**(*cmd: list[str]*, *model: str*, *model_vers: str*) → str

Parameters
cmd – the command use dto launch this analyse

Model
The name of model family

Model_vers
The version of the model

Returns
The header of the result file

macsypy.scripts.macsyprofile.**init_logger**(*level: Union[Literal['NOTSET', 'DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'], int] = 'INFO'*, *out: bool = True*)

Parameters

- **level** – The logger threshold could be a positive int or string among: 'CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG'
- **out** – if the log message must be displayed

Returns
logger

macsypy.scripts.macsyprofile.**main**(*args: list[str] | None = None*, *log_level: str | int | None = None*) → None
main entry point to macsyprofile

Parameters

- **args** – the arguments passed on the command line without the program name
- **log_level** – the output verbosity

macsypy.scripts.macsyprofile.**parse_args**(*args: list[str]*) → Namespace

Parameters
args – The arguments provided on the command line

Returns
The arguments parsed

macsypy.scripts.macsyprofile.**verbosity_to_log_level**(*verbosity: int*) → int

transform the number of -v option in loglevel :param verbosity: number of -v option on the command line :return: an int corresponding to a logging level

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W