sORF datafreezer - User's manual

Sébastien A. Choteau

Contents

Purpose **3**

Starting with the sORF datafreezer **3**
System requirements and dependencies . . . 3
Installing the sORF datafreezer . . . 4

Quick start **4**
Important information about the program . . . 4
Running a strategy . . . 4
Available strategies . . . 5
Mandatory options . . . 6

Log files and verbosity levels **6**
Main log file . . . 6
Log codes . . . 7
Log file dedicated to gene references problems . . . 7

Config file **7**
Description . . . 7
Database connection settings . . . 7
Data-freeze settings . . . 8
Other settings . . . 8
Example of config file . . . 9

Build new databases, check existing databases, add a release version **13**
Build new databases or check existing ones . . . 13
DatabaseCheck command line . . . 13
Add release version . . . 14
AddReleaseVersion command line . . . 14

Freeze the data sources in a DS database **15**
Insert data sources . . . 15
Information regarding the Insertion strategy . . . 15
Insertion command line . . . 15
Description of the rules of insertion . . . 16
Dealing with conflicting information about ORF properties . . . 18
Resume an insertion that failed . . . 18
Information regarding the ForceInsertion strategy . . . 18
ForceInsertion command line . . . 19
Remove data sources . . . 19
Information regarding the Deletion strategy . . . 19
Deletion command line . . . 20

Normalize data **20**
Convert the genomic coordinates (lift over) . . . 20
Information regarding the LiftOver strategy . . . 20
LiftOver command line . . . 21
Description of the rules of lift over . . . 21

Merge redundant data into unique entries **22**
Create the PRO database from a DS database . . . 22
Information regarding the Merge strategy . . . 22
Merge command line . . . 27
Resume a merging that failed . . . 28
Information regarding the ResumeMerge strategy . . . 28
Troubleshooting . . . 29
ResumeMerge command line . . . 29

Complete the missing information, normalize data and compute new information **30**
Download missing information and normalize cell contexts . . . 30
Information regarding the ComputeMissingInfo strategy . . . 31
computeMissingInfo command line . . . 31
Resume a ComputeMissingInfo that failed . . . 32
Troubleshooting . . . 32
Computing the relative coordinates . . . 32
Information regarding the ComputeRelCoord strategy . . . 32
ComputeRelCoord command line . . . 32
Resume a ComputeRelCoord that failed . . . 33
Troubleshooting . . . 33
Get the start flanking sequence and information regarding the Kozak context . . . 33
Information regarding the ComputeKozakContext strategy . . . 34
ComputeKozakContext command line . . . 34
Resume a ComputeKozakContext that failed . . . 34
Normalize the ORF categories, compute new ORF annotations . . . 34
Information regarding the AnnotateORF strategy . . . 35
AnnotateORF command line . . . 36
Resume a AnnotateORF that failed . . . 36

Filter in the database content and create a new database **36**
Information regarding the Filter strategy . . . 37
Filter command line . . . 37

Export the database content **38**
Export the ORF sequences at fasta format . . . 38
Information regarding the headers . . . 38
GenerateFastaFile command line . . . 39
Export the ORF information at BED format . . . 40
Information regarding the BED format . . . 40
Adding a track line to the file . . . 40
Information regarding the additional columns . . . 41
Converting the BED file at bigBed format . . . 42
Additional information regarding the GenerateBEDFile strategy . . . 42
GenerateBEDFile command line . . . 42
GenerateBEDContent command line . . . 43
Generate a trackDb file for track hub implementation . . . 43
Information regarding the trackDb file generated . . . 43
GenerateTrackDbFile command line . . . 44
Export the ORF information at GFF format [BETA VERSION] . . . 45
Information regarding the GFF3 file . . . 45
GenerateGFFFile command line . . . 47

Diagnosis tools **47**
Assess consistency of the database content . . . 47
Assessment of DS databases . . . 47
Assessment of PRO databases . . . 49
AssessDatabaseContent command line . . . 50
Get a summary of log files . . . 50
GenerateStatFiles command line . . . 50
Statistical analysis of the database content . . . 51

Backup, restore and convert databases **51**
Database backup . . . 51
Backup command line . . . 51
Restoring a database . . . 51
Restore command line . . . 51
Convert SQLite databases at MySQL format and vice versa . . . 52

Information to developers **52**

List of available options **53**

List of default values **55**

Additional information **56**

Authors, license and copyright **57**

Last update of the manual: 25/05/2020 **57**

Purpose

This manual is dedicated to sORF datafreezer users and aims to describe the main functions of this tool and to provide useful information regarding its use. The sORF datafreezer is able to handle data from pre-defined data sources, parse them and insert them into MySQL or SQLite databases and process these information. The processing steps includes data normalization, completion of missing information, computation of new informations as well as the export of data at convenient formats (Fasta, BED …) for further use. The sORF datafreezer come along with some diagnosis tools allowing to ensure the consistency of data, integrity of the database as well as export / import databases.

Starting with the sORF datafreezer

System requirements and dependencies

The sORF datafreezer is a software build in Python 2.7. It can be run on any operating system where dependencies have been successfully installed. A minimum of 62 GB of RAM memory, 12 threads is highly recommended to run the software. We advice to use machines with at least 40 threads, 192 GB of RAM if you intend to use all strategies. A stable connection to the Internet is also required as some information are queried from different databases accessible online.

To be able to use all functions of the program, the following dependencies are required:

- `Python 2.7`, with packages:
  - `SQLAlchemy`
  - `Pandas`
  - `PyEnsembl`
  - `PyBiomart`
  - `PyLiftOver`
  - `wget`
  - `statistics`
  - `BioPython`
  - `mysql-connector-python`
  - `pathos`
- `R`, with packages:
  - `getopt`
  - `devtools`
  - `Bioconductor`: `ensembldb`, `AnnotationHub`
- `MySQL`
- `SQLite3`

- `MUSCLE` (Multiple sequence alignment software)
- UCSC utils
  - `fetchChromSizes`
  - `bedToBigBed`

Please see the official documentation of these sotwares and packages for more information regarding their installation. A dockerfile as well as a Singularity image providing the appropriate environment are available with the source code.

Installing the sORF datafreezer

The ORF datafreezer may be run using the Singularity image provided with the code source. In such cases, Singularity needs to be installed on the computer. Please refer to the official documentation for more information about the use and installation of Singularity.

The program may also be run using the docker image provided with the source code. Please refer to the official documentation of Docker for more information about the use and installation of Docker. Please note that we do not guarantee the success of building a new docker image using the dockerfile as some external links could be out-of-date or programs source code / binaries could be no longer available.

The `PYTHONPATH` environment variable needs to be defined and point to the path to the `06_src` folder (`export PYTHONPATH=/path/to/06_src`).

In this manual, the alias `sORFdatafreezer` will be used to refer to the following command: `python $PYTHONPATH/fr/tagc/uorf/uorf.py` represents the entrypoint of the source code and must **always** be called to use any function defined in the current manual.

Quick start

Important information about the program

This program has been developed in order to build a datafreeze of short open reading frames (sORFs)-related data. It includes several **"strategies"** that may be run in order to build SQLite or MySQL databases, parse and insert data from some data sources and process the data. This manual describes all available strategies and provides the necessary information to use this program. It also provides general information about the way the data are handled within each strategy.

This datafreezer uses several databases to handle the data. Each database uses a particular model (*i.e.* a particular structure) to store the data. More precisely, the data from the sources are parsed and stored in a "**DS** database" (standing for *Data Source* database, and corresponding to an actual data-freeze of available data), while the processed data are stored in a "**PRO** database" (standing for *PROcessed* database).

New databases may be created from the PRO database, for instance by filtering all the entries on a particular list of genes or to get only the ORFs having a particular annotation (*e.g.* sORFs, uORFs…). In such cases, the resulting databases are called **FILT** databases, and follow the same model as the **PRO** database, meaning that both have the same structure but contain different data. Hence, the term "**PRO-like databases**" used in this manual refers indifferently to PRO or FILT databases. Any strategy requiring a PRO-like database may be used on both PRO and FILT databases.

The source code has been developed in order to be able to handle both MySQL and SQLite database. Nevertheless, all strategies and options have **not** been tested on SQLite databases. Hence we strongly encourage to use MySQL databases.

Running a strategy

In order to start a process, `sORFdatafreezer` needs to be followed by the name of the **strategy** to run and the options to use, using a syntax similar to the following one:

```
sORFdatafreezer [StrategyKeyword] [Options]
```

The options may be provided using indifferently short (`-h`) or long tags (`--help`). Use the tag `-h` / `--help` to display help. The command `sORFdatafreezer -h` will display on the terminal a general help and list available strategies whilst `sORFdatafreezer [StrategyKeyword] -h` will display all the options available for the strategy with a short description. Options are case-sensitive.

Available strategies

The sORF datafreezer includes several strategies that may be run independently from each other. Nevertheless, some strategies may need an other strategy to have been run successfully in order to run properly. Each strategy may require the access to one of several databases. All the strategies are described more extensively in the following sections of this manual.

The following strategies are available:

- Build databases, check existing databases, add a release version
  - **DatabaseCheck**: Build new empty DS and PRO databases; check existing databases are reachable and use the appropriate models.
  - **AddReleaseVersion**: Tag a database by adding a release version in the metadata table.
- Build a data-freeze
  - **Insertion**: Parse and insert data from a pre-defined set of sources into the DS database.
  - **Deletion**: Remove all the data related to a particular data source from the DS database.
  - **ForceInsertion**: Insert data that have already been parsed (using the **Insertion** strategy), but for which insertion in the DS database failed.
- Normalize data
  - **LiftOver**: Lift over the genomic coordinates contained in the DS database, *i.e* convert the genomic coordinates of all the DSORF and DSTranscript entries from their original annotation version to the current version.
- Merge redundant data
  - **Merge**: Build a new PRO database by merging the redundant data present in the DS database.
  - **ResumeMerge**: Resume at a particular checkpoint a merging that failed.
- Complete missing data and compute new information
  - **ComputeMissingInfo**: Complete the missing information of the PRO database by computing it or downloading it.
  - **ComputeRelCoord**: Compute ORF and CDS relative coordinates on transcripts.
  - **ComputeKozakContext**: Compute the Kozak contexts for each ORF on each known transcript.
  - **AnnotateORFStrategy**: Perform a classification of ORFs by categories (short, upstream…).
- Filter the database
  - **Filter**: Create a new **FILT** database by filtering the entries on a list of genes, cell contexts, ORF categories or ORF annotations.
- Export data
  - **GenerateFastaFile**: Generate a fasta file of all nucleic or amino acid sequences of the *ORF* or *ORFTranscriptAsso* table.
  - **GenerateBEDFile**: Generate a BED file that may be used to visualize the data in a genome browser.
  - **GenerateTrackDbFile**: Generate a trackDb file that may be used to implement track hubs compatible with UCSC and Ensembl genome browsers.
  - **GenerateGFFFile**: **[Beta-version]** Generate a GFF3 file from the content of the *ORF* table of the PRO database. Currently the GFF3 generated does not respect the sequence ontology, this should be fix in a future version.
- Diagnosis tools

  - **AssessDatabaseContent**: Check the consistency of the data contained in a database.
  - **GenerateStatFiles**: Generate a set of files that summarize information contained in logs.
- Backup and restore
  - **Backup**: Backup a DS or PRO database in '.dcorf' files.
  - **Restore**: Restore a DS or PRO database previously saved using the **Backup** strategy.

Mandatory options

For any strategy, this is necessary to provide the database(s) type using the `--databaseType` (`-T`) option. If this option is not provided by the user, the program uses SQLite database type by default (as it does not need an available MySQL server to run properly). Be aware that for any strategy using several databases at the same time (such as the **Merge** or the **Filter** strategies for instance), all the databases **must** be of the same type and located in the same folder (SQLite) or on the same server (MySQL).

Most of the strategies uses a config file that needs to be provided using the `--configfile` (`-c`) option. A section of this manual is dedicated to the format in which this file has to be provided. Please see the *Config file* section of the current manual for more information.

Verbosity level may be set using the `-v` / `--verbosity` option. See the **Log files and verbosity levels** section of the current manual for more information.

Log files and verbosity levels

Main log file

During the execution of the program, all useful information are written in log files (`execution.log`) and displayed on the console at runtime. By default, the level of verbosity is set to `info` but it is possible to set it to an other level using the `--verbose` (`-v`) option. By default, the log files may not exceed 1 MB for each. When a file reaches this value, a new one is created adding the `.n` suffix.

The following levels of verbosity are available:

- **debug**: Show all the logs generated during the execution of the program. These include extensive information about the processing of each step of the program and logs essentially dedicated to the developers. Information exclusively displayed in this mode are not essentials for most of the users.
- **info**: Show information about the execution of the program. These include information related to the major steps of the processing which are dedicated to the user and/or allowing the user to follow the process. All warning, error and critical logs are also displayed in this mode.
- **warning**: Show information related to problems that may alter the data (and eventually the processing). Messages logged at the warning level does not provoke the halt of the program but may reveal some important problems related to the operation performed by the program or related to the data itself. All warning, error and critical logs are displayed but information about the normal processing of the program are not.
- **error**: Show information about errors that happened during processing. Note that errors logged at this levels do not stop the program. Only error and critical logs are displayed in this mode.
- **critical**: Show information about critical errors that occurred and were responsible of the premature end of the program. A critical message is logged when the program has been stopped due to a major problem. When available, information about the operation that raised the error and / or a traceback are added to the message. Only critical logs are displayed when the verbosity is set to this level. Critical errors dues to the error are expected to always display a clear message about the problem (*e.g.* missing config file, incorrect path provided …). If this is not the case, please contact the developer with a copy of the log file.

If you do not know which level use, we advice to use default `info` level.

Each log is preceded by the date and precise hour at which the message has been generated as well as the level of the log (in the format `yyyy-mm-dd hh:mm:ss,sss :: VERBOSITY_LEVEL :: Message logged`).

Log codes

Most of the warning and error messages contain a unique **"log code"** allowing to easily extract the information from the log files. A "hierarchy" of log codes has been set based upon the related problem reported by the message, so this makes possible to extract all the logs related to a category of problems at the same time (using the `grep` command for instance). The list of all available log codes (warning and errors) is provided with the documentation at `.csv` format.

Log file dedicated to gene references problems

All problems related to gene references (*e.g.* when the program was not able to find in the database an unique gene corresponding to a particular alias) are logged in separated log files (called `generefwarnings.log`). Please note that this is not possible to change the level of verbosity of the messages logged of this file as it is automatically set by the program. This file is generated when a warning related to gene references appears for the first time and its creation is indicated at warning level in the main log file (`execution.log`).

In this log file, one of the following prefix is usually added prior to the message:

- **CROSSREF WARNING**: Report a warning related to the cross-references (*e.g.* when the same official symbol is used for several genes).
- **GENE UPDATE**: Report a warning related to the update of a gene or gene alias (*e.g.* when the chromosome information of a gene was missing and filled in later using information provided in a source).
- **MISSING REFERENCE**: Report a warning when the program has not been able to find a particular gene or alias.
- **NEW ENTRY**: Report a message when the program creates a new entry in the Gene or in the *GeneAlias* table. Note that the program does not log the creation of all entries during the insertion of the gene lists (*i.e.* it only logs the entries created later, such as during the insertion of data).
- **CONFLICTING INFO**: Report a warning when conflicting information have been found for the same gene (*e.g* when the same gene is found in a source with a different chromosome location than the one already registered for this gene).
- **GENE SEARCH**: Report a warning related to problems during gene search (*e.g.* when several genes have been found associated to one single alias and the program has no way to determine which one to use).

Config file

Description

Most of the strategies need a config file in order to run properly. The config file should be a file without any extension (or eventually with a `.txt` extension) and aims to provide file paths and options necessary to run the program.

The config file may contain sections embedding themselves items. The section names have to be declared with brackets (*e.g.* `[SECTION]`) whilst the items of the section are followed by an equal symbol (*e.g.* `ITEM = value`). Both section and item names are **case-sensitive**. Depending on the strategy used, some sections and items are mandatory whilst other are not (in such cases default values are used when they are missing from the config file).

Comments may be added anywhere in the config file starting the line with `#`.

```
# Example of config file
[SECTION_NAME]
ITEM_1 = value
ITEM_2 = value
```

The sections and their items are described here. See the chapter of the current manual related to the strategy you are willing to use for more information about these options.

Database connection settings

- `[DATABASE]` section: Aims to contain general information related to the database.

- DATABASE_SPECIES: Short scientific name of the species contained in the database. Only Hsapiens and Mmusculus are allowed at the moment. One single species must be provided at the time, as the program is not able to build database containing data from multiple species.
- DS_DATABASE_NAME: Name of the DS database.
- PRO_DATABASE_NAME: Name of the PRO (or PRO-like) database.
- FILT_DATABASE_NAME: Name of the FILT database.
- Items allowing connection to MySQL database:
  * DATABASE_HOST_IP: IP of the MySQL server.
  * DATABASE_PORT: Port of the MySQL server.
  * DATABASE_USER_NAME: MySQL user.
  * DATABASE_USER_PASSWD: MySQL user's password.
- Items allowing connection to SQLite database:
  * DATABASE_FOLDER: Folder where the SQLite file is located or has to be created.

Data-freeze settings

- [GENE_LISTS] section: Aims to contain information about the gene lists to insert. Each list to insert has to be provided as a new line of this section and with the **absolute** path to its associated file, using the format GeneListName = /aboslute/path/to/file.ext). Be aware that the name of the gene list (GeneListName) has to be the same (with same case) than the name of one parser. A list of available parsers with a short description is provided in the *Additional information* section of this manual. The [GENE_LIST_ORDER_OF_INSERTION] section may be filled when several gene lists are provided to insert the lists in a particular order. In such case, the name of the gene lists to insert first has to be provided using the item named GL_INSERTION_ORDER, separating each gene list name by a comma (,, the list may include spaces between each gene list) and the name of each gene list **has to be the same as the one used to provide the path to its file**.
- [DATASOURCE] section: Aims to contain information about the data to insert. Each data source has to be provided as an item of this section and with the **absolute** path to the file (using the format DataSource = /absolute/path/to/file.ext). Like for the gene lists, be aware that the name of the data source (DataSource) has to be the same (and using the same case) as the name of one parser (see *Additional information* section). The [DATA_INSERTION_ORDER] section offers the possibility to insert data in a particular order when several datasets are provided. If this option is used, the name of the sources to insert first have to be provided using the item DATA_INSERTION_ORDER, separating each source name by a comma (,, the list may include spaces between each datasource name) and the name of each source has to be the same as the one used to provide the path to its file.
- NB: When the [GENE_LIST_ORDER_OF_INSERTION] or the [DATA_ORDER_OF_INSERTION] section is provided, the program first inserts all the sources explicitly named using the option (respectively using the GL_INSERTION_ORDER and DATA_INSERTION_ORDER lists) in the appropriate order. If there are other sources provided in the config file which are not present in this list, the program will then treats them after in a random order.

Other settings

- [MERGE_PARAMETERS] section: Aims to contain parameters necessary to run the Merge strategy.
  - GENOMIC_LENGTH_DIFF_THRESHOLD: Threshold for absolute difference in genomic length. The genomic length (defined as the cumulative sum of an ORF exons) is computed for each entry of the ORF table prior and after the lift over (*i.e.* the conversion of genomic coordinates from an annotation version to the current one) and their absolute difference is calculated. This item allow to exclude all the entries that have a difference larger or equal to the provided threshold. Setting the threshold to -1 allow to ignore this option.
  - SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD: Threshold to use to include a nucleotide or an amino acid in a sequence consensus.
  - MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS: Maximum difference between the lengths of two ORF to belong to the same cluster when merging the DSORFTranscriptAsso entries (in amino acids). When setting this

parameter to 3, two DSORFTranscriptAsso will be considered as being part of two different "clusters" if the difference of their length equals or exceeds 4.

- `[COMPUTE_MISSING_INFO_PARAMETERS]` section: Aims to contain parameters to compute missing data.
  - `CELL_CONTEXTS_DICTIONARY`: A dictionary (using Python syntax) that inform how the cell contexts have to be renamed and / or merged. This dictionary should be provided using a syntax similar to the following one:

```
CELL_CONTEXTS_DICTIONARY = { 'NewContext1' : [ 'context1', 'context2' ], 'NewContext2' : [ 'context3' ] }
# Using this dictionary,
# context1 and context2 will be merged and renamed into NewContext1
# context3 will be renamed into NewContext2

# The default dictionary is available in the additional information section of the current manual
```

- `[ANNOTATE_ORF_PARAMETERS]` section: Aims to contain parameters necessary to run the ORFAnnotation strategy.
  - `CATEGORY_ASSOCIATION_DICTIONARY`: A dictionary (using Python syntax) that informs how the provided ORF categories have to be renamed and / or merged. See the **Additionnal information** section of the current manual for the default dictionary.
  - `SHORT_ORF_ANNOTATION_SIZE_THRESHOLD`: Until which length in in amino acids the ORFs have to be annotated as short.
- `[FILTERING]` section: Aims to provide parameters necessary to build a **FILT** database
  - `FILTERING_TYPE`: Should the query use an intersection or an union of the provided values? (`intersection` or `union`). This item is mandatory.
  - `GENE_LIST_FILTER`: The absolute path to a list of genes.
  - `CELL_CONTEXT_FILTER`: A comma-separated list of cellular contexts.
  - `ORF_CATEGORY_FILTER`: A comma-separated list of ORF categories.
  - `ORF_ANNOTATION_FILTER`: A comma-separated list of ORF annotations.

The sections or items that are not necessary for a strategy will be ignored. Any line starting with `#` will be considered as a comment and hence ignored.

Example of config file

```
# The structure of this file and section names should not be changed!
# See the documentation for more information.

# ============================================================================
# Information
# ============================================================================

# This is an example of config file that may be used
# with the sORF datafreezer.


# ============================================================================
# Database
# ============================================================================

## DATABASE
#  --------
# This section contains general information about the database.

[DATABASE]

# Name of the database
```

```
DS_DATABASE_NAME = Hsapiens_DS
PRO_DATABASE_NAME = Hsapiens_PRO
FILT_DATABASE_NAME = Hsapiens_PRO_uORF

# Species contained in the database
DATABASE_SPECIES = Hsapiens

# MySQL server settings
DATABASE_USER_NAME = root
DATABASE_USER_PASSWD = root
DATABASE_HOST_IP = 127.0.0.1
DATABASE_PORT = 3306




# ============================================================================
# Gene lists of reference
# ============================================================================

## GENE_LISTS
#  ----------
# This section contains the paths to the gene lists and cross-references that need to be
# parsed and for which data need to be inserted in the database.
# For a given list, provide the name of the list and the absolute path to its file.
# The name of the list has to be the same as the name of one file parser.

[GENE_LISTS]

# HGNC gene list (HGNC IDs used as unique gene IDs in the database)
HGNCGeneList = /07_input/cross_references/hsapiens_HGNC.txt



## GENE_LIST_ORDER_OF_INSERTION
#  ----------------------------
# This section has to be provided if the gene lists need to be parsed and inserted in a
# specific order. List names have to be separated with a comma and using the same name as
# in the "DATASOURCE" section. This section does not need to be provided if there is one
# single list use for cross-references.

[GENE_LIST_ORDER_OF_INSERTION]
GL_INSERTION_ORDER = HGNCGeneList




# ============================================================================
# Data sources to insert in the database
# ============================================================================

## DATASOURCE
#  ----------
# This section contains the paths to the files that need to be parsed and for which data
# need to be inserted in the database.
# For a given source, provide the name of the source and the absolute path to its file.
# The name of the source has to be the same as the name of one file parser.

[DATASOURCE]

# sORFs database (Olexiouk et al., 2017)
```

```
sORFs_org_Human = /07_input/ORF_datasources/hsapiens_sORFs.org.txt

# Erhard et al., 2018
Erhard2018 = /07_input/ORF_datasources/hsapiens_Erhard2018.csv

# Mackowiak et al., 2015
Mackowiak2015 = /07_input/ORF_datasources/hsapiens_Mackowiak2015.txt

# Johnstone et al., 2016
Johnstone2016 = /07_input/ORF_datasources/hsapiens_Johnstone2016.txt

# Laumont et al., 2016
Laumont2016 = /07_input/ORF_datasources/hsapiens_Laumont2016.csv

# Samandi et al., 2017
Samandi2017 = /07_input/ORF_datasources/hsapiens_Samandi2017.tsv



## DATA_ORDER_OF_INSERTION
#  ----------------------
# This section has to be provided if the raw data need to be parsed and inserted in a
# specific order. Sources have to be separated with a comma and using the same name as
# in the "DATASOURCE" section. The program will first treat the sources provided in this
# list in the appropriate order and then parse and insert data from the other sources if
# all of them are not present in this list. Usually this section does not need to be
# provided.

[DATA_ORDER_OF_INSERTION]
DATA_INSERTION_ORDER = sORFs_org_Human, Erhard2018, Mackowiak2015, Johnstone2016, Laumont2016, Samandi2017




# ============================================================================
# Parameters for the merging
# ============================================================================

## MERGE_PARAMETERS
#  ----------------
# This section contains parameters necessary to build the PRO database.

[MERGE_PARAMETERS]

# Threshold for the genomic length differences over which the DSORF entry
# have to be excluded from the PRO database. Use -1 to ignore this option.
GENOMIC_LENGTH_DIFF_THRESHOLD = 1

# Threshold to use for a letter (nucleotide / amino acid) to not be considered ambiguous
# (i.e. minimal proportion of sequences that needs to contains a particular letter at a
# given place to consider this letter in the consensus).
SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD = 0.66


# Maximal difference to use between the maximal and minimal lengths of DSORFTranscriptAsso
# to belong to the same group (value included).
# e.g. If max_len_diff = 3, Two DSORFTranscriptAsso will be considered as being part
#      of two different "clusters" if the difference of their length exceed 4.
MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS = 3
```

```
# ============================================================================
# Parameters for the computation of missing information
# ============================================================================

## COMPUTE_MISSING_INFO_PARAMETERS
#  ------------------------------
# This section contains parameters necessary compute missing information
# in the PRO database.

[COMPUTE_MISSING_INFO_PARAMETERS]

# The following dictionary allows to associate the cell contexts as
# provided in the data sources to "normalized" cell contexts
CELL_CONTEXTS_DICTIONARY = {
                                'BJ':               [ 'loayza_puch_2013', 'rooijers_2013', 'ji_BJ_2015' ],
                                'B_cell':           [ 'B cells' ],
                                'Blood':            [ 'mills_2016' ],
                                'Brain':            [ 'gonzalez_2014' ],
                                'Brain_tumor':      [ 'Human brain tumor', ' Human brain tumor' ],
                                'Breast':           [ 'ji_breast_2015' ],
                                'HAP1':             [ 'jakobsson_2017' ],
                                'HCT116':           [ 'crappe_2014' ],
                                'HEK293':           [ 'lee_2012', 'andreev_2015', 'sidrauski_2015', 'liu_2
                                                      'ingolia_2012', 'ingolia_2014', 'calviello_2016', 'i
                                                      'park_2017', 'zhang_2017', ' HEK293' ],
                                'HEK293T':          [ 'eichorn_2014', 'jan_2014' ],
                                'HFF':              [ 'Primary human foreskin fibroblasts (HFFs)',
                                                      'Primary human fibroblast (HFF)', 'rutkowski_2015' ]
                                'HeLa':             [ 'wang_2015', 'niu_2014', 'yoon_2014', 'liu_2013_HeLa
                                                      'park_2016', 'zur_2016', 'shi_2017' ],
                                'hES':              [ 'werner_2015', 'xu_2016' ],
                                'Jurkat':           [ 'gawron_2016' ],
                                'LCL':              [ 'cenik_2015' ],
                                'MCF7':             [ 'Loayza_Puch_2016' ],
                                'MDA-MB-231':       [ 'rubio_2014' ],
                                'MM1S':             [ 'wiita_2013' ],
                                'Monocyte':         [ 'su_2015' ],
                                'NCCIT':            [ 'grow_2015' ],
                                'RPE-1':            [ 'tanenbaum_2015', 'tirosh_2015' ],
                                'Skeletal_muscle':  [ 'wein_2014' ],
                                'THP-1':            [ 'fritsch_2012', 'stern_ginossar_2012' ],
                                'U2OS':             [ 'elkon_2015' ]
                            }


## ORF ANNOTATE_ORF_PARAMETERS
#  --------------------------
# This section contains parameter necessary compute the ORF annotations.

[ANNOTATE_ORF_PARAMETERS]

# Length in amino acids up to which an ORF may be considered as short
SHORT_ORF_ANNOTATION_SIZE_THRESHOLD = 100

# The following dictionary allows to associate the ORF categories as
# provided in the data sources to "normalized" categories
# Values associated with 'None' will be ignore and no message will
# be logged by the program for them.
CATEGORY_ASSOCIATION_DICTIONARY = { 'None':             [ 'NO_FRAME', 'other', 'ANTISENSE', 'TEC' ],
```

```
  'sORF':             [ 'sORF', 'uORF', 'uoORF', 'iORF', 'dORF' ],
  'Upstream':         [ '5UTR', '5_UTR', 'uoORF', 'uORF', 'utr5' ],
  'Overlapping':      [ 'CDS_overlap', 'uoORF' ],
  'Exonic':           [ 'EXON', 'exonic' ],
  'Intronic':         [ 'INTRON', 'iORF', 'intronic', 'RETAINED_INTRON'
  'Downstream':       [ '3UTR', '3_UTR', 'dORF', 'utr3' ],
  'Intergenic':       [ 'INTERGENIC', 'lincRNA', 'lncrna', 'ncRNA', 'pse
  'CDS':              [ 'annotated', 'Annotated', 'Isoform' ],
  'NMD':              [ 'NMD' ],
  'NSD':              [ 'NSD' ],
  'Alternative':      [ 'Out', 'alternative_frame' ],
}
```

```
# ============================================================================
# Filter settings
# ============================================================================

## FILTERING
#  ---------
# This section contains parameters necessary to create a new PRO-like
# database by filtering the content of an existing PRO database.

[FILTERING]

# Should the program use an union (OR) or an intersection (AND)
# of the provided filters?
FILTERING_TYPE = intersection

# Get all ORF that have both short and upstream annotations
ORF_ANNOTATION_FILTER = sORF, Upstream
```

Build new databases, check existing databases, add a release version

Build new databases or check existing ones

The **DatabaseCheck** strategy allows to build new empty DS and PRO databases or to check that existing ones are reachable and contain the expected structures.

If a database does not yet exist, an empty database is built using the appropriate model (tables, primary keys and relationships) and the program insert the name of the species in the (*PRO*)*SpeciesCatalog* table.

If a database of the same name exists at the same path (SQLite) or on the same server (MySQL):

- and the `--forceOverwrite` (`-f`) option has been selected, then the program first removes the existing database and then creates a new database of this name.
- and the `-f` option has not been selected, the program ensures the database is reachable and uses the expected model. Then it tries to get the name of the species contained in the *SpeciesCatalog* table and report it to the user. If the database checked does not follow the expected model (*i.e* does not have the expected tables, attributes or primary keys), then the program asks the user if it has to overwrite the existing database.

This strategy may be directly run by the user from the command line but is also automatically run by most of the other strategies prior to perform the expected tasks. When called by the user, the strategy check both the **DS** and **PRO** databases. If the `-f` option is selected, it will delete and re-built both of them.

DatabaseCheck command line

To run the DatabaseCheck strategy, use:

`sORFdatafreezer DatabaseCheck -c configFilePath [OPTIONS]`

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `DS_DATABASE_NAME` item.
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one.
- `-v`, `--verbosity`: Set the level of verbosity.

Add release version

The **AddReleaseVersion** strategy allows to tag a database by adding a version tag and description in the metadata table.

AddReleaseVersion command line

To run the AddReleaseVersion strategy, use:

`sORFdatafreezer AddReleaseVersion -N databaseName -M databaseModel -r tag -d description [OPTIONS]`

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-M`, `--databaseModel`: The schema of the database (PRO / DS).
- `-r`, `--releaseNumber`: The tag of the version.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-d`, `--releaseDescription`: The description of the version.
- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Overwrite any existing version tag / description.
- `-v`, `--verbosity`: Set the level of verbosity.

Freeze the data sources in a DS database

Insert data sources

The **Insertion** strategy allows to parse and insert a set of data sources in the database. Running this strategy automatically runs the **DatabaseCheck** strategy.

Information regarding the Insertion strategy

The program first parses and inserts data from the gene lists in order to fill in the *Gene* and *GeneAlias* tables of the DS database. When this step has been completed, it computes for each unique alias registered in the *GeneAlias* table the list of gene IDs (as registered in the *Gene* table) that may be associated to it. The associations are saved as new entries of the *UTGeneFromAlias* table, allowing to improve the computation time of future steps.

Then, it sequentially parses and insert data from the provided sources in the appropriate order (either the one defined in the config file or in a random order).

If the Insertion strategy is run a second time (without using the `-f` option) after a modification of the config file, then it skips the parsing and insertion of data that have already been inserted and only insert the new data sources. Be aware that removing a data source of a config file after its insertion will **NOT** delete it from the database and that updating the file of a data source that has already been inserted will **not** update the content of the database.

If a data source has been updated and you want to change the database, then the database must be rebuilt! You must use the **Insertion** strategy with the `-f` option in such case. Note that the **ForceInsertion** and **Deletion** strategy may be of particular interest in such cases (see the following sections of the current manual related for more information).

Insertion command line

To run the Insertion strategy, use:

```
sORFdatafreezer Insertion -c configFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `DS_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.
- `[GENE_LISTS]` section:
  - Each of the item must have the name of one of a gene list parser and may only appears once in the file.
- `[DATASOURCE]` section:
  - Each of the item must have the name of one of a data source parser and may only appears once in the file.

The following section and items of the config file may be provided:

- `[GENE_LIST_ORDER_OF_INSERTION]` section:
  - `GL_INSERTION_ORDER` item: A comma-separated list of gene list names to be inserted in a particular order. The list does not necessarily needs to contain all gene lists defined in the `[GENE_LISTS]` section but **all** gene lists declared in the current item have to be defined in the `[GENE_LISTS]` section (using the same name).
- `[DATA_ORDER_OF_INSERTION]` section:
  - `DATA_INSERTION_ORDER` item: A comma-separated list of data source to be inserted in a particular order. The list does not necessarily needs to contain all data sources defined in the `[DATASOURCE]` section but **all** data sources declared in the current item have to be defined in the `[DATASOURCE]` section (using the same name).

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.

  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one and to run the insertion.
- `-v`, `--verbosity`: Set the level of verbosity.

Description of the rules of insertion

The current section of the manual describes more extensively the rules that are used to parse and insert the data. These rules are applied for **all** the data source.

Chromosome names:

- The chromosome names are always stored in the database without the '**chr**' prefix.
- The sexual chromosome are changed to the `X|Y` value.
- Mitochondrial chromosomes are changed to `MT`.

Genomics coordinates:

All the coordinates used in the databases are registered using a "1-based, fully-close" counting system. This means that the first nucleotide on the chromosome is located at the position `1` and both start and end coordinates describing an ORF (or any other feature) are included in the feature.

When the same attribute is defined in a table (of the DS database) both with and without `raw_` prefix, the prefix starting with `raw_` contains the coordinates in the original genome annotation (*i.e.* as provided by the data source) whilst the attribute without the prefix contains the coordinates after the lift over (see the **Normalization** section of the manual for more information about the lift over). Hence, these last are left empty during the **Insertion** strategy and only computed later when running the **LiftOver** strategy.

- In the **DSORF** table:
  - The **strand** attribute corresponds to the ORF strand.
    - The ***start_pos*** attribute corresponds:
      - to the genomic coordinates of the first nucleotide of the start codon for the ORFs on the '+' strand.
      - to the genomic coordinates of the last nucleotide of the stop codon for the ORFs on the '-' strand.
    - The ***stop_pos*** attribute corresponds:
      - to the genomic coordinates of the last nucleotide of the stop codon for the ORFs on the '+' strand.
      - to the genomic coordinates of the first nucleotide of the start codon for the ORFs on the '-' strand.
    - The ***splice_starts*** attributes correspond to an underscore-separated list of the genomic coordinates of the first nucleotide of each "exon" of the ORF, in the actual order of the exons on the RNA, meaning that:
      - the first value is the coordinates of the first nucleotide of the first exon (*i.e.* of the start codon),
      - the second the coordinates of the first nucleotide of the 2nd exon,
      - etc. and this whatever the strand of the ORF is.
    - The ***splice_ends*** attributes correspond to an underscore-separated list of the genomic coordinates of the last nucleotide of each "exon" of the ORF, in the actual order of the exons on the RNA, meaning that:
      - the first value is the coordinates of the last nucleotide of the 1st exon,
      - the second value is the coordinates of the last nucleotide of the 2nd exon,
      - etc., and the last value is the coordinates of the last nucleotide of the last exon (*i.e.* of the stop codon), and this whatever the strand of the ORF is.

Sequences:

- In the *DSORFTranscriptAsso* table:
  - The ORF sequence in nucleotide (**raw_sequence**) includes both the start and the stop codons.

  - The ORF sequence in amino acids (**raw_sequence_aa**) includes the start codon but excludes the stop codon.
  - The `orf_length_nt` correspond to the ORF length in nucleotides including both the start and the stop codons.
  - The `orf_length` correspond to the ORF length in amino acids including the start codon but excluding the stop codon.

Missing transcript IDs:

- When the transcript ID is missing, a "fake" transcript ID is created by concatenating the `UNKNOWN_TRANSCRIPT_` prefix, the name of the source and the row index of the entry.

Recovering missing gene symbols, alias and IDs:

When the gene symbol is missing, `pyensembl` is used to query Ensembl databases and:

- If the transcript ID is provided, then it tries to get the corresponding gene alias,
- Otherwise,
  - if the chromosome name is provided and if the ORF strand is provided and:
    - there is one single gene overlapping with the ORF coordinates, then the ORF is associated to this gene.
    - there are several genes overlapping with the ORF coordinates, then the symbols of these genes are concatenated (or only the first and last symbols of the list if the ORF overlaps with more than two genes) to create a "fake" gene ID with the `OVERLAPPING_GENES_` prefix.
    - there is no gene overlapping with the ORF coordinates, then the program search for lncRNAs overlapping with the ORF coordinates and:
      - there is one single lncRNA overlapping with the ORF coordinates, then the alias of the lncRNA is used as gene symbol.
      - there are several lncRNAs overlapping with the ORF coordinates, then the aliases of these lncRNAs are concatenated (or only the first and last aliases of the list if the ORF overlaps with more than two lncRNAs) to create a "fake" gene ID with the `OVERLAPPING_LNCRNAS_` prefix.
      - there is no lncRNAs overlapping with the ORF coordinates, then a "fake" gene ID is created concatenating the `INTERGENIC_GENE_` prefix and the chromosome name.
  - if the chromosome name is provided but the ORF strand is not provided, then a "fake" gene ID is created concatenating the `UNKNOWN_GENE_` prefix and the chromosome name.
  - if the chromosome name is not provided, then a "fake" gene ID is created using the `UNKNOWN_GENE_chr_UNKNOWN` alias.

Identifying the appropriate gene from a symbol, alias or ID:

For any gene symbol, gene alias or gene ID, the following algorithm is used to identify the appropriate entry in the *Gene* table:

The program first get the list of all *Gene* entries that have the provided alias (or gene ID) as one of their alias (using the *UTGeneFromAlias* table). Then,

- If there is one single *Gene* entry in the list and:
  - if the chromosome name associated to the ORF is known and:
    - the chromosome name of the ORF is the same as the chromosome name of the *Gene* entry, then the ORF is associated to this entry.
    - the chromosome name of the ORF is different from the chromosome name of the *Gene* entry, then the ORF is associated to a newly created "fake" *Gene* entry with an ID obtained by concatenating the `UNKNOWN_GENE_` prefix, the alias, and the chromosome name.
  - and the chromosome name associated to the ORF is not provided, then the ORF is associated to this *Gene* entry.
- If there are several *Gene* entries in the list and:

  - if there is one single *Gene* entry on the chromosome associated to the ORF, then the the ORF is associated to this entry.
  - if there are several *Gene* entries on the chromosome associated to the ORF, then the ORF is associated to a newly created "fake" *Gene* entry with an ID obtained by concatenating the `UNKNOWN_GENE_` prefix, the alias, and the chromosome name.
  - if there is no *Gene* entry on the chromosome associated to the ORF and:
    * the chromosome associated to the ORF is known and there are *Gene* entries missing their chromosome name and:
      · there is one single *Gene* entry missing its chromosome information, then this last is updated and the ORF is associated to the *Gene* entry.
      · there are several *Gene* entries missing their chromosome information, then the ORF is associated to a newly created "fake" *Gene* entry with an ID obtained by concatenating the `UNKNOWN_GENE_` prefix, the alias, and the chromosome name.
    * the chromosome associated to the ORF is not known or if there is no *Gene* entries of the list missing their chromosome information, then the ORF is associated to a newly created "fake" *Gene* entry with an ID obtained by concatenating the `UNKNOWN_GENE_` prefix, the alias, and the chromosome name.
- Otherwise, if there is no *Gene* entry in the list, a new entry is created in the *Gene* table using the alias and chromosome information associated to the ORF in the data source.

Dealing with conflicting information about ORF properties

Any conflicting information found during the insertion of data is logged at the warning or error level in the main log files.

Sometimes, a provided transcript ID may be found associated to several *Gene* entries during the insertion of data. This may happen for instance when a line of a source file being parsed provides both the gene symbol and transcripts ID related to the ORF whilst an other line of the same source provides only the transcript ID related to an other ORF. In such cases, the second line may associate the transcript to an other *Gene* entry if the alias returned when querying the Ensembl databases associated with the provided transcript ID is not an alias registered in the database. Then the same *Transcript* entry should theoritically be associated with two different *Gene* entries. Nevertheless, as a transcript is actually expected to be related to one single gene and in order to respect the referential integrity, the program logs an error message and creates a new *Gene* entry using the prefix `CONFLICT_GENE_` for the `gene_id` attribute. To avoid reporting several times the same conflicts, all the association of the transcript ID with the list of related gene ids are saved in the `UTDSTranscriptGeneConflict` table.

Resume an insertion that failed

When the **Insertion** strategy is run, at the end of the parsing of each gene list and source and prior the insertion of data in the DS database, a file is automatically generated in order to store all the entries created during the parsing (the objects are serialized and saved in hidden files). The **ForceInsertion** strategy may be useful when the parsing happened correctly but a problem was encountered during the insertion into the database, as it allows to try again the insertion of data using the files previously generated. Obviously, the **ForceInsertion** strategy must **never** be run prior to the **Insertion** strategy and it is **absolutely necessary** to make sure the parsing happened successfully prior to run it (check the main log file to ensure the parsing was successful).

Information regarding the ForceInsertion strategy

The **ForceInsertion** strategy first loads the data from the file corresponding to the data source and try again the insertion into the DS database. Hence, running this strategy may be useful to save the time of the parsing, but be aware that trying to force the insertion of data may raise several problems or silently introduce errors in the database.

In particular:

- If you try to insert the sources in a different order that the one in which they were initially treated. Explanation: In such cases, the program may for instance create a new entry of the *Gene* table during the parsing of a first source, which will then be used by a second source; then trying to insert them in the reverse order will report a non-respect of the database integrity (SQL duplicate exception will be raised).

- If a first source (*e.g. source1*) is parsed successfully but its insertion fails, and then a second source (*e.g. source2*) is parsed *and* inserted successfully, trying to build a new empty DS database and to insert the two sources (in the same order) using the ForceInsertion strategy may cause new problems. Explanation: the program may for example create a new entry (*e.g.* in the *Gene* table) during the parsing of the first source (*source1*), which were then not available during the parsing of the second source (*source2*) and then trying to create a new entry with the same primary-key but different non-primary-key attributes, raising an exception due to duplicates in the primary keys.

Hence, in order to avoid these problems, when the insertion of data from a source failed whilst running the **Insertion** strategy, we highly recommend to:

- First, erase the existing database and rebuilt a new one, using the **DatabaseCheck** strategy **with the `--f` option**,
- Then, insert again all the gene list and data sources **properly parsed prior the first insertion fail** and in **the same order as the one in which they were initially inserted** (either the one provided in the config file or the one generated by the program - in any case this order is logged in the main log file) using the **ForceInsertion** strategy. You need there to add the `UTGeneFromAlias` as first value of the list of data sources to re-insert, *i.e.* after the last list of genes and prior the first data source.
- Finally, run again the Insertion strategy to perform (a new time) the parsing and insertion of sources that had to be inserted after the first source for which the insertion failed, using the same config file.

We also highly recommend to provide an order of insertion in the config file (see **Config file** section of the current manual) putting the largest data source first, even if this is not mandatory.

ForceInsertion command line

To run the ForceInsertion strategy, use:

```
sORFdatafreezer ForceInsertion -N databaseName -s sourceNames [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-s`, `--source`: A comma-separated list of the data source to insert in the database (without space), in the order in which they need to be inserted. You must use `UTGeneFromAlias` as a source name to re-insert data computed after the insertion of the last gene list and prior to the first data source name if you are re-inserting both gene lists and data sources.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.

Remove data sources

The **Deletion** strategy allows to delete from a database a source that has been previously parsed and inserted using the **Insertion** strategy.

Information regarding the Deletion strategy

The **Deletion** strategy allows to remove data related to a particular data source from the database. It removes all the information related to the source in the *DataSource*, *DSORF*, *DSTranscript* and *DSORFTranscriptAsso* tables.

Nevertheless, be aware that any *Gene* (and subsequent *GeneAlias*) entry generated during the insertion of the data source will not be removed.

Several data sources to remove may be provided as a comma-separated list (without any space between each data source name). The program report the number of entries expected to be deleted and ask the user to confirm the deletion prior to perform it.

You should **never** delete a data source with this strategy and insert it again using the **ForceInsertion** strategy as entries related to this source but persisting (*i.e.* *Gene* or *GeneAlias* entries) may have been updated, which could raise problems related to primary-key duplicates. Hence, if a source that has been deleted needs to be inserted again, we strongly advice to use the **Insertion** strategy.

Deletion command line

To run the Deletion strategy, use:

```
sORFdatafreezer Deletion -N databaseName -s sourceNames [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-s`, `--source`: A comma-separated list of the data source to delete from the database (without space).

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.

Normalize data

Convert the genomic coordinates (lift over)

The **LiftOver** strategy allows to convert all the genomic coordinates contained in the entries of the DS database from their original genome annotation version (the one of the data source) to the current one (GRCh38 or GRCm38).

Information regarding the LiftOver strategy

To perform the lift over of the genomic coordinates, the program uses the `PyLiftOver` package and chain files downloaded from UCSC.

Note that the original values (as provided by the data sources) are stored in the DS database under the attributes starting with the `raw_` prefix whilst the corresponding attributes not starting with this prefix contain the converted coordinates (*e.g* the `raw_start_pos` attribute of a DSORF entry contains the start position of this ORF as it is provided by the source, whilst the `start_pos` attribute contains the start position of this ORF after the lift over).

The program updates the following coordinates:

DSORF table:

- The strand, start position, stop position and splicing positions (starts and ends of exons) of the all the entries are lifted over.

- At the end of the process, the cumulative genomic length (*i.e.* the total length of the exons in nucleotides) is computed for each ORF using the coordinates in the original annotation and using the lift overed coordinates for each ORF for which the conversion succeed. These values are respectively stored in the `raw_genomic_length` and `genomic_length` attributes of the *DSORF* table. The difference between these two length is computed too and store under the `genomic_length_diff` attribute. Hence, this value may be used to ensure the right conversion of all coordinates for an ORF and is used as filtering criteria by the **Merge** strategy.

DSTranscript table:

- The strand, start position, end position, CDS start and CDS stop positions of all the entries are lifted over.

LiftOver command line

To run the LiftOver strategy, use:

```
sORFdatafreezer LiftOver -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `DS_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.

Description of the rules of lift over

The current section of the manual describes more extensively the rules that are used to lift over the genomic coordinates.

DSORF table:

- If the lift over returns a different chromosome after conversion, then a critical message is logged and the program is interrupted.
- If the strand information after conversion:
  - of the start **and** stop positions are both missing (*i.e.* the conversion failed for both of them), then the conversion is stopped and there are no new attributes registered.
  - of the start **or** of the stop positions is missing, but the other one is available (*i.e.* the conversion succeeded for only one of them), then it is check if the strand after conversion is the same as prior. If the strand changed, then the start and stop coordinates are reversed.
  - of the start and of the stop positions are provided (*i.e.* the conversion succeeded for both of them), then:
    - if the strands of the start and stop positions are the same and did not changed after conversion (*i.e.* same as the raw strand), then the new positions are registered.
    - if the strands of the start and stop positions are the same but changed after conversion (*i.e.* opposite of the raw strand), then the start and stop coordinates are reversed.

        * if the strands of the start and stop positions are different, then the conversion is stopped, an error is logged and none of the new attribute values is registered.

- For `splice_starts` and `splice_stops` attributes, if the conversion fails for one of the splicing coordinates (impossible conversion or strand that is different after conversion compared to the "raw" strand), then the conversion is stopped and the none of the new attribute is registered.

DSTranscript table:

- The chromosome of the transcript is get from the information of the gene related to this transcript.
- If the lift over returns a different chromosome after conversion, then a critical message is logged and the program is interrupted.
- If the strand information after conversion:
  - of the start **and** end positions are both missing (*i.e.* the conversion failed for both of them), then the conversion is stopped and there are no new attribute registered.
  - of the start **or** of the end positions is missing, but the other one is available (*i.e.* the conversion succeeded for only one of them), then it is check if the strand after conversion is the same as prior. If the strand changed, then the start and end coordinates are reversed.
  - of the start and of the end positions are provided (*i.e.* the conversion succeeded for both of them), then:
    * if the strands of the start and end positions are the same and did not changed after conversion (*i.e.* same as the raw strand), then the new positions are registered.
    * if the strands of the start and end positions are the same but changed after conversion (*i.e.* opposite of the raw strand), then the start and end coordinates are reversed.
    * if the strands of the start and end positions are different, then the conversion is stopped, an error is logged and none of the new attribute values is registered.
- For `cds_start_pos` and `cds_stop_pos` attributes, if the strand after conversion is different from the one of the transcript, then the conversion of this attribute is stopped and it is not registered. Otherwise, the attribute is saved and the start and stop coordinates are reversed if the strand changed during the conversion.

For all entries of *DSORF* and *DSTranscript* tables, if the data source was already using the current genome annotation, then the attributes are just copied as no conversion needs to be performed.

Merge redundant data into unique entries

Create the PRO database from a DS database

The **Merge** strategy allows to merge the redundant information into single entries by creating a new PRO database from an existing DS database. In other words a given ORF and a given transcript are registered only once in the PRO database, whilst several entries may be actually describing the same ORF or the same Transcript in the DS database due to the redundancy of some information between the data sources. The relationships between PRO and DS databases are registered in the PRO database.

Caution: This strategy needs the **LiftOver** strategy to have been run successfully to be used.

Information regarding the Merge strategy

An empty PRO database is first build and the merging the consists of 6 successive important steps: - the copy of conserved information - the merging of the ORF entries - the merging of the Transcript entries - the resolution of the associations between the ORFs and the Transcripts - the merging of the ORF-Transcript associations - a final step to clean the database

These steps are described more extensively below.

Copying conserved information All the metadata contained in the DS database are copied as is in the PRO database. This includes at least the genome version and the species scientific short name.

All the information related to the genes and their alias (symbol / alias / IDs) are copied in the PRO database.

Merging the ORFs Two entries of the *DSORF* table are considered as actually describing the same ORF when all the following conditions are met:

- they are located on the same chromosome
- they are located on the same strand
- they have the same start position
- they have the same stop position
- they have the same number of exons
- their exons have the same starts and ends

All the coordinates here are absolute genomics coordinates in the current annotation version (*i.e.* after lift over).

When the `GENOMIC_LENGTH_DIFF_THRESHOLD` item is set in the config file, all the DSORF entries that have a `genomic_length_diff` equal or over the provided value are discarded of the list of ORFs to look for when performing the merging (see the section about the **LiftOver** strategy for more information about this attribute). If this parameter is set to `-1`, then this option will be ignored (`0` by default).

The merging of the ORFs is solved in three successive steps.

First all the ORFs:

- that share the same chromosome, strand, start and stop positions, number of exons and start and end position of all exons,
- **and** for which **all** of these attributes are known,

are merged together.

As all the information required to properly characterize an ORF are known, this merging is considered **unambiguous**.

Then all the remaining ORFs:

- that share the same chromosome, strand, start and stop positions, number of exons and start and end position of all exons,
- **and** for which the chromosome, the strand and the start position are known **and**
  - that are unspliced **or**
  - that are spliced but for which the start and/or stop positions of the exons are missing,

are compared with the ORFs previously merged and added to the PRO database:

- If one of the entry of the PRO database matches with the current ORFs, then they are merged with it,
- Otherwise, if there is not any entry of the PRO database matching with the current ORFs, then they are merged together and a new entry is created in the PRO database.

As the ambiguity is solely related to the coordinates of the exons and all the most important properties to characterize an ORF are known, this merging is considered **unambiguous**.

Finally all the remaining ORFs:

- that share the same chromosome, strand, start and stop positions, number of exons and start and end position of all exons,
- **and** for which the chromosome is known **and**
  - for which the strand is unknown and the start position is unknown,
  - **or** for which the strand is unknown and the stop position is unknown,
  - **or** for which both the start and stop positions are unknown
- **and** on which there is no constraint regarding the splicing information (number and coordinates of exons may eventually be missing)

are compared with the ORFs previously merged and added to the PRO database:

- If one of the entry of the PRO database matches with the current ORFs, then they are merged with it,
- Otherwise, if there is not any entry of the PRO database matching with the current ORFs, then they are discarded from the PRO database as there are missing at least one of the important information necessary to characterize properly the ORF.

As there are some important information missing for these ORFs, this merging is considered **ambiguous**. Note that due to this ambiguity, a same DSORF entry could actually be merged with different ORF entries in the PRO database.

NB: - The eventual entries of the DS database not previously described (*e.g.* missing both their strand, start and stop positions) are actually missing too many crucial information to be considered for the merging. Hence these are not used to create the PRO database. - The `count_ds` and `count_ds_ambiguous` attributes of a *ORF* entry respectively inform the number of DSORF entries merged in an unambiguous and in an ambiguous way. - The *ORFDSAsso* table records for each entry of the *ORF* table the IDs of the *DSORF* table that have been merged together to create it.

Merging the transcripts Two entries of the *DSTranscript* table are considered as actually describing the same transcript when all the following conditions are met:

- they have the same "official" ID (such as Ensembl ID)
- they are related to the same gene

The merging of the transcripts is solved in two successive steps:

First all the transcripts that have a gene ID which is not a "fake" transcript (*i.e.* which ID is not `UNKNOWN_TRANSCRIPT` and does not start by this prefix) are merged together. This merging is considered **unambiguous**.

Then all the remaining transcripts (*i.e.* the ones with a "fake" transcript ID) that are related to the same gene and share the same strand, start position, position and CDS start and end positions are compared with the transcripts previously merged and added to the PRO database:

- If one of the entry of the PRO database matches with the current transcripts, then they are merged with it,
- Otherwise, if there is not any entry of the PRO database matching with the current transcripts, then they are merged together and a new entry is created in the PRO database. This merging is considered **ambiguous**.

NB: - We decided not to use the genomics coordinates as all of the data sources currently used do not provide the transcript coordinates. - The `count_ds` and `count_ds_ambiguous` attributes of a *Transcript* entry respectively inform the number of DSTranscript entries merged in an unambiguous and in an ambiguous way. - The *TranscriptDSAsso* table records for each entry of the *Transcript* table the IDs of the *DSTranscript* table that have been merged together to create it.

Finding out which DSORFTranscriptAsso entries to merge At this stage of the merging, the similar ORFs have been merged and inserted in the PRO database and the similar transcripts have been merged and inserted in the PRO database too. Hence each "biological feature" (ORF, transcript or gene) is recorded one unique and single time in the PRO database, whatever the number of time it has been described by the original data sources.

Nevertheless, the relationship between the ORFs and the transcripts has not yet been recorded in the PRO database at this stage. In particular, it has to be noticed that an ORF can be located on several transcripts whilst a transcript can obviously harbor several ORFs. In the DS database, this level of information is recorder in the *DSORFTranscriptAsso* table. Thus, the information from this table needs to be summarized and registered in the PRO database.

The following steps are performed:

1. First, for each entry of the *DSORFTranscriptAsso* table, the ID of the unique *DSORF* entry and the ID of the unique *DSTranscript* ID related to it are get.
2. Then, knowing that a *DSORF* entry may have been either totally discarded from the PRO database, or used one single time during the merging (unambiguous merging) or used several times during the merging (ambiguous merging), the list of the *ORF* entrie(s) that derives from this *DSORF* entry are get.
3. Similarly to the previous step, knowing that a *DSTranscript* entry may have been either totally discarded from the PRO database, or used one single time during the merging (unambiguous merging) or used several times during the merging (ambiguous merging), the list of the *Transcript* entrie(s) that derives from this *DSTranscript* entry are get.
4. Thus, at this stage we are able to associate to each *DSORFTranscriptAsso* ID the list of *ORF* and *Transcript* entries that may be related to it. The cartesian product, define as the list of all (*ORF* ID, *Transcript* ID) that may exist, is computed from these list.
5. Using the result of the previous steps for **all** *DSORFTranscriptAsso* entries, we associate to each unique (*ORF* ID, *Transcript* ID) the list of *DSORFTranscriptAsso* IDs that register their interaction.
6. All the *DSORFTranscriptAsso* of the same list are merged together (see next main step of the algorithm).

Example

For more clarity about this algorithm, here is an example. We assume that: - The *DSORFTranscriptAsso* with ID `DSOTA_1` is related to the *DSORF* entry with ID `DSORF_2` and with the *DSTranscript* entry with ID `DSTranscript_3`. - The *DSORFTranscriptAsso* with ID `DSOTA_4` is related to the *DSORF* entry with ID `DSORF_5` and with the *DSTranscript* entry with ID `DSTr_6`. - The *DSORF* with ID `DSTranscript_6` and the *DSORF* with ID `DSORF_5` have been merged together and the *ORF* entry with ID `ORF_7`derives from them. - The *DSTranscript* with ID `DSTranscript_3` and the *DSTranscript* with ID `DSTranscript_6` have been merged together and the *Transcript* entry with ID `Transcript_8` derives from it.

```
# NB: The following lines of codes are written in pseudo-language for more clarity

# Loop over the DSORFTranscriptAsso entries
for dsorftranscriptasso_id in ( DSOTA_1, DSOTA_2 ):

    # --------------- Let's see what happen when dsorftranscriptasso_id equals DSOTA_1
    # 1. Get the ID of the unique DSORF entry related to it and
    #    the ID of the unique DSTranscript entry related to it
    dsorf_id = DSORF_2
    dstranscript_id = DSTranscript_3

    # 2. Get the list of ORF IDs that derives from this DSORF
    orf_ids = [ ORF_7 ]

    # 3. Get the list of Transcript IDs that derives from this DSTranscript
    transcript_ids = [ Transcript_8 ]

    # 4. Compute the cartesian product of ORF IDs and Transcript IDs
    cartesian_prod = orf_ids x transcript_ids
    cartesian_prod = [ ORF_7 ] x [ Transcript_8 ]
    cartesian_prod = [ (ORF_7, Transcript_8) ]

    # We keep the result of this algorithm in memory


    # Restart the previous algorithm with the next DSORFTranscriptAsso ID


    # --------------- Let's see what happen when dsorftranscriptasso_id equals DSOTA_4
    # 1bis. Get the ID of the unique DSORF entry related to it and
    #    the ID of the unique DSTranscript entry related to it
    dsorf_id = DSORF_5
    dstranscript_id = DSTranscript_6

    # 2bis. Get the list of ORF IDs that derives from this DSORF
    orf_ids = [ ORF_7 ]

    # 3bis. Get the list of Transcript IDs that derives from this DSTranscript
    transcript_ids = [ Transcript_8 ]

    # 4bis. Compute the cartesian product of ORF IDs and Transcript IDs
    cartesian_prod = orf_ids x transcript_ids
    cartesian_prod = [ ORF_7 ] x [ Transcript_8 ]
    cartesian_prod = [ (ORF_7, Transcript_8) ]

    # We keep the result of this algorithm in memory


# 5. We collect the information from the for loop to associate to each unique (ORF ID, Transcript ID) the
(ORF_7, Transcript_7) : [ DSOTA_1, DSOTA_4 ]

# Hence, at the end of this process, we know that the DSORFTranscriptAsso with IDs 1 and 4 needs to be mer
```

Merge the DSORFTranscriptAsso Once the *DSORFTranscriptAsso* to merge together have been identified (see previous section), it is then necessary to merge them.

When the `--checkDSOTA` option has been selected, the consistency of the information registered **in a single** *DSORFTranscriptAsso* entry is assessed. This optional step will check the following attributes:

- The agreement between the nucleic and amino acid sequences.
- The agreement between the provided nucleic sequence and length.
- The agreement between the provided amino acid sequence and length.
- The agreement between the nucleic and amino acid lengths.

And any inconsistent information will be removed from the entry and the operation will be logged in the main log file. Be aware that selecting this option could be highly time-consuming; thus we advice not to use it with the six original data sources when you cannot use more than 20 threads.

If there is one single entry in the list of *DSORFTranscriptAsso* entries to merge, then the algorithm just basically copy the information registered in it in a new *ORFTranscriptAsso* entry.

Otherwise, when there are several *DSORFTranscriptAsso* entries to merge together, the following steps are performed:

- The identification methods (bioinformatics prediction, Ribo-seq, MS) that detected the ORF - Transcript relationship is summarized (set to `True` if detected at least once with the method).
- If the `--computeConsensus` option has been selected, a consensus of the amino acid and nucleic sequences is computed (aligning the sequences with MUSCLE and computing the consensus with the `Bio.Align` methods of the `BioPython` package). If it is not selected then the sequence is replaced by an integer representing the number of sequences that would have been available to compute the consensus. The `SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD` item may be provided as a float in the database and will be used to compute the consensus. For each position in the sequence, if the percentage of the most common residue is greater than the threshold, then this residue will be considered as being the consensus for the position (see the documentation of the `threshold` parameter in the `gap_consensus()` function of `Bio.Align` module for more information). Be aware that selecting this option could be highly time-consuming; thus we advice not to use it with the six original data sources when you cannot use more than 20 threads.
- A consensus of the start codon sequence is computed.
- The Kozak context information is summarized. If the Kozak context has been reported by at least one of the data sources, then it is registered as existing. Be aware that this is an information that is computed using the information extracted from the data sources; an other Kozak context may be computed later using our own algorithm (see the **Complete the missing information and compute new information** section of the current manual).
- A summary of the lengths and various scores (PhyloCSF, ORF score, PhastCons, FLOSS) is made by getting the minial and maximal values, by computing the median and by registering the list of values.
- "Clusters" of *DSORFTranscriptAsso* entries are identified. Due to the high complexity of the algorithm previously described to merge the ORF, the transcripts and then identify the *DSORFTranscriptAsso* that need to be merge together, it may sometimes happen that *DSORFTranscriptAsso* entries that are actually not describing the same ORF - Transcript relationship are merged together. Hence, it is important to have a way to identify these cases and get a confidence about the quality of the merging. As the length is one of the most important feature available to characterize an ORF, we decided to establish "clusters" or *DSORFTranscriptAsso* entries based on their length. Briefly, *DSORFTranscriptAsso* entries that have a difference in their ORF length that exceeds the `MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS` value (in amino acids, that may be defined in the config file) will belong to two different clusters. For instance if this value is set to `3`, then as soon as the difference between two lengths equal or exceeds `4` amino acids, then they will belong to different clusters. As a consequence, a number of clusters equal to `1` suggests that the selection of *DSORFTranscriptAsso* entries to be merged together was probably relevant. On a contrary, a high value may suggest that the merging was probably irrelevant; and we should be worried by a large amount of *ORFTranscript* entries for which the number of cluster computed does not equals `1`. The number of computed cluster and their composition are registered in the *ORFTranscriptAsso* attribute. As this computation is exclusively based on the length in amino acids, the number of lengths used to performed this computation is registered too. Indeed, with this algorithm clusters of *DSORFTranscriptAsso* entries missing their ORF length value cannot be computed, and in such cases they are considered as belonging to the same cluster.
- All the cell contexts in which the ORF has been reported on the transcript are recorded and registered in the *CellContext* table.

- All the ORF categories for the ORF on the transcript as provided by the data sources are recorded and registered in the *ProvidedCategory* table. Be aware that this is an information that is computed using the information extracted from the data sources; other ORF categories may be computed later using our own algorithm (see the **Complete the missing information and compute new information** section of the current manual).
- Finally, all the FLOSS classes reported by the data sources are recorded in the *FlossClass* table of the PRO database.

Clean the PRO database Finally the database is clean by removing:

- all the *ORF* entries for which no *ORFTranscriptAsso* may have been computed,
- all the *Transcript* entries for which no *ORFTranscriptAsso* may have been computed.

Be aware that none of the gene (or their symbol / alias / IDs) will be removed; as it may be useful to know that an ID / alias / symbol is known by the database and there are just no ORF on it. Indeed this case is different from the case where a gene harbor ORFs but one of its alias or ID is missing from the cross-references.

Merge command line

To run the Merge strategy, use:

```
sORFdatafreezer Merge -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `DS_DATABASE_NAME` item.
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

The following section and items of the config file may be provided:

- `[MERGE_PARAMETERS]` section:
  - `GENOMIC_LENGTH_DIFF_THRESHOLD` item: Threshold for absolute difference in genomic length. The genomic length (defined as the cumulative sum of an ORF exons) is computed for each entry of the ORF table prior and after the lift over (*i.e.* the conversion of genomic coordinates from an annotation version to the current one) and their absolute difference is calculated. This item allow to exclude all the entries that have a difference larger or equal to the provided threshold. Setting the threshold to `-1` allow to ignore this option.
  - `SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD` item: Threshold to use to include a nucleotide or an amino acid in a sequence consensus.
  - `MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS` item: Maximum difference between the lengths of two ORF to belong to the same cluster when merging the DSORFTranscriptAsso entries (in amino acids). When setting this parameter to 3, two DSORFTranscriptAsso will be considered as being part of two different "clusters" if the difference of their length equals or exceeds 4.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing PRO database at the provided path / on the server prior to build a new one. The DS database will not be affected.

- `-v`, `--verbosity`: Set the level of verbosity.
- `-t`, `--threads`: Number of threads available. If not provided the program try to use all the threads available on the computer.
- `-d`, `--checkDSOTA`: Should the content of the DSORFTranscriptAsso table need to be check prior to run the strategy? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option.
- `-s`, `--computeConsensus`: Should a consensus of the DSORFTranscriptAsso sequences be computed? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option.

Resume a merging that failed

Related to its high complexity, the **Merge** strategy is a highly resources-consuming and time-consuming strategy. For some reasons independent from the source code, the merging may failed (*e.g.* the computer run out of memory, connection with the MySQL server has been lost…). Hopefully, this is possible to resume the merging from where it failed using the **ResumeMerge** strategy.

Information regarding the ResumeMerge strategy

To use this strategy, you need first to identify the step that failed (consulting the main log file) as it will be necessary to provide this information to the program.

Here are some hints to **identify the step that failed**:

- Using the log file, if the last log at the **INFO** level is:
  - `Copying the entries of the gene-related and metadata-related tables into the PRO database.`, then the merging failed during the copy of conserved information (step 1).
  - one of the following: `Starting to merge the entries of the DSORF table.`, `Starting to regroup the perfectly identical entries of the DSORF table (DS database) into new ORF entries (PRO database).`, `Starting to regroup the identical entries of the DSORF table (DS database) into new ORF entries (PRO database).`, `Starting to regroup the similar entries of the DSORF table (DS database) into existing ORF entries (PRO database).`, then the merging failed during the merging of the ORFs (step 2).
  - one of the following: `Starting to merge the entries of the DSTranscript table.`, `Starting to regroup the entries of the DSTranscript table (DS database) with the same official ID into new Transcript entries (PRO database).`, then the merging failed during the merging of the transcripts (step 3).
  - `Starting to merge the entries of the DSORFTranscriptAsso table.` or a log regarding the save of (ORF, Transcript) associations in csv files, then the merging failed during the resolution of the associations between the ORFs and the transcripts (or whilst saving these information in files in the output folder) (step 4).
  - `Starting to merge the DSORFTranscriptAsso entries for all existing ( ORF ID, Transcript ID ) couples.`, then the merging failed during the merging of ORF-Transcript associations (step 5).
  - `Starting to clean the PRO database.`, then the merging failed trying to clean the database (step 6)
- Consulting the database (with adminer, phpMyAdmin or SQLite browser for instance):
  - If the *PROSpeciesCatalog* or the *PROGene* or the *PROGeneAlias* table is empty, then the merging failed during the copy of conserved information (step 1).
  - If the *ORF* or the *Transcript* table is empty, then the merging failed during the merging of the ORFs (step 2) or during the merging of the transcripts (step 3). **Caution**: existing entries in the *ORF* table does not necessarily mean that the ORF merging was completed successfully!
  - If the *Transcript* or the *ORFTranscriptAsso* is empty, then the merging failed during the merging of the transcripts (step 3) or during the merging of the ORF-Transcript associations (step 4 or 5). **Caution**: existing entries in the *Transcript* table does not necessarily mean that the transcript merging was completed successfully!

- If the *ORFTranscriptAsso* is empty, then the merging failed during the resolution of the associations between the ORFs and the transcripts (or whilst saving these information in files in the output folder) (step 4) or during the merging of ORF-Transcript associations (step 5). **Caution**: existing entries in the *ORFTranscriptAsso* table does not necessarily mean that the ORF-transcript associations merging was completed successfully!
- If entries of the *ORF* or of the *Transcript* have not any child in the *ORFTranscriptAsso* table, then the merging failed during the resolution of the associations between the ORFs and the transcripts (or whilst saving these information in files in the output folder) (step 4) or during the merging of ORF-Transcript associations (step 5) or trying to clean the database (step 6).

Always prefer to use the log to identify this information, except if you set the verbosity at an inappropriate level to clearly identify the last step that run successfully (*e.g.* log set at `critical` level).

When this step is identified, the **ResumeMerge** strategy may be started using the `--resumeAtStep` option. This option takes one of the following values:

- `after_conserved`: to restart **after** the copy of conserved information. Use this when the last step successfully performed was the step 1.
- `after_orf`: to restart **after** the merging of the ORF entries. Use this when the last step successfully performed was the step 2.
- `after_transcript`: to restart **after** the merging of the Transcript entries. Use this when the last step successfully performed was the step 3.
- `after_ota_id_asso`: to restart **after** the resolution of the associations between the ORFs and the transcripts. Use this when the last step successfully performed was the step 4.
- `during_ota`: to restart **during** the merging of the *DSORFTranscriptAsso* entries. `after_ota_id_asso` will restart this step from the principle whilst `during_ota` will first identify the merging that have been performed successfully and restart this step from where it failed. Use this when the last step successfully performed was the step 4.
- `after_ota`: to restart **after** the merging of the *DSORFTranscriptAsso* entries. Use this when the last step successfully performed was the step 5.

Be aware that these strategy can only work successfully if you did **not** alter neither any of the hidden (`.dcorf`) files nor the DS nor the PRO database! For more information about any of these steps, read the documentation of the **Merge** strategy.

Troubleshooting

If the program run out of memory, it will be halted and raise errors such as `OSError: [Errno 12] Cannot allocate memory`. When it happen, this issue is usually met during the computation of the *ORFTranscriptAsso* entries (step 5) and may be fixed by reducing the amount of entries managed at the same time. In fact, the program is taking advantage of multi-processing for this step, which could require high amounts of memory. You can try to fix the error by reducing the amount of entries handled at the same time by lowering the value of the `MAX_POOL_SIZE` variable in the `fr.tagc.uorf.core.utils.Constants` module. This should slightly reduce the efficiency of the computation but may allow to fix the issue.

ResumeMerge command line

To run the ResumeMerge strategy, use:

```
sORFdatafreezer ResumeMerge -c ConfigFilePath -a stepName [OPTIONS]
```

Options and config sections / items that may be provided to the **Merge** strategy may be provided to the **ResumeMerge** too (expected for the `-f` option).

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.
- `-a`, `--resumeAtStep`: The name of the step at which the merging should be resumed.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `DS_DATABASE_NAME` item.
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

The following section and items of the config file may be provided:

- `[MERGE_PARAMETERS]` section:
  - `GENOMIC_LENGTH_DIFF_THRESHOLD` item: Threshold for absolute difference in genomic length. The genomic length (defined as the cumulative sum of an ORF exons) is computed for each entry of the ORF table prior and after the lift over (*i.e.* the conversion of genomic coordinates from an annotation version to the current one) and their absolute difference is calculated. This item allow to exclude all the entries that have a difference larger or equal to the provided threshold (`1` by default). Setting the threshold to `-1` allow to ignore this option.
  - `SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD` item: Threshold to use to include a nucleotide or an amino acid in a sequence consensus.
  - `MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS` item: Maximum difference between the lengths of two ORF to belong to the same cluster when merging the DSORFTranscriptAsso entries (in amino acids). When setting this parameter to 3, two DSORFTranscriptAsso will be considered as being part of two different "clusters" if the difference of their length equals or exceeds 4.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-t`, `--threads`: Number of threads available. If not provided the program try to use all the threads available on the computer.
- `-d`, `--checkDSOTA`: Should the content of the DSORFTranscriptAsso table need to be check prior to run the strategy? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option.
- `-s`, `--computeConsensus`: Should a consensus of the DSORFTranscriptAsso sequences be computed? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option.

Complete the missing information, normalize data and compute new information

Download missing information and normalize cell contexts

Once the PRO database has been created with the **Merge** strategy, there are still some information that could be normalized as well as some data that are missing and could be recovered using external databases (Ensembl). The **ComputeMissingInfo** strategy aims to perform these operations.

Caution: This strategy needs the **Merge** strategy to have been run successfully to be used.

Information regarding the ComputeMissingInfo strategy

First the **computeMissingInfo** strategy will compare for each *ORFTranscriptAsso* entry its genomic length (`length_nt_min`, `length_nt_max`) with the genomic length registered in the related *ORF* entry (`orf_genomic_length` attribute, see the documentation about the **LiftOver** strategy for more information). If the values of the `length_nt_min` and `length_nt_max` attributes are different, then the `gen_len_eq_orf_len` attribute of the *ORF-TranscriptAsso* entry is set to False. Otherwise, their value is compared to the genomic length of the *ORF* entry and if they are equal, the `gen_len_eq_orf_len` attribute takes the `True` value.

Then, the names of the cell contexts are normalized. At this stage, the cell contexts are using the name provided by the data sources. Hence an actual same context can be registered under several names in the database (*e.g.* both `Primary human fibroblast (HFF)`, `HFF` and `rutkowski_2015` refers to primary human fibroblasts). To do so, the program uses a dictionary that associates to each new name to use for a cellular context the list of names used by the data sources. A custom dictionary may be provided in the config file with the `CELL_CONTEXTS_DICTIONARY` item, using a Python-like syntax. Be aware that the old cell type names will be erased, so this is no longer possible to give different names to cell contexts that have already been merged together by previously running the current strategy.

When the `--downloadMissingInfo` option has been selected, the strategy will try to download missing information. This step uses both the `PyEnsembl` package and the Ensembl REST API to download the information. The following information are recorded:

- For the *ORF* entries:
  - The nucleic sequences are downloaded (using the Ensembl REST API). For the ORFs which are spliced, only the sequences of the exons are downloaded. The stop codon is included in the sequence. This information is registered under the `sequence` attribute.
  - The amino acid sequences are computed by *in silico* translation of the DNA sequence previously downloaded (with the `Bio.Seq` module of the `BioPython` package). This information is registered under the `sequence_aa` attribute.
- For the *Transcript* entries:
  - When the transcript has an official ID (not starting with `UNKNOWN`), the strategy try to get its name, strand, absolute start and end genomic coordinates, absolute CDS start and stop genomic coordinates (for protein coding transcripts), its RNA biotype and its full sequence (including 3' and 5' UTRs) (using the `PyEnsembl` package).
  - If there are entries missing their sequence, then it try to download it with the Ensembl REST API using the start and stop genomics coordinates if they are provided.

Finally, a list of all RNA biotypes existing in the database is computed and recorded in the `UTRNABiotypeCatalog` table.

NB: - For the steps that use the Ensembl REST API, if an HTTP request fails due to an error that is not related to the data itself (*e.g.* server unreachable, connection lost…), then the attempt to download the sequence is repeated until success.

computeMissingInfo command line

To run the computeMissingInfo strategy, use:

```
sORFdatafreezer computeMissingInfo -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

The following section and item of the config file may be provided:

- `[COMPUTE_MISSING_INFO_PARAMETERS]` section:
  - `CELL_CONTEXTS_DICTIONARY` item: The dictionary that inform how the cell contexts have to be renamed and / or merged. This dictionary must be provided using a Python-like syntax.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Enforce the computation of all steps, including the ones that already succeed. When this option is not selected, the strategy will resume from where it failed.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-d`, `--downloadMissingInfo`: Download the missing information (such as ORF and Transcript sequences) from external databases. Note that selecting this option may be highly time-consuming.

Resume a ComputeMissingInfo that failed

To resume a **ComputeMissingInfo** strategy that failed, just restart the strategy using the same command line as previously used. Make sure to do **not** select the `-f` option, as this would restart the strategy from the beginning!

Troubleshooting

When the `--downloadMissingInfo` option has been selected, the **ComputeMissingInfo** may be susceptible to log a lot of error messages reporting bad request status to Ensembl databases. This may happen when:

- Your connection to the Internet is unstable.
- Other processes or users are using the Ensembl REST API at the same time on your network. This is susceptible to happen if you try to start the ComputeMissingInfo strategy at the same time on different databases.
- Ensembl servers are under maintenance or encoutering issues.

Computing the relative coordinates

So far, all coordinates registered in the database are absolute genomic coordinates. The **ComputeRelCoord** strategy allows to compute the relative coordinates for the entries related to a **transcript having an official ID**.

Information regarding the ComputeRelCoord strategy

This strategy uses the R `annotation_hub` and `ensembldb` libraries to perform this operation. The computation of relative coordinates takes obviously into account the phenomena of (alternative) splicing when occurring.

The following coordinates are converted:

- Transcript CDS start and stop coordinates: For transcripts with an official ID and that harbor a CDS (coding transcripts), the relative start and stop of the CDS are computed and registered under the `rel_cds_start_pos` and `rel_cds_stop_pos` attributes of the *Transcript* table.
- ORF start and stop coordinates: For the ORFs that are located on one (or several) identified transcript(s) and for which the official ID(s) is known, the relative start and stop coordinates are computed **for each transcript** (as the position is relative to the start of the transcript) are thus registered under the `rel_start_pos` and `rel_stop_pos` attributes of the *ORFTranscriptAsso* table.

ComputeRelCoord command line

To run the ComputeRelCoord strategy, use:

```
sORFdatafreezer ComputeRelCoord -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.
- `-N`, `--databaseName`: The database name.
- `-M`, `--databaseModel`: The schema of the database (PRO / DS).

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Compute again any existing relative coordinates.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-t`, `--threads`: Number of threads available. If not provided the program try to use all the threads available on the computer.

Resume a ComputeRelCoord that failed

To resume a **ComputeRelCoord** strategy that failed, just restart the strategy using the same command line as previously used. Make sure to do **not** select the `-f` option, as this would restart the strategy from the beginning!

Troubleshooting

If the program run out of memory, it will be halted and raise errors such as `OSError: [Errno 12] Cannot allocate memory`. This issue may be fixed by reducing the amount of coordinates converted at the same time. In fact, the program is taking advantage of multi-processing for this step, which could require high amounts of memory. You can try to fix the error by reducing the amount of subprocess handled in the same pool by lowering the value of the `MAX_POOL_SIZE` variable in the `fr.tagc.uorf.core.utils.Constants` module and / or by reducing the number of entries handled at the same time by a process by lowering the value of the `MAX_ENTRIES_PER_DATAFRAME` variable in the `fr.tagc.uorf.core.utils.Constants` module. This may reduce the efficiency of the computation but could help fixing the issue.

Get the start flanking sequence and information regarding the Kozak context

As described earlier, the `kozak_context` attribute of the *ORFTranscriptAsso* table records information regarding the Kozak context (as a boolean) from the data sources. Nevertheless most of the data sources are missing this information and the criteria used to define a Kozak context are not necessarily the same.

Hence, the **ComputeKozakContext** strategy allows to get the sequences flanking the start codon of the ORFs and to provide new levels of information about the Kozak context.

Caution: This strategy needs the **ComputeRelCoord** strategy to have been run successfully to be used.

Information regarding the ComputeKozakContext strategy

First, for each ORF that is related to one (or several) transcript(s) which has(ve) an official ID, and for which the computation of the relative start coordinates has succeed, the sequence flanking the start codon is computed. This sequence includes the -6 to +4 nucleotides, the first nucleotide of the start codon being at the +1 position.

e.g. An **ATG** start codon could have the flanking sequence: GCCACC**ATG**G.

This strategy is able to define four kinds of Kozak contexts, based on Hernandez et al., Trends in Biochemical Sciences, 2019 (doi: 10.1016/j.tibs.2019.07.001). The following regular expression are used to find out the type of Kozak context:

- Optimal: `GCC[AG]CC.{3}G`
- Strong: `.{3}[AG].{2}.{3}G`
- Moderate: `(.{3}[AG].{2}.{3}[ATC]|.{3}[CT].{2}.{3}G)`
- Weak: `.{3}[CT].{2}.{3}[ACT]`

ComputeKozakContext command line

To run the ComputeKozakContext strategy, use:

```
sORFdatafreezer ComputeKozakContext -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Should all Kozak contexts be computed again?
- `-v`, `--verbosity`: Set the level of verbosity.

Resume a ComputeKozakContext that failed

To resume a **ComputeKozakContext** strategy that failed, just restart the strategy using the same command line as previously used. Make sure to do **not** select the `-f` option, as this would restart the strategy from the beginning!

Normalize the ORF categories, compute new ORF annotations

As in the section dedicated to the **Merge** strategy, the ORF categories provided by the data sources are registered in the *ProvidedCategory* table. Nevertheless, like for the cellular contexts, several names could have been used to define a particular category (*e.g.* both `5'UTR` and `upstream` may be describing upstream ORFs).

The **AnnotateORF** strategy allows to normalize this information by: - Normalizing the vocabulary to describe the ORFs. - Computing new ORF categories with normalized definitions

Caution: This strategy needs the **ComputeMissingInfo** strategy to have been run successfully to be used.

Information regarding the AnnotateORF strategy

When the `--computeCatFromSource` option is selected, the names of the provided categories are normalized. The method is similar to the one used for the cellular contexts in the **ComputeMissingInfo** strategy. To do so, the program uses a dictionary that associates to each new name to use for a category the list of names used by the data sources. A custom dictionary may be provided in the config file with the `CATEGORY_ASSOCIATION_DICTIONARY` item, using a Python-like syntax. Contrary to the cell types, the old category names will be kept in the database. Hence if you update the dictionary and restart the strategy it will still be possible to redefine the categories as if it has not yet been performed. These information are stored in the *ORFCategory* table.

When the `--computeAnnot` option is selected the ORFs categories are computed from scratch (*i.e.* provided categories are not used for this computation) and the information are registered in the *ORFAnnotation* table.

This annotation is performed in sevel successive steps based on these 5 criteria:

- The **length** of the ORFs. ORFs which size in amino acids is lower of equal to the size defined in the config file using the `SHORT_ORF_ANNOTATION_SIZE_THRESHOLD` item will be annotated as short (`sORF`).
- The **strand** of the ORF compared to the one of the transcript. When the ORF is located on the opposite strand than its transcripts, it is annotated `Opposite`.
- The **reading frame** of the ORF. When the ORF reading frame is not the same than the one of the CDS, it is annotated `Alternative`.
- The **biotype** of the transcript. The following ORF annotations are defined according to the type of their transcript:
  - ORFs located on `Non sense mediated decay`, `NMD`, `nonsense_mediated_decay` or `non_stop_decay` biotypes are annotated `NMD`.
  - ORFs located on `Pseudogene`, `IG pseudogene`, `Polymorphic pseudogene`, `Processed pseudogene`, `processed_pseudogene`, `Transcribed pseudogene`, `transcribed_processed_pseudogene`, `transcribed_unproces unprocessed_pseudogene`, `transcribed_unitary_pseudogene`, `Translated pseudogene`, `Unitary pseudogene`, `unitary_pseudogene` or `Unprocessed pseudogene` biotypes are annotated `Pseudogene`.
  - ORFs located on `3' overlapping ncRNA` biotypes are annotated `Downstream`.
  - ORFs located on `Readthrough` or `Stop codon readthrough` biotypes are annotated `Readthrough`.
  - ORFs located on `Long intergenic ncRNA` or `lincRNA` biotypes are annotated `Intergenic`.
  - ORFs located on `Non coding`, `ncRNA`, `Processed transcript`, `processed_transcript`, `Long non-coding RNA`, `lncRNA`, `3' overlapping ncRNA`, `Macro lncRNA`, `Long intergenic ncRNA`, `lincRNA`, `miRNA`, `miscRNA`, `piRNA`, `rRNA`, `siRNA`, `snRNA`, `snoRNA`, `tRNA` or `vaultRNA` biotypes are annotated `ncRNA`.
  - ORFs located on `retained_intron`, `sense_intronic`, `sense_overlapping` biotypes are annotated `Intronic`.
  - ORFs located on `Antisense`, `antisense` biotypes are annotated `Overlapping`.
- The **relative position** of the ORF compared with the CDS (main coding sequence) of the transcript looking at the absolute genomic coordinates. The following ORF annotations are defined according to this relative position:
  - If the ORF start codon is located upstream of the CDS start codon, and the ORF stop codon is located upstream of the CDS stop codon, then the ORFs is annotated `Upstream`.
  - If the ORF start codon is located downstream of the CDS start codon, and stop codon is located downstream of the CDS stop codon, then the ORF is annotated `Downstream`.
  - If the ORF start codon is located upstream of the CDS start codon, and the ORF stop codon is located upstream of the CDS stop codon and within the CDS, then the ORF is annotated `Overlapping`.
  - If the ORF start codon is located within the CDS, and stop codon is located downstream of the CDS stop codon, then the ORF is annotated `Overlapping`.
  - If the ORF start codon is located upstream of the CDS start codon and the ORF stop codon is located at the CDS stop codon, then the ORF is the annotated `InCDS`.
  - If the ORF start codon is located at the CDS start codon and the ORF stop codon is located at the CDS stop codon, then the ORF is the annotated `CDS`.

- If the ORF start codon is located upstream of the CDS start codon, and the ORF stop codon is located downstream of the CDS stop codon, then the ORF is annotated `NewCDS`.

AnnotateORF command line

To run the AnnotateORF strategy, use:

```
sORFdatafreezer AnnotateORF -c ConfigFilePath -a -s [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and item of the config file may be provided:

- `[ANNOTATE_ORF_PARAMETERS]` section:
  - `CATEGORY_ASSOCIATION_DICTIONARY` item: The dictionary that inform how the provided categories have to be renamed and / or merged. This dictionary must be provided using a Python-like syntax.
  - `SHORT_ORF_ANNOTATION_SIZE_THRESHOLD` item: The maximal size for an ORF to be annotated as short (in amino acids, *e.g.* if set to `100`, then all ORF with a length shortest or equal to 100 amino acids will be annotated as short).

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `PRO_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_HOST_IP`: The IP of the database host.
  - `DATABASE_PORT`: The port to use to establish the connection to the database.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete all the entries of the ORFCategory and ORFCategoryCatalog and/or of the ORFAnnotation and ORFAnnotationCatalog tables (PRO database, depending on the other options selected) prior to run the strategy.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-s`, `--computeCatFromSource`: Compute the ORF categories from the categories provided by the datasource (ORFCatagory table).
- `-a`, `--computeAnnot`: Annotate ORFs using our own algorithm based on length, biotype, strand and relative position (ORFAnnotation table).

Resume a AnnotateORF that failed

To resume a **AnnotateORF** strategy that failed, just restart the strategy using the same command line as previously used. Make sure to do **not** select the `-f` option, as this would restart the strategy from the beginning!

Filter in the database content and create a new database

Once the PRO database has been created and completed with previously described strategies, it may sometimes be useful to filter the content of the database using one or several filters. This can obviously be achieved by querying directly the database. Nevertheless, it may be of interest for some users to be able to extract a "subset" of the full

data contained in the database to restrict its content to the data related to a list gene or an ORF annotation for instance. The **Filter** strategy aims to help doing this.

Caution: This strategy may need the the **ComputeMissingInfo** and/or the **AnnotateORF** strategies to have been run successfully to be used (depending on the filters required).

Information regarding the Filter strategy

The data can be filtered using either:

- A list of genes (`GENE_LIST_FILTER` item of the config file): A path to a csv file has to be provided. The list could be either located on a line or a column (this will be detected automatically), but the file must not have several columns and several lines at the same time.
- A list of cellular contexts (`CELL_CONTEXT_FILTER` item of the config file): A comma-separated list.
- A list of ORF categories (`ORF_CATEGORY_FILTER` item of the config file): A comma-separated list.
- A list of ORF annotations (`ORF_ANNOTATION_FILTER` item of the config file): A comma-separated list.

Caution: The strategy currently allows only **one single type** of filter, so this is not possible to provide a list of genes and ORF annotations at the same time for instance. If you need to do so, we advice to first use the **Filter** strategy with one of the filter in order to create an intermediate database and then to use again the **Filter** strategy on this intermediate database with the second filter. The order in which the filters are provided does not matter.

You need then to provide a type of filtering in the config file with the `FILTERING_TYPE` item. You may provide either `intersection` or `union` values.

- `intersection`: Extract the ORFs for which all the criteria of the list are respected. For instance if the filtering is performed on ORF annotations providing the list `sORF, upstream`, then only the ORFs that have **both** of these annotation will be get. This type of filtering is not allowed for gene lists as an ORF cannot belong to two different genes according to the database model we use.
- `union`: Extract the ORFs for which at least one of the criteria of the list are respected. For instance if the filtering is performed on ORF annotations providing the list `sORF, upstream`, then all the ORFs that have **at least** one of these annotation will be get.

The strategy consist of successive queries to extract all the data that satisfy the criteria provided by the user.

NB: - Comma-separated list of values may eventually contain one space after the comma. - For more information about the differences between the ORF categories and the ORF annotations, please see the *Normalize the ORF categories, compute new ORF annotations* section of the current manual.

Filter command line

To run the Filter strategy, use:

```
sORFdatafreezer Filter -c ConfigFilePath [OPTIONS]
```

The following options are mandatory:

- `-c`, `--configfile`: Absolute path to the config file.

The following section and items of the config file are mandatory:

- `[DATABASE]` section:
  - `PRO_DATABASE_NAME` item.
  - `FILT_DATABASE_NAME` item.
  - `DATABASE_SPECIES` item.
- `[FILTERING]` section:
  - `FILTERING_TYPE` item.
  - One of the following item has to be provided:
    - `GENE_LIST_FILTER`: The absolute path to a csv file containing a list of genes.
    - `CELL_CONTEXT_FILTER`: A comma-separated list of cell contexts.
    - `ORF_CATEGORY_FILTER`: A comma-separated list of ORF categories.
    - `ORF_ANNOTATION_FILTER`: A comma-separated list of ORF annotations.

When using MySQL databases, the following section and items of the config file may be used:

- `[DATABASE]` section:

  - `DATABASE_HOST_IP`: The IP of the databases host.
  - `DATABASE_PORT`: The port to use to establish the connection to the databases.
  - `DATABASE_USER_NAME`: The username to use to connect to MySQL server.
  - `DATABASE_USER_PASSWD`: The password to use to connect to MySQL server.

When using SQLite databases, the following section and items of the config file may be used:

- `[DATABASE]` section:
  - `DATABASE_FOLDER`: The folder of the databases.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing FILT database at the provided path / on the server prior to build a new one. The PRO database from which data is get will not be affected.
- `-v`, `--verbosity`: Set the level of verbosity.

Export the database content

The sORF datafreezer comes with some strategies that allow to export the content of a PRO database at different convenient formats (Fasta, BED…). This section of the manual presents more extensively these utils.

Export the ORF sequences at fasta format

The sequences registered in the *ORF* and *ORFTranscriptAsso* tables may be exported at the fasta format using the **GenerateFastaFile** strategy. This strategy allow to export the content of one of this table, adding long or short fasta headers depending on the option selected. The fasta headers follows the UniProt recommendations and should be compatible with most of the tools that accept fasta files.

Information regarding the headers

When exporting the data from the **ORF** table, the headers look like:

- Short headers:

```
>db|UniqueIdentified|EntryName
```

- Long headers:

```
>db|UniqueIdentifier|EntryName OS=OrganismName OX=OrganismIdentifier chr=ChromosomeName strand=Strand star
```

When exporting the data from the **ORFTranscriptAsso** table, the headers look like:

- Short headers:

```
>db|UniqueIdentified|EntryName
```

- Long headers:

```
>db|UniqueIdentifier|EntryName OS=OrganismName OX=OrganismIdentifier rel_start_pos=RelativeStartPos rel_st
```

Where:

- `db` is the alias of the database ('mORF' standing for MetamORF).
- `UniqueIdentifier` is the unique identifier of the ORF or of the OTA (respectively with `ORF` or `OTA` prefix).
- `EntryName` is the unique identifier followed with the taxon code (separated by an underscore).
- `OrganismName` is the scientific name of the organism as defined by UniProt.
- `OrganismIdentifier` is the unique identifier of the organism, assigned by the NCBI.
- `DatabaseName` is the full name of the database ('MetamORF').
- `DatabaseUrl` is the URL of the user-friendly web interface of the database.
- `DatabaseVersion` is the tag of the release of the database used to generate the file.

- `ChromosomeName` is the chromosome (or scaffold) name.
- `Strand` is the ORF strand (+/-).
- `start_pos` is the absolute genomics coordinate of the start position (*i.e.* position of the first nucleotide of the start codon for the ORFs located on the + strand, position of the last nucleotide of the stop codon for the ORFs located on the - strand).
- `stop_pos` is the absolute genomics coordinate of the stop position (*i.e.* position of the last nucleotide of the stop codon for the ORFs located on the + strand, position of the first nucleotide of the start codon for the ORFs located on the - strand).
- `NumberOfExons` is the number of exons in the ORF.
- `RelativeStartPos` is the relative coordinate of the start position on the transcript (*i.e.* relative position of the first nucleotide of the start codon for the ORFs located on the + strand, relative position of the last nucleotide of the stop codon for the ORFs located on the - strand). This value is only available for the ORFs located on transcripts that have a canonical CDS.
- `RelativeStopPos` is the relative coordinate of the stop position on the transcript (*i.e.* relative position of the last nucleotide of the stop codon for the ORFs located on the + strand, relative position of the first nucleotide of the start codon for the ORFs located on the - strand). This value is only available for the ORFs located on transcripts that have a canonical CDS.
- `RelatedORFid` is the unique identifier of the ORF described in this ORF-Transcript association.
- `RelatedTranscriptID` is the unique identifier of the transcript described in this ORF-Transcript association.

All coordinates are 1-based, *i.e.* the first position of the chromosome is 1.

Examples:

```
>mORF|ORF13|ORF13_HUMAN
>mORF|ORF246|ORF246_HUMAN OS=Homo sapiens OX=9606 chr=6 strand=+ start_pos=32972748 stop_pos=32972927 exon

>mORF|OTA6|OTA6_HUMAN
>mORF|OTA7|OTA7_HUMAN OS=Homo sapiens OX=9606 rel_start_pos=312 rel_stop_pos=344 orf_id=4199 transcript_id
```

GenerateFastaFile command line

To run the GenerateFastaFile strategy, use:

```
sORFdatafreezer GenerateFastaFile [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-s`, `--seqType`: Type of the sequence required (`DNA` or `PROT`).
- `-q`, `--queryTable`: Table to query to generate the FASTA file (`ORF` for *ORF* table, `OTA` for *ORFTranscriptAsso* table).
- `-e`, `--excludeSqcesWithStop`: If selected, all the sequences that contains stop codons (at any other place that their end) will be excluded of the fasta file.

- `-l`, `--longHeader`: Use this option to get long fasta headers.
- `-o`, `--outputFolder`: The absolute path to the folder in which the GFF file has to be saved.
- `-a`, `--fastaFilename`: The name for the FASTA file generated (without its extension).

Export the ORF information at BED format

The content of the *ORF* table may be exported at BED format using the **GenerateBEDFile** strategy. The BED file generated is a 12 columns files compatible with all tools accepting this format. In particular, it may be used with UCSC and Ensembl genome browsers as well as IGV software.

The `--bigBed` option of the strategy may be used to generate an additional file at the bigBed format.

Information regarding the BED format

For extensive information regarding the Bed format, you may consult:

- http://genome-euro.ucsc.edu/FAQ/FAQformat.html#format3
- https://genome.ucsc.edu/goldenPath/help/customTrack.html#TRACK

The BED files created by the sORF datafreezer are tab-delimited and contains 12 columns by default and 12+5 columns when the `--extendBed` option is selected. Missing values are registered as `.`.

A line of the BED file looks like:

```
chrom  chromStart  stopPosition  name  score  strand  thickStart  thickEnd  itemRgb  blockCount  blockSize
```

Where:

- `chrom` is the chromosome name (or scaffold), with the `chr` prefix.
- `chromStart` is the absolute genomics coordinate of the start position (*i.e.* position of the first nucleotide of the start codon for the ORFs located on the + strand, position of the last nucleotide of the stop codon for the ORFs located on the - strand) **in a 0-based system**. Hence, these positions correspond to the ones recorded in our database **minus one**.
- `chromEnd` is the absolute genomics coordinate of the stop position (*i.e.* position of the last nucleotide of the stop codon for the ORFs located on the + strand, position of the first nucleotide of the start codon for the ORFs located on the - strand) **in a 1-based system**. Hence, these positions correspond to the ones recorded in our database.
- `name` is the unique identifier for the ORF in the database (integer, without the `ORF` prefix).
- `score` is currently not used and always set to `0`. This may evolve in the future release of the tool.
- `strand` is the strand of the ORF.
- `thickStart` is the start position at which the feature is drawn thickly (in coordinates relative to the ORF start).
- `thickEnd` is the end position at which the feature is drawn thickly (in coordinates relative to the ORF start).
- `itemRgb` is a color in the R,G,B format. One color is currently use for each strand. This is susceptible to evolve in the future releases.
- `blockCount` is the number of exons constituting the ORF.
- `blockSizes` is a comma-separated list of exon sizes.
- `blockStarts` is a comma-separated list of the exon starts, relatively to the position of the ORF start codon (`chromStart`).

BED files are using "0-start, half-open" systems (also inaccurately called "0-based start, 1-based end" systems sometimes; which may be easier to understand). You may find more information about counting systems for genomics coordinates on the UCSC blog.

Example:

```
chr1  8361096  8361132  4093  0  -  8361096  8361132  0,0,255  1  36  0
```

Adding a track line to the file

For extensive information regarding the Bed format, you may consult:

- References provided in the previous section
- https://software.broadinstitute.org/software/igv/TrackLine

By default, the BED file is generated without any track line, header or comment. It is possible to add a track line before the first line by using the `--trackLine` option. The track line may help the user to visualize its data on a genome browser as it will provide the visualization software the best settings (as space separated "key-value" information, this includes using the right genome assembly). Be aware that, excepting the genome browser / viewer, numerous software that accept BED files ask the track line to be removed.

The trackline looks like:

```
track name=trackName description=trackDescription htmlUrl=trackHtmlDescription url=databaseUrl db=assembly
```

Where:

- `trackName` is the name of the track.
- `trackDescription` is a short description of the track.
- `trackHtmlDescription` is the link to the HTML description page to be displayed with the track.
- `databaseUrl` is the link to the user-friendly web interface of the database.
- `assembly` is the genome assembly for which the data is intended, using the UCSC assembly name (*e.g* `hg38`).
- `visibility` defines the initial display mode of the annotation track.
- `defaultColor` is the main color for the annotation track.
- `itemRgb` inform the genome browser if it must use the RGB value to color items.

NB: - If you are trying to convert the BED file at BigBed format by yourself, using UCSC bedToBigBed tool will failed if you provide a BED that contains a track line needs to be removed. Nevertheless, this option is **not** incompatible with the `--bigBed` option of the strategy. Hence if you select both the `--trackLine` and the `--bigBed` options of this strategy, you will get both the BED file with the track line and the BigBed file.

Information regarding the additional columns

Five additional columns providing extensive information may be added to the BED file using the `--extendBed` option.

In such cases, a line of the BED file looks like:

```
chrom  chromStart  stopPosition  name  score  strand  thickStart  thickEnd  itemRgb  blockCount  blockSize
```

Where:

- First 12 elements are the same as the one previously described
- `transcriptIDs` is a comma-separated list of transcript IDs ("official" IDs, such as Ensembl transcript IDs) that harbor the ORF. The `UNKNOWN_TRANSCRIPT` ID indicates that at least one of the original data sources did not provide any information regarding the transcript harboring this ORF.
- `rnaBiotypes` is a comma-separated list of the RNA biotypes harboring this ORF.
- `orfAnnotations` is a comma-separated list of the annotations related to this ORF. Please read the **Normalize the ORF categories, compute new ORF annotations** section of the current manual for more information. This list includes information from the *ORFAnnotation* table **exclusively**.
- `KozakContext` is a comma-separated list of the Kozak context computed for this ORF. The context corresponds to the one computed using our own algorithm. Please read the **Get the start flanking sequence and information regarding the Kozak context** of the current manual for more information.

Example:

```
chr1  8361096  8361132  4093  0  -  8361096  8361132  0,0,255  1  36  0  ENST00000377464  protein_coding E
```

Caution: - Be aware that there are **no** correspondences between list-element of the five last columns. For instance, the first transcript ID of the list does **not** necessarily correspond to the first annotation of the list. This is explained by the fact that the transcript IDs, biotypes, annotations and computed Kozak contexts are provided as **non-redudant** lists. Moreover an ORF may have several distinct annotations when located on the same transcript (*e.g.* it can be *Upstream* and *Overlapping* at the same time for instance) and there may be missing data in the database (*e.g.* biotype of a transcript unknown), making impossible any correspondence between the other columns and this column.

NB: - Using this option will generate a new file with `.as` extension in the output folder. This file is an auto-sql file generated by our program and necessary to perform conversion at the BigBed format. It contains a description of the 12+5 columns contained in the BED file.

Converting the BED file at bigBed format

Conversion at the BigBed format can be easily performed using the UCSC bedToBigBed tool. Nevertheless, the conversion could sometimes be tricky; in particular the BED file needs to do not contain track line, an auto-sql file needs to be provided when using files at non-conventional formats (such as 12+5 format). In order to make the conversion easier, we implement the `--bigBed` option.

Using this option will first prepare a temporary file of your BED file, without track line, as well as the appropriate auto-sql (`.as`) file necessary to perform the conversion.

Using this option will generate tow new files in the output folder:

- The bigBed file (with `.bb` extension).
- A file with the `.chrom.sizes` extension and containing all the chromosome sizes. As, the chromosome sizes are required to perform the conversion, this file is download from the UCSC using the `fetchChromSizes` executable.

NB: - This option is compatible with the `--trackLine` option. - This option is compatible with the `--extendBed` option. - This option is not incompatible with the `--includeNonConventionalChr` option but the conversion at BigBed format is susceptible to fail when both of them are used. Hence, if there are chromosomes registered in the database that do not exists in this file, the conversion will fail. - The `.as` and `.chrom.sizes` file may eventually be removed manually, as they were only necessary for the conversion. - Using this option will **not** delete the file generate at the BED format.

Additional information regarding the GenerateBEDFile strategy

The **GenerateBEDFile** uses the **GenerateBEDContent** strategy to first generate one line respecting the BED format (12+5 columns) for **all** the ORF entries and save them in the PRO database (in the *UTBEDContent* table). In a second time, it query the database and assembles the lines to generate output file.

Doing this instead of directly create the BED file allows to:

- Lower the computation time required to create the BED file, as the lines will be computed only once (the first time the **GenerateBEDFile** strategy is used), even if several BED files are generated (with and without track line for instance),
- Easily create new BED files restricted to a subset of the database, with any interface (web interface, R API…). Indeed, this is no longer necessary to collect the information necessary to build a line and to parse and compute some information (such as exon relative start positions), as it is just needed to query the lines corresponding to the set ORF entries desired. This allows user that would like to build new BED files restricted to some particular *ORF* entries to do it easily without needing any knowledge regarding BED formatting.

If you need to compute the content of the *UTBEDContent* without generating BED files, you can directly use the **GenerateBEDContent** strategy.

GenerateBEDFile command line

To run the GenerateBEDFile strategy, use:

```
sORFdatafreezer GenerateBEDFile [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).

- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which the BED file has to be saved.
- `-b`, `--bedFilename`: Name for the BED file generated (without ".bed" extension).
- `-a`, `--generateBEDTableContent`: Should the content of the UTBEDContent table be removed and computed again? When not selected, the BED file will be built with the existing content of the UTBEDContent. It is not necessary to use this option when running this strategy for the first time.
- `-l`, `--trackLine`: Add the track line at the beginning of the BED file.
- `-n`, `--includeNonConventionalChr`: Should the ORFs located on "non conventional" chromosomes (*e.g.* mitochondrial, scaffold) be included in the BED file?
- `-e`, `--extendBed`: Extend the BED file at 12+5 format.
- `-g`, `--bigBed`: Convert the BED file at the BigBed.

GenerateBEDContent command line

To run the GenerateBEDContent strategy, use:

```
sORFdatafreezer GenerateBEDContent [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.

Generate a trackDb file for track hub implementation

It might be interesting to implement track hubs accessible from UCSC and Ensembl genome browsers. Once the bigBed file has been successfully generated (see previous section), it is necessary to prepare couple of file allowing the UCSC and Ensembl genome browsers to access the genome track. The **GenerateTrackDbFile** strategy allows to generate the `trackDb.txt` file necessary to implement a new track hub with some advanced settings, such as filters on ORF annotations or RNA biotypes.

Please, read the following documentation for extensive information regarding the track hubs, the other files required and how to connect the UCSC and Ensembl genome browsers to a new track hub:

- https://genome.ucsc.edu/goldenPath/help/hubQuickStart.html
- http://genome.ucsc.edu/goldenPath/help/trackDb/trackDbHub.html
- http://genome-euro.ucsc.edu/cgi-bin/hgHubConnect

Information regarding the trackDb file generated

The strategy generates a `trackDb.txt` file that may be directly used to implement track hubs.

It includes lists of transcript IDs, RNA biotypes, cell types, ORF annotations and computed Kozak contexts that may be used as filter in the UCSC or Ensembl genome browsers.

Example of trackDb file:

```
track MetamORF_Hsapiens_hg38
shortLabel MetamORF_Hsapiens_hg38
longLabel MetamORF Hsapiens track hub (hg38)
```

```
html ../MetamorfHub
bigDataUrl MetamORF.bb
type bigBed 12 +
visibility full
thickDrawItem on
itemRgb on
maxItems 100000
exonArrows on
exonNumbers on
filterType.transcripts multipleListOr
filterText.transcripts *
filterLabel.transcripts Transcript IDs
filterValues.transcripts ENST00000175506,\
ENST00000216268,\
ENST00000218364,\
ENST00000222673,\
ENST00000225171,\
ENST00000225698,\
ENST00000249289
filterType.rna_biotypes multipleListOr
filterText.rna_biotypes *
filterLabel.rna_biotypes RNA biotypes
filterValues.rna_biotypes TEC,\
antisense_RNA,\
processed_transcript,\
protein_coding
filterType.cell_types multipleListOr
filterText.cell_types *
filterLabel.cell_types Cell types (cell lines, tissues...)
filterValues.cell_types Brain,\
Breast,\
Flp-In T-REx-293,\
HEK293,\
HFF,\
HeLa,\
Jurkat
filterType.kozak_contexts multipleListOr
filterText.kozak_contexts *
filterLabel.kozak_contexts Computed Kozak context
filterValues.kozak_contexts moderate,\
strong,\
weak
labelFields name, transcripts, rna_biotypes, cell_types, orf_annotations, kozak_contexts
urls name="http://metamorf.univ-amu.fr/orfID/$$"\
transcripts="http://metamorf.univ-amu.fr/transcriptCentric/$$"
```

NB: - As this strategy has been implemented to facilitate the creation of MetamORF track hubs and inclusion of these track hubs in our website, some of the parameters (such as the URLs) are hard-coded and cannot be set by the user. Please contact the developer of the sORF datafreezer if you intend to change these parameters (see **Contact** section at the end of the current manual).

GenerateTrackDbFile command line

To run the GenerateTrackDbFile strategy, use:

```
sORFdatafreezer GenerateTrackDbFile [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which the trackDb file has to be saved
- `-f`, `--trackFilename`: Name for the trackDb file generated (without ".txt" extension).
- `-g`, `--bigBed`: Create the bigBed file corresponding to the trackDb file at the same time. See the **Export the ORF information at BED format** section of the current manual for more information. If selected, both the BED, bigBed, `.as` and `.chrom.sizes` files will be generated in the same folder than the `trackDb.txt` file. The output generated by the use of this option are the same than the one generated using the **GenerateBEDFile** strategy **with** `--extendBed` and `--bigBed` options and **without** `--includeNonConventionalChr` options.

Export the ORF information at GFF format [BETA VERSION]

In addition to the BED format, the content of the *ORF* table may be exported at the GFF3 format using the **GenerateGFFFile** strategy. As BED format, the GFF3 format is accepted by many tools, in particular genome browsers and sequence alignment tools. GFF3 files propose the information in an highly structure way, using sequence ontology and relationships between the features.

Caution: Be aware that the **GenerateGFFFile** strategy is currently efficiently working and will generate a "GFF3-like formated" file, **respecting all the GFF3 requirement excepted the sequence ontology**. More information regarding this issue are provided below. Hence, if you need a file at GFF3 format but the sequence ontology does not matter for you, then you can use this strategy. Next release of the sORF datafreezer may include an update of this strategy in order to fix this issue and propose GFF3 files that respect the sequence ontology.

Information regarding the GFF3 file

For extensive information regarding the GFF3 format, you may consult:

- https://www.ensembl.org/info/website/upload/gff3.html
- http://gmod.org/wiki/GFF3
- https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md
- http://www.sequenceontology.org/so_wiki/index.php/Category:SO:SOFA

The GFF3 files created by the sORF datafreezer are tab-delimited and contains 9 columns. Missing values are registered as `.`. The first line of the file contains `##gff-version 3`. No comments are added in the file, but may eventually be added manually after its generation using the `#` prefix. Columns are tab-delimited.

A line of the GFF3 file looks like:

```
chr  source  type  start  end  score  strand  phase  attributes  ID=52.start;Parent=52
```

Where:

- `chr` is the name of the chromosome (or scaffold), without the `chr` prefix.
- `source` is the name of the database (MetamORF).
- `type` is the type of feature. The type is necessarily one defined in the Sequence Ontology (SOFA subset).
- `start` is the absolute genomis coordinate of the start position of the feature(*i.e.* the position of the first nucleotide of the feature) **in a 1-based system**. Hence, these positions correspond to the ones recorded in our database.
- `end` is the absolute genomis coordinate of the start position of the feature(*i.e.* the position of the last nucleotide of the feature) **in a 1-based system**. Hence, these positions correspond to the ones recorded in our database.
- `score` is currently not used (always replaced with `.`). This may evolve in the future releases of the tool.
- `strand` is the strand of the feature.
- `phase` is required for all features of type 'CDS' and indicates where the feature begins with reference to the reading frame. The phase indicates the number of bases that should be removed from the beginning of the feature to reach the first base of the next codon. 0 indicates the next codon begins at the first base of the

region, 1 at the second base and 2 at the third base. `phase` is currently not used (always replaced with `.`). This may evolve in the future releases of the tool. NB: This is not too be confused with the frame. See external documentation for more information.
- `attributes` is a semi-colon separated list of tag-value pairs providing addition information about the features. It contains at list the unique entry of the feature in our database, at the format `ID=`.

The following types of features are currently used:

- The `ORF` feature is used to describe open reading frames (*i.e.* entries of the *ORF* table).
- The `start_codon` feature is used to describe the ORF start codons (this information correspond to the information registered in the `start_pos` or `stop_pos` attributes of the *ORF* entries, for the ORFs respectively located on the + and - strand).
- The `stop_codon` feature is used to describe the ORF stop codons (this information correspond to the information registered in the `stop_pos` or `start_pos` attributes of the *ORF* entries, for the ORFs respectively located on the + and - strand).
- The `exon` feature is used to describe the ORF exons (this information correspond to a part of the information registered in the `splice_starts` and `splice_ends` attributes of the *ORF* entries).

NB: GFF3 files are using "1-start, full-closed" systems (also inaccurately called "1-based start, 1-based end" systems sometimes; which may be easier to understand). You may find more information about counting systems for genomics coordinates on the UCSC blog.

In the current version, the following sequence ontology is used. Be aware that this ontology does **not** respect the official Sequence Ontology.

- A `start_codon` is located on an ORF and thus as an `ORF` as parent.
- A `stop_codon` is located on an ORF and thus as an `ORF` as parent.
- An `exon` is located on an ORF and thus as an `ORF` as parent.
- An `ORF` is orphan and has:
  - One unique `start_codon` feature as child.
  - and one unique `stop_codon` feature as child.
  - and one or several `exon` features as children (depending on the ORF splicing).

Depending on the feature type, the content of the attributes column vary:

- `ORF` features have `ID=orfID` attributes, where `orfID` is the unique ID of the ORF in our database.
- `start_codon` features have `ID=orfID.start;Parent=orfID` attributes, where `orfID` is the unique ID of the ORF in our database.
- `stop_codon` features have `ID=orfID.stop;Parent=orfID` attributes, where `orfID` is the unique ID of the ORF in our database.
- `exon` features have `ID=orfID.nb;Parent=orfID` attributes, where `orfID` is the unique ID of the ORF in our database and `nb` is the logical number of the exon on the ORF starting at 0 (*i.e.* the first exon is `0`, the second is `1` and so on, in the reading order of the ORF - not in the increasing or decreasing order of start positions).

Examples:

```
1   MetamORF  ORF          114733767   114734026   .   -   .   ID=61
1   MetamORF  start_codon  114734024   114734026   .   -   .   ID=61.start;Parent=61
1   MetamORF  stop_codon   114733767   114733769   .   -   .   ID=61.stop;Parent=61
1   MetamORF  exon         114733767   114734026   .   -   .   ID=1009.0;Parent=1009

1   MetamORF  ORF          1091150     1091543     .   -   .   ID=1009
1   MetamORF  start_codon  1091541     1091543     .   -   .   ID=1009.start;Parent=1009
1   MetamORF  stop_codon   1091150     1091152     .   -   .   ID=1009.stop;Parent=1009
1   MetamORF  exon         1091150     1091374     .   -   .   ID=1009.1;Parent=1009
1   MetamORF  exon         1091472     1091543     .   -   .   ID=1009.0;Parent=1009

1   MetamORF  ORF          25983956    25990816    .   -   .   ID=1393
1   MetamORF  start_codon  25990814    25990816    .   -   .   ID=1393.start;Parent=1393
1   MetamORF  stop_codon   25983956    25983958    .   -   .   ID=1393.stop;Parent=1393
1   MetamORF  exon         25990727    25990816    .   -   .   ID=1393.0;Parent=139
1   MetamORF  exon         25989448    25989601    .   -   .   ID=1393.1;Parent=1393
1   MetamORF  exon         25988231    25988327    .   -   .   ID=1393.2;Parent=1393
1   MetamORF  exon         25984460    25984528    .   -   .   ID=1393.3;Parent=1393
1   MetamORF  exon         25983956    25984102    .   -   .   ID=1393.4;Parent=1393
```

GenerateGFFFile command line

To run the GenerateGFFFile strategy, use:

`sORFdatafreezer GenerateGFFFile [OPTIONS]`

The following options are mandatory:

- `-N`, `--databaseName`: The database name.

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which the GFF file has to be saved.
- `-g`, `--gffFilename`: Name for the GFF file generated (without the ".gff" or ".gff3" extension).

Diagnosis tools

The sORF datafreezer provides some tools that might help to ensure of the consistency of the data registered in the database. They might help to detect any problems related to the data, error that happened during the execution of the program or bugs that need to be fixed.

If you detect any bug, please contact the developers (see the **Contact** section at the end of this manual).

Assess consistency of the database content

The **AssessDatabaseContent** strategy allows to ensure the consistency of data in a DS or PRO-like database. It performs a couple of checks on the data. These check output are registered in a 3-columns csv file where the first column contains the name of the check, the second contains the output message of the check and the third its status.

For each step, three kind of status may be registered:

- `[OK]` means that the consistency is ensured.
- `[ERROR]` means that the consistency is not respected. Most of the time this is due to a programming error. If you see this king of error, please report it to the developer.
- `[TO CHECK]` means that the data have to be checked manually. Checking the content of the second columns is generally sufficient to make sure of the data consistency.

Assessment of DS databases

The following checks are performed during the execution of the strategy:

- Check the chromosomes registered in the *DSORF* table (the list needs to be checked manually).
- Check the raw strands registered in the *DSORF* table (`+/-` allowed).
- Check the strands registered in the *DSORF* table (`+/-` allowed).
- Check all the raw start positions are lower then the raw stop positions in the *DSORF* table.
- Check all the start positions are lower then the stop positions in the *DSORF* table.
- Check that all the ORFs that are not spliced have one single exon (in the *DSORF* table).
- Check that all the ORFs that are not spliced do not have any exon coordinates registered (in the *DSORF* table).

- Check that all the ORFs that are spliced have at least two exons (in the *DSORF* table).
- Check that all the ORF raw genomic lengths are positive (in the *DSORF* table).
- Check that all the ORF genomic lengths are positive (in the *DSORF* table).
- Check that all the raw start positions of the exons are in the increasing order, all the raw stop positions are in the increasing order, and that the raw start position of exons are always higher than the raw stop position of the exon that precede it, for the ORFs located on the + strand (in the *DSORF* table).
- Check that all the raw start positions of the exons are in the drecreasing order, all the raw stop positions are in the drecreasing order, and that the raw start position of exons are always lower than the raw stop position of the exon that precede it, for the ORFs located on the - strand (in the *DSORF* table).
- Check that all the start positions of the exons are in the increasing order, all the stop positions are in the increasing order, and that the start position of exons are always higher than the stop position of the exon that precede it, for the ORFs located on the + strand (in the *DSORF* table).
- Check that all the start positions of the exons are in the drecreasing order, all the stop positions are in the drecreasing order, and that the start position of exons are always lower than the stop position of the exon that precede it, for the ORFs located on the - strand (in the *DSORF* table).
- Check that all the raw start positions are lower than the corresponding raw end positions in the *DSTranscript* table.
- Check that all the start positions are lower than the corresponding end positions in the *DSTranscript* table.
- Check that all the CDS raw start positions are lower than the corresponding CDS raw stop positions in the *DSTranscript* table.
- Check that all the CDS start positions are lower than the corresponding CDS stop positions in the *DSTranscript* table.
- Check the RNA biotypes used in the *DSTranscript* table (the list needs to be checked manually).
- Check that all the nucleic lengths are consistent with the amino acid lengths in the *DSORFTranscriptAsso* table.
- Check the chromosomes registered in the *Gene* table (the list needs to be checked manually).

Example:

```
---  Assessment of the consistency the DSORF table content
List of chromosomes:    1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 3, 4, 5, 6, 7, 8, 9, GLO
Raw strands:    +, -    [OK]
Strands:    +, -    [OK]
Raw positions:  All DSORFs have the raw start position lower than the raw stop position [OK]
Positions:  All DSORFs have the start position lower than the stop position [OK]
Splicing / Exon number: All DSORFs not spliced have a single exon   [OK]
Splicing / Exon positions:  All DSORFs not spliced do not have exonic positions [OK]
Splicing / Exon number: All DSORF spliced have at least two exons   [OK]
Raw genomic lengths:    All DSORFs have a positive raw genomic length   [OK]
Genomic lengths:    All DSORFs have a positive genomic length   [OK]
Spliced position (DSORFs on + raw strand):  All DSORFs on + raw strand have their raw spliced positions in
Spliced position (DSORFs on - raw strand):  All DSORFs on - raw strand have their raw spliced positions in
Spliced position (DSORFs on + strand):  All DSORFs on + strand have their spliced positions in the increas
Spliced position (DSORFs on - strand):  All DSORFs on - strand have their spliced positions in the decreas

--- Assessment of the consistency the DSTranscript table content
Raw positions:  All DSTranscripts have the raw start position lower than the raw end position   [OK]
Positions:  All DSTranscripts have the start position lower than the end position   [OK]
Raw CDS positions:  All DSTranscripts have the raw CDS start position lower than the raw CDS stop position
CDS positions:  All DSTranscripts have the CDS start position lower than the CDS stop position  [OK]
List of RNA biotypes:   TEC, antisense, lincRNA, non_stop_decay, nonsense_mediated_decay, processed_pseudo
--- Assessment of the consistency the DSORFTranscriptAsso table content
Lengths:    All DSORFTranscriptAsso have their nucleic lengths consistent with their lengths in amino acid

--- Assessment of the consistency the Gene table content
List of chromosomes:    1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 3, 4, 5, 6, 7, 8, 9, GLO
```

Assessment of PRO databases

The following checks are performed during the execution of the strategy:

- Check the chromosomes registered in the *ORF* table (the list needs to be checked manually).
- Check the strands registered in the *ORF* table (+/- allowed).
- Check all the start positions are lower then the stop positions in the *ORF* table.
- Check that all the ORFs that are not spliced have one single exon (in the *ORF* table).
- Check that all the ORFs that are not spliced do not have any exon coordinates registered (in the *ORF* table).
- Check that all the ORFs that are spliced have at least two exons (in the *ORF* table).
- Check that all the ORF genomic lengths are positive (in the *ORF* table).
- Check that all the start positions of the exons are in the increasing order, all the stop positions are in the increasing order, and that the start position of exons are always higher than the stop position of the exon that precede it, for the ORFs located on the + strand (in the *ORF* table).
- Check that all the start positions of the exons are in the drecreasing order, all the stop positions are in the drecreasing order, and that the start position of exons are always lower than the stop position of the exon that precede it, for the ORFs located on the - strand (in the *ORF* table).
- Check that all the start positions are lower than the corresponding end positions in the *Transcript* table.
- Check that all the CDS start positions are lower than the corresponding CDS stop positions in the *Transcript* table.
- Check that all the CDS relative start positions are lower than the corresponding CDS relative stop positions in the *Transcript* table.
- Check that all the relative ORF start positions are lower than the relative stop position (*ORFTranscriptAsso* table).
- Check the RNA biotypes used in the database (the list needs to be checked manually).
- Check the chromosomes registered in the *PROGene* table (the list needs to be checked manually).
- Check the provided ORF categories (registered in the *ProvidedCategoryCatalog* table, the list needs to be checked manually).
- Check the cell contexts (registered in the *CellContextCatalog* table, the list needs to be checked manually).
- Check the FLOSS classes (registered in the *FLOSSClassCatalog* table, the list needs to be checked manually).
- Check the ORF categories (registered in the *ORFCategoryCatalog* table, the list needs to be checked manually).

Example:

```
--- Assessment of the consistency the ORF table content
List of chromosomes:    1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 3, 4, 5, 6, 7, 8, 9, MT,
Strands:    +, -    [OK]
Positions:  All ORFs have the start position lower than the stop position   [OK]
Splicing / Exon number: All ORFs not spliced have a single exon [OK]
Splicing / Exon positions:  All ORFs not spliced do not have any exonic positions   [OK]
Splicing / Exon number: All ORF spliced have at least two exons [OK]
Genomic lengths:    All ORFs have a positive genomic length [OK]
Spliced position (ORFs on + strand):    All ORFs on + strand have their spliced positions in the increasin
Spliced position (ORFs on - strand):    All ORFs on - strand have their spliced positions in the decreasin

--- Assessment of the consistency the Transcript table content
Positions:  All Transcripts have the start position lower than the end position [OK]
CDS positions:  All Transcripts have the CDS start position lower than the CDS stop position    [OK]
List of RNA biotypes:   TEC, antisense, antisense_RNA, lincRNA, non_stop_decay, nonsense_mediated_decay, p

--- Assessment of the consistency the PROGene table content
List of chromosomes:    1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 22, 23, 3, 4, 5, 6, 7, 8, 9,
```

```
--- Assessment of the consistency the ORF categories, cell context and FLOSS classes
List of (provided) ORF categories:  3_UTR, 3UTR, 5_UTR, 5UTR, alternative_frame, annotated, Annotated, ANT
List of cell contexts:  B_cell, BJ, Blood, Brain, Brain_tumor, Breast, Flp-In T-REx-293, HAP1, HCT116, HEK
List of FLOSS classes:  Extreme, Good, No reads, Not in cutoff range    [TO CHECK]
List of (computed) ORF categories:  Alternative, CDS, Downstream, Exonic, Intergenic, Intronic, ncRNA, NMD
```

AssessDatabaseContent command line

To run the AssessDatabaseContent strategy, use:

```
sORFdatafreezer AssessDatabaseContent [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-M`, `--databaseModel`: The schema of the database (PRO / DS).

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which all the files generated have to be saved.
- `-f`, `--filename`: Name for the log file generated.

Get a summary of log files

Depending of the level of the verbosity set to run the strategies, the main log files can contain a lot of data (in particular the **Insertion** strategy, around 50,000 - 100,000 lines logged for a full built with the 6 data sources in *H. sapiens* for instance). The **GenerateStatFiles** strategy allow to generate two files summarizing the information contained in these log files.

- One file registering the number of logs at each verbosity level (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`). Any message logged at the `CRITICAL` should be checked.
- One file registering the number of logs for each existing warning and error code (cf. file registering the log codes that may be used by the sORF datafreezer).

In addition, we advice to use the `grep -i 'exc' execution.log*` command line that is expected to return no result. Otherwise, it will highlight any exception raised using the sORF datafreezer.

Caution: If the main log files have been altered, then this strategy will obviously not provide accurate information.

GenerateStatFiles command line

To run the GenerateStatFiles strategy, use:

```
sORFdatafreezer GenerateStatFiles [OPTIONS]
```

The following options may be used:

- `-f`, `--forceOverwrite`: Overwrite any existing files.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which all the files generated have to be saved.

Statistical analysis of the database content

The statistical analysis of the databases may help detecting any inconsistency in the data or issue that happened during the execution of one of the strategy. Such analysis can **not** be performed using the sORF datafreezer as it is impossible to make it fully automated and has to be performed manually. Nevertheless, a R package (**RqueryORF**) has been developed to help performing this analysis and provide convenient functions that might help. Please see the documentation of this package for more information.

Backup, restore and convert databases

SQLite databases can be backup easily by copying the unique SQLite3 file. MySQL databases can be dumped at convenient formats, such as `.sql.gz` using database management tools such as adminer or phpMyAdmin.

Nevertheless, two **Backup** and **Restore** strategies have been implemented respectively to save the content of a database in hidden files and import the content of those files to restore the database. More precisely, the content of each table is saved as serialized objects in `.dcorf` files.

Caution: If the hidden files have been altered or deleted, the **Restore** strategy will obviously failed or may result in silent errors.

Database backup

Backup command line

To run the Backup strategy, use:

```
sORFdatafreezer Backup [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-M`, `--databaseModel`: The schema of the database (PRO / DS).

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-v`, `--verbosity`: Set the level of verbosity.
- `-o`, `--outputFolder`: Absolute path to the folder in which the files have to be saved.
- `-x`, `--filePrefix`: Prefix to add to the file names where data are saved.

Restoring a database

If there is a database of this name already existing and the `--forceOverwrite` has not been selected, the sORF datafreezer will ask the user if the existing database has to be deleted. Otherwise, it will automatically create the database prior to restore its content.

Restore command line

To run the Restore strategy, use:

```
sORFdatafreezer Restore [OPTIONS]
```

The following options are mandatory:

- `-N`, `--databaseName`: The database name.
- `-M`, `--databaseModel`: The schema of the database (PRO / DS).

When using MySQL databases, the following options may be used:

- `-H`, `--databaseHost`: The IP of the database host.
- `-P`, `--databasePort`: The port to use to establish the connection to the database.
- `-u`, `--databaseUser`: The username to use to connect to MySQL server.
- `-p`, `--databasePassword`: The password to use to connect to MySQL server.

When using SQLite databases, the following options may be used:

- `-F`, `--databaseFolder`: The folder of the database.

The following options may be used:

- `-T`, `--databaseType`: Type of database (MySQL or SQLite).
- `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to restore the database from backup.
- `-v`, `--verbosity`: Set the level of verbosity.
- `-i`, `--inputFolder`: Absolute path to the folder in which the files are located.
- `-x`, `--filePrefix`: Prefix used when generated the files with the Restore strategy.

Convert SQLite databases at MySQL format and vice versa

According to the way the data are handled by the **Backup** and **Restore** strategies, this is theoretically possible to convert a SQLite database at the MySQL format and vice versa. To do this, use sequentially these strategies with different `--databaseType` options.

Caution: This method of conversion has not been extensively assessed or bench-marked and may be susceptible to result in unexpected errors. Hence, we advice to avoid as most as possible converting SQLite databases to MySQL format and vice versa.

Information to developers

A documentation dedicated to the developers (generated with Doxygen) is available at HTML format and provided with the source code. Please refer to this documentation for extensive information about the source code.

This section of the manual only presents **some** of the constants which could be interesting to change in occasional cases. Nevertheless, we **highly discourage** the change of these constant values and **do not guarantee** the successful execution of the strategies when modifying these values.

To change the default output main folder, update the `OUTPUT_FOLDER` defined in the `fr.tagc.uorf.core.util.DefaultOutputFo` module.

To change the default temporary main folder, update the `TEMPORARY_FOLDER` defined in the `fr.tagc.uorf.core.util.DefaultTe` module.

Description of some constantes defined in the `Constants` file (`fr.tagc.uorf.core.util.Constants`):

- The list of accepted species are defined in the **General constant** section. Be aware that some strategies uses tools which are not necessarily compatible with other species than *H. sapiens* and *M. musculus*. In any case, you need to provide the correspondences between species common, scientific names, genome annotations names etc. as defined by the constants of this section.
- `EMPTY_VALUES`: List of values that have to be considered as empty / missing values in the data sources.
- `ALIAS_OFF_PREFIX` allows to set the prefix to add to official symbols when filling the GeneAlias table.
- `CURRENT_ENSEMBL_RELEASE` defines the number of the current Ensembl release.
- `PATH_LOG` defines the path to use for the main log file.
- `PATH_GENEREF_LOG` defines the path to use for the gene references log file.
- `LOG_SIZE_MAX` and `GENEREF_LOG_SIZE_MAX` define the maximum size of log files.

- `LOG_MAX_FILES_NB` and `GENEREF_LOG_MAX_FILES_NB` define the maximum number of log files that can be generated.
- `MAX_POOL_SIZE` defines the maximum number of process that can be pushed in a pool (when multi-processing computations).
- `CHROMOSOME_NAME_XY` defines the chromosome name to use for the sexual chromosome.
- `MITOCHONDRIAL_CHR` defines the chromosome name to use for the mitochondrial chromosome. `MITOCHONDRIAL_CHR_LIST` is the list of chromosome names to be considered as mitochondrial chromosome.
- `PREFIX_FAKE_TRANSCRIPT` defines the prefix to use for "fake" transcripts.
- `UNKNOWN_TRANSCRIPT` defines the string to use for unknown transcripts.
- `PREFIX_UNKNOWN_GENE` defines the prefix to use for unknown genes.
- `PREFIX_OVERLAPPING_GENES` and `PREFIX_OVERLAPPING_LNCRNAS` define the prefix to use for ORFs overlapping with several genes or lncRNAs.
- `PREFIX_INTERGENIC_GENE` defines the prefix to use for ORF located in intergenic regions.
- `PREFIX_CONFLICT_GENE_TRANSCRIPT` defines the prefix to use for genes on related to a transcript that for which the actual genes information is conflicting.
- `FAKE_CROSSREF` defines the string to use for cross-references of fake genes.
- `ORF_SPLICING_COORD_SEPARATOR` defines the separator to use for exon coordinates in the *DSORF* and *ORF* tables.
- `OTA_LIST_VALUES_SEPARATOR` defines the separators to use for list of values in the *ORFTranscriptAsso* table.
- `DENCELLORFOBJ_AMBIGUOUS_ATT` defines the flag to use for conflicting information from a data source.
- The **Constants relative to the Metadata table** section defines strings to use in the metadata tables.
- `SEQUENCE_AMBIGUOUS_DNA_BASE` defines the letter to use for ambiguous nucleotide.
- `SEQUENCE_AMBIGUOUS_PROT_AA` defines the letter to use for ambiguous amino acid.
- `MAX_ENTRIES_PER_DATAFRAME` defines the maximal number of entries that may be stored in data frame when computing the relative coordinates.
- The **Constants relative to the ORF annotations** section defines values to use for the ORF annotations and categories.

List of available options

The following options can be used with **all** strategies: - `-v`, `--verbosity`: Set the level of verbosity.

The following options are **common** to several strategies: - `-T`, `--databaseType`: Type of database (MySQL or SQLite). - `-c`, `--configfile`: Absolute path to the config file. - `-N`, `--databaseName`: The database name. - `-M`, `--databaseModel`: The schema of the database (PRO / DS). - `-H`, `--databaseHost`: The IP of the database host. - `-P`, `--databasePort`: The port to use to establish the connection to the database. - `-u`, `--databaseUser`: The username to use to connect to MySQL server. - `-p`, `--databasePassword`: The password to use to connect to MySQL server. - `-F`, `--databaseFolder`: The folder of the database. - `-t`, `--threads`: Number of threads available. If not provided the program try to use all the threads available on the computer.

The following options are specific to one strategy:

DatabaseCheck strategy: - `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one.

AddReleaseVersion strategy: - `-r`, `--releaseNumber`: The tag of the version. - `-d`, `--releaseDescription`: The description of the version.

Insertion strategy: - `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one and to run the insertion.

ForceInsertion strategy: - `-s`, `--source`: A comma-separated list of the data source to insert in the database (without space), in the order in which they need to be inserted. You must use `UTGeneFromAlias` as a source name

to re-insert data computed after the insertion of the last gene list and prior to the first data source name if you are re-inserting both gene lists and data sources.

Deletion strategy: - `-s`, `--source`: A comma-separated list of the data source to delete from the database (without space).

Merge strategy: - `-f`, `--forceOverwrite`: Delete any existing PRO database at the provided path / on the server prior to build a new one. The DS database will not be affected. - `-d`, `--checkDSOTA`: Should the content of the DSORFTranscriptAsso table need to be check prior to run the strategy? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option. - `-s`, `--computeConsensus`: Should a consensus of the DSORFTranscriptAsso sequences be computed? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option.

ResumeMerge strategy: - `-d`, `--checkDSOTA`: Should the content of the DSORFTranscriptAsso table need to be check prior to run the strategy? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option. - `-s`, `--computeConsensus`: Should a consensus of the DSORFTranscriptAsso sequences be computed? Be aware that selecting this option may be highly time-consuming. We advice to provide as many threads as possible when using this option. - `-a`, `--resumeAtStep`: The name of the step at which the merging should be resumed.

computeMissingInfo strategy: - `-f`, `--forceOverwrite`: Enforce the computation of all steps, including the ones that already succeed. When this option is not selected, the strategy will resume from where it failed. - `-d`, `--downloadMissingInfo`: Download the missing information (such as ORF and Transcript sequences) from external databases. Note that selecting this option may be highly time-consuming.

ComputeRelCoord strategy: - `-f`, `--forceOverwrite`: Compute again any existing relative coordinates.

ComputeKozakContext strategy: - `-f`, `--forceOverwrite`: Should all Kozak contexts be computed again?

AnnotateORF strategy: - `-f`, `--forceOverwrite`: Delete all the entries of the ORFCategory and ORFCategoryCatalog and/or of the ORFAnnotation and ORFAnnotationCatalog tables (PRO database, depending on the other options selected) prior to run the strategy. - `-s`, `--computeCatFromSource`: Compute the ORF categories from the categories provided by the datasource (ORFCatagory table). - `-a`, `--computeAnnot`: Annotate ORFs using our own algorithm based on length, biotype, strand and relative position (ORFAnnotation table).

Filter strategy: - `-f`, `--forceOverwrite`: Delete any existing FILT database at the provided path / on the server prior to build a new one. The PRO database from which data is get will not be affected.

GenerateFastaFile strategy: - `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to build a new one. - `-s`, `--seqType`: Type of the sequence required (`DNA` or `PROT`). - `-q`, `--queryTable`: Table to query to generate the FASTA file (`ORF` for *ORF* table, `OTA` for *ORFTranscriptAsso* table). - `-e`, `--excludeSqcesWithStop`: If selected, all the sequences that contains stop codons (at any other place that their end) will be excluded of the fasta file. - `-l`, `--longHeader`: Use this option to get long fasta headers. - `-o`, `--outputFolder`: The absolute path to the folder in which the GFF file has to be saved. - `-a`, `--fastaFilename`: The name for the FASTA file generated (without its extension).

GenerateBEDFile strategy: - `-o`, `--outputFolder`: Absolute path to the folder in which the BED file has to be saved. - `-b`, `--bedFilename`: Name for the BED file generated (without ".bed" extension). - `-l`, `--trackLine`: Add the track line at the beginning of the BED file. - `-n`, `--includeNonConventionalChr`: Should the ORFs located on "non conventional" chromosomes (*e.g.* mitochondrial, scaffold) be included in the BED file? - `-e`, `--extendBed`: Extend the BED file at 12+5 format. - `-g`, `--bigBed`: Convert the BED file at the BigBed.

GenerateTrackDbFile strategy: - `-o`, `--outputFolder`: Absolute path to the folder in which the trackDb file has to be saved - `-f`, `--trackFilename`: Name for the trackDb file generated (without ".txt" extension). - `-g`, `--bigBed`: Create the bigBed file corresponding to the trackDb file at the same time. See the **Export the ORF information at BED format** section of the current manual for more information. If selected, both the BED, bigBed, `.as` and `.chrom.sizes` files will be generated in the same folder than the `trackDb.txt` file. The output generated by the use of this option are the same than the one generated using the **GenerateBEDFile** strategy **with** `--extendBed` and `--bigBed` options and **without** `--includeNonConventionalChr` options.

GenerateGFFFile strategy: - `-o`, `--outputFolder`: Absolute path to the folder in which the GFF file has to be saved. - `-g`, `--gffFilename`: Name for the GFF file generated (without the ".gff" or ".gff3" extension).

AssessDatabaseContent strategy: - `-o`, `--outputFolder`: Absolute path to the folder in which all the files generated have to be saved. - `-f`, `--filename`: Name for the log file generated.

GenerateStatFiles strategy: - `-f`, `--forceOverwrite`: Overwrite any existing files. - `-o`, `--outputFolder`: Absolute path to the folder in which all the files generated have to be saved.

Backup strategy: - `-o`, `--outputFolder`: Absolute path to the folder in which the files have to be saved. - `-x`, `--filePrefix`: Prefix to add to the file names where data are saved.

Restore strategy: - `-f`, `--forceOverwrite`: Delete any existing database at the provided path / on the server prior to restore the database from backup. - `-i`, `--inputFolder`: Absolute path to the folder in which the files are located. - `-x`, `--filePrefix`: Prefix used when generated the files with the Restore strategy.

List of default values

The following values are used by default when no provided in the config file or by an option:

- `DATABASE_HOST_IP`: 127.0.0.1
- `DATABASE_PORT`: 3306
- `DATABASE_USER_NAME`: root
- `DATABASE_USER_PASSWD`: DenCellORFMySQL
- `DATABASE_FOLDER`: /output
- `GENOMIC_LENGTH_DIFF_THRESHOLD`: 1
- `SEQUENCE_CONSENSUS_AMBIGUOUS_THRESHOLD`: 2/3
- `MAX_LEN_DIFF_FOR_DSOTA_CLUSTERS`: 3
- `CELL_CONTEXTS_DICTIONARY`:

```
{   HSAPIENS:   {
                    'BJ':               [ 'loayza_puch_2013', 'rooijers_2013', 'ji_BJ_2015' ],
                    'B_cell':           [ 'B cells' ],
                    'Blood':            [ 'mills_2016' ],
                    'Brain':            [ 'gonzalez_2014' ],
                    'Brain_tumor':      [ 'Human brain tumor' ],
                    'Breast':           [ 'ji_breast_2015' ],
                    'HAP1':             [ 'jakobsson_2017' ],
                    'HCT116':           [ 'crappe_2014' ],
                    'HEK293':           [ 'lee_2012', 'andreev_2015', 'sidrauski_2015', 'liu_2013_HEK',
                                          'liu_HEK_2013', 'ingolia_2012', 'ingolia_2014', 'calviello_2016'
                                          'iwasaki_2016', 'park_2017', 'zhang_2017' ],
                    'HEK293T':          [ 'eichorn_2014', 'jan_2014' ],
                    'HFF':              [ 'Primary human foreskin fibroblasts (HFFs)',
                                          'Primary human fibroblast (HFF)', 'rutkowski_2015' ],
                    'HeLa':             [ 'wang_2015', 'niu_2014', 'yoon_2014', 'liu_2013_HeLa', 'liu_Hela
                                          'stumpf_2013', 'park_2016', 'zur_2016', 'shi_2017' ],
                    'hES':              [ 'werner_2015', 'xu_2016' ],
                    'Jurkat':           [ 'gawron_2016' ],
                    'LCL':              [ 'cenik_2015' ],
                    'MCF7':             [ 'Loayza_Puch_2016' ],
                    'MDA-MB-231':       [ 'rubio_2014' ],
                    'MM1S':             [ 'wiita_2013' ],
                    'Monocyte':         [ 'su_2015' ],
                    'NCCIT':            [ 'grow_2015' ],
                    'RPE-1':            [ 'tanenbaum_2015', 'tirosh_2015' ],
                    'Skeletal_muscle':  [ 'wein_2014' ],
                    'THP-1':            [ 'fritsch_2012', 'stern_ginossar_2012' ],
                    'U2OS':             [ 'elkon_2015' ],
                    'Flp-In_T-REx-293': [ 'malecki_2017' ]
                },
    MMUSCULUS:  {
                    '3T3':              [ 'eichorn_3t3_2014' ],
                    'BMDC':             [ 'jovanovic_2015', 'fields_2015' ],
                    'B_cell':           [ 'eichorn_bcell_2014' ],
                    'Brain':            [ 'gonzalez_2014_mmu', 'cho_2015', 'laguesse_2015' ],
```

```
                'C2C12':            [ 'deklerck_2015' ],
                'E14':              [ 'ingolia_2014_mmu', 'Ingolia_2011' ],
                'Glioma':           [ 'Mouse gliomal cells', ' Mouse gliomal cells' ],
                'Liver':            [ 'Mouse liver cell', 'eichorn_liver_2014', 'gao_liver_2014',
                                      'gerashchenko_2016', 'janich_2015' ],
                'MEF':              [ ' Mouse Embryonic Fibroblast (MEFs)', 'Mouse Embryonic Fibroblas
                                      'thoreen_2012', 'lee_2012_mmu', 'gao_mef_2014',
                                      'reid_er_2016', 'reid_cytosol_2016', 'reid_2014' ],
                'MESC':             [ 'Mouse Embryonic Stem Cells' ],
                'NSC':              [ 'katz_2014' ],
                'Neutrophil':       [ 'guo_2010_mmu' ],
                'R1E':              [ 'you_2015' ],
                'Skin_tumor':       [ 'blanco_2016' ],
                'Spleen_B_cell':    [ 'diaz_munoz_2015' ],
                'Testis':           [ 'castaneda_2014' ],
                'v6-5':             [ 'hurt_2013' ]
            }
}
```

- `CATEGORY_ASSOCIATION_DICTIONARY`:

```
{
    'None':             ['NO_FRAME', 'other', 'TEC', 'In'],
    'Intergenic':       ['lincRNA', 'INTERGENIC', 'intergenic'],
    'Upstream':         ['5UTR', '5_UTR', 'uoORF', 'uORF', 'utr5'],
    'Intronic':         ['INTRON', 'iORF', 'intronic', 'RETAINED_INTRON'],
    'Exonic':           ['EXON', 'exonic'],
    'ncRNA':            ['lincRNA', 'lncrna', 'ncRNA'],
    'Pseudogene':       ['pseudogene'],
    'Downstream':       ['3UTR', '3_UTR', 'dORF', 'utr3'],
    'NSD':              ['NSD'],
    'NMD':              ['NMD'],
    'Overlapping':      ['CDS_overlap', 'uoORF', 'ANTISENSE'],
    'sORF':             ['sORF'],
    'CDS':              ['annotated', 'Annotated', 'Isoform'],
    'Alternative':      ['Out', 'alternative_frame']
}
```

- Default output folder: /output
- Default temporary folder: /output/.tmp
- Default ambiguous letter to use for unknown nucleotide: N
- Default ambiguous letter to use for unknown amino acid: X

Additional information

- List of accepted species in the config file: `Hsapiens`, `Mmusculus`.
- List of files that can be parsed:
  - Cross-references:
    - `HGNCGeneList`: List of cross-reference provided by the HUGO Gene Nomenclature Committee (HGNC).
    - `NCBIGeneList`: List of cross-references provided by the NCBI.
  - ORF datasources:
    - `Johnstone2016`: Dataset from Johnstone *et al.*, *EMBO*, 2016 ("Location and translation data for all analyzed transcripts and ORFs", datasets EV2 and EV3).
    - `Mackowiak2015`: Dataset from Mackowiak *et al.*, *Genome Biol.*, 2015 ("All sORF information", Table S1 and S2).
    - `Samandi2017`: Dataset from Samandi *et al.*, *eLIFE*, 2017 ("Alternative protein predictions", TSV files).

    * `sORFs_org`: Databases from sORFs.org, Olexiouk *et al.*, *Nucl. Ac. Res.*, 2018, databases download from sORFs.org using the Biomart Graphic User Interface. The ORF data source name used in the config file **must** be `sORFs_org_Human` or `sORFs_org_Mouse`.
    * `Erhard2018`: Dataset from Erhard *et al.*, *Nat. Meth.*, 2018 ("Identified ORFs", supplementary table 3).
    * `Laumont2016`: Dataset from Laumont *et al.*, *Nat. Commun.*, 2016 ("List of all cryptic MAPs", supplementary data 2).

- List of accepted genome annotation version:
  - Hsapiens: GRCh38 / hg38, GRCh37 / hg19.
  - Mmusculus: GRCm38 / mm10.

Authors, license and copyright

Sébastien A. Choteau1,2, Lionel Spinelli1,2, Philippe Pierre2, Christine Brun1,3

1. Aix-Marseille Univ, INSERM, TAGC, CENTURI, Marseille, France
2. Aix-Marseille Univ, CNRS, INSERM, CIML, CENTURI, Marseille, France
3. CNRS, Marseille, France

If you have questions or comments, please write to:

- Sébastien A. Choteau at sebastien.choteau@univ-amu.fr

Last update of the manual: 25/05/2020