SMARTER Database Documentation

Release 0.4.9

cnr-ibba

Sep 27, 2023

CONTENTS

This documentation describe how the SMARTER database is made. This project is started from cookiecutter data science project template and was then adapted to model additional requirements. In order to work correctly with this project, you will need both Docker and Docker Compose installed and configured for your user. You will need also Anaconda to manage project dependencies and installing required software. After that you need to configure some stuff in order to properly use this project.

This project is currently managed with git: you should track only scripts or dependencies, don't try to track `data` folder (which can be very large and could change in any times) the `.env` configuration files (for security reasons) or any other files declared in `.gitignore` file. By following the instruction in *Getting started* section, you will be able to run your local instance of the *SMARTER-database* project!

CHAPTER

ONE

BACKGROUND

Small ruminant populations play a fundamental role for the livelihood and socio-economic well-being of human settlements, especially in marginal areas of Europe. Under-utilized sheep and goat breeds may be highly valuable in increasing the profitability of small ruminant farming in such marginal areas. These breeds are valuable because they have peculiar and often atypical genetic make-up which make them a potentially extraordinary resource to be exploited for adaptation to (harsh) environments, resilience to farming conditions, resistance to biotic and abiotic stressors, and the production of quality of products of animal origin (see Biscarini et al. 2015).

The SMARTER-database GitHub project is a collection of tools and scripts to collect, standardize and provide to the partners of the SMARTER Work Package 4 (and later to the whole community) a collection of genotype data and metadata information in hardy goat and sheep populations by combining new with already existing datasets. Those data can be exploited to characterize the genetic diversity and demography of sheep and goat breeds with a particular focus on underutilized breeds and to contribute to understanding the genetic basis of resilience and adaptation to the environment of hardy breeds.

This project is part of the SMARTER project which aims to develop and deploy innovative strategies to improve Resilience and Efficiency (R&E) related traits in sheep and goats.

CHAPTER

TWO

DOCUMENTATION CONTENTS

2.1 Getting started

Table of Contents

2.1.1 The SMARTER-database project

The SMARTER-database projects is a repository where partners of Work Package 4 (WP4) of the SMARTER project can share their genotype and phenotypes data. The main objective of this WP is to quantify the genetic diversity in hardy and underutilized breeds and identify signatures of selection related to specific breed adaptation to geo-climatic environments. New and available data on R&E phenotypic and genotypic information on different breeds from partners, from previous projects and from other WPs will be used to develop strategies to combine such heterogeneous data. To accomplish this task, data need to be standardized, merged and then referred to their metadata.

The SMARTER-database project is a collection of scripts and code to standardize and integrate information in an unique place available to WP4 partners and later to the all community. Processed genotype data will be available through FTP, while metadata will be available through the SMARTER-backend with the help of the r-smarter-api R package and SMARTER-frontend.

This project is structured as described by Cookiecutter Data Science documentation: the key idea is to structure a data science project in a standardized way. Every folder within the project has a precise scope which is described in both Cookiecutter Data Science documentation and in README.md. All data produced within this project is reproducible and the structure imposed by this project let people to understand where to find data or code of interest in order to get information on a certain element without having a full understanding of every script/module/data file inside this project.

There are two major distinct areas regarding the SMARTER data: The first is the database related folder, which keep information regarding the SMARTER MongoDB instance and is managed using docker-compose. This database need to be up and running in order to work properly with SMARTER data. Moreover, database need to be populated with data like SNP coordinates which comes from SNPchimp, Ensembl or EVA. There's also the need to upload data coming from custom chips, in order to have a more precise picture of all the variants. Data need also to be integrated with additional information like breeds and their codes. All those steps are managed by python scripts which are stored in the second area: those scripts let to interact with data relying on the MongoDB instance and transform the content in the `data/raw` folder into `data/processed` output, which is the final output generated by the *SMARTER-database* project

SMARTER-database requirements

SMARTER-database is managed through a conda environment, in which python executable and other non-python dependencies are specified. Moreover, python dependencies are managed through a `requirements.txt` file. Dependencies and environment set up is managed through the GNU linux `make` command. The MongoDB instance of this project is managed by docker and docker-compose, however you can configure an environment variable to set up a connection with an external MongoDB instance. See *Configure environment variables* for more information.

2.1.2 Installation and configuration

Clone this project with GIT

In order to install *SMARTER-database* project, you need to clone it from GitHub using git:

```
git clone https://github.com/cnr-ibba/SMARTER-database.git
```

Now enter into the smarter cloned directory: from now and in the rest of this documentation this `SMARTER-database` directory will be referred as **the project home directory**:

```
cd SMARTER-database
export PROJECT_DIR=$PWD
```

Note: If you plan to install this project in a shared folder, take a look before at Shared folders and permissions and in particular at the Setting permissions section in the BIOINFO Guidelines documentation

Tip: In order to better share this project with other users on the same machine, its better to clone this project inside a directory with the **SGID** special permission (see Using SGID for more information)

Warning: Every file you create in a **SGID** directory will have the correct permissions and ownership, however if you **copy** a file through `scp`, `rsync` or you move a file from a non **SGID** directory, the permission will be the standard ones defined for your user. You should check that permissions are correct after *moving* or *copying* files,

in particular for `data` directory. To add the **SGID** permission on the current directory and subfolder, you could do like this:

```
find . -user $USER -type d -exec chmod g+s {} \;
```

This command should be called inside a *interactive bash login session*, since bash will ignore commands which try to set the **SGID** permission.

Configure environment variables

In order to work properly *SMARTER-database* needs some environment variables defined in two environment files. Those files **must not be tracked with GIT** for security reasons, and should be defined **before** start working with this project.

The first `.env` file is located inside the `database` folder and is required in order to start the MongoDB and mongoexpress images and to set up the required collections and validation constraints. So edit the `$PROJECT_DIR/database/.env` file by setting these two variables:

```
MONGODB_ROOT_USER=<smarter root database username>
MONGODB_ROOT_PASS=<smarter root database password>
MONGOEXPRESS_USER=<smarter mongoexpress username>
MONGOEXPRESS_PASS=<smarter mongoexpress password>
```

The second `.env` file need to be located in the **project HOME directory** and need to define the credentials required to access the MongoDB instance using a new *smarter* user (a user granted to fill up the database and to retrieve information to process the genotype files). Start from this template and set your credentials properly in `$PROJECT_DIR/.env` file:

```
# Environment variables go here, can be read by `python-dotenv` package:
#
#   `src/script.py`
#   ----------------------------------------------------------------
#    import dotenv
#
#    project_dir = os.path.join(os.path.dirname(__file__), os.pardir)
#    dotenv_path = os.path.join(project_dir, '.env')
#    dotenv.load_dotenv(dotenv_path)
#   ----------------------------------------------------------------
#
# DO NOT ADD THIS FILE TO VERSION CONTROL!
MONGODB_SMARTER_USER=<smarter username>
MONGODB_SMARTER_PASS=<smarter password>
MONGODB_SMARTER_HOST=localhost
MONGODB_SMARTER_PORT=27017
```

Hint: You can configure the MongoDB instance on a different host, or call the import process from another location by setting the proper `MONGODB_SMARTER_HOST` and `MONGODB_SMARTER_PORT` values in the environment file.

Start the MongoDB instance

The *MongoDB* instance is managed using `docker-compose`: database will be created and configured when you start the docker container for the first time. Local files are written in the `$PROJECT_DIR/database/mongodb-data` that will persist even when turning down and destroying docker containers . First check that the `$PROJECT_DIR/database/.env` file is configured correctly as described by the section *before*. Next, in order to avoid annoying messages when saving your mongo-client history, set `mongodb-home` *sticky dir* permission:

```
cd $PROJECT_DIR/database
chmod o+wt mongodb-home/
```

This let you to save and see mongodb history using a different user than the user used inside the MongoDB docker container. Moreover, this folder can be used to import/export a *SMARTER-database* dump. Next download, build and initialize the *SMARTER-database* containers with:

```
docker-compose pull
docker-compose build
docker-compose up -d
```

Now is time to define create the *smarter* user with the same credentials used in your `$PROJECT_DIR/.env` environment file. You could do this using *docker-compose* commands:

```
docker-compose run --rm --user mongodb mongo sh -c 'mongo --host mongo \
  --username="${MONGO_INITDB_ROOT_USERNAME}" \
  --password="${MONGO_INITDB_ROOT_PASSWORD}"'
```

Then from the mongodb terminal create the *smarter* user using the values of `$MONGODB_SMARTER_USER` and `$MONGODB_SMARTER_PASS` variables. You require both the *read/write* privileges to update and retrieve smarter data:

```
use admin
db.createUser({
  user: "<user>",
  pwd: "<password>",
  roles: [{
    role: "readWrite",
    db: "smarter"
  }]
})
```

For more information on the smarter *MongoDB* database usage, please refer to the README.md documentation in the `$PROJECT_DIR/database` folder.

Setting up python environment

In order to install all the conda requirements and libraries, move into the `$PROJECT_DIR` (which is the *SMARTER-database* folder cloned using git) and then install dependencies using make:

```
cd $PROJECT_DIR
make create_environment
```

This will create a `SMARTER-database` conda environment and will install all the required softwares (like plink, vcftools, tabix, ...). Then you need to manually activate the `SMARTER-database` before installing all the required python dependencies:

```
conda activate SMARTER-database
make requirements
```

Note: All project dependencies will be installed in the `SMARTER-database` conda environment. You will need to activate this environment every time you need to use a *SMARTER-database* script or dependency.

2.1.3 Initialize and populate SMARTER database

In order to populate the *SMARTER-database* with data, you need to collect data provided by the partners from the SMARTER repository. Moreover you have to retrieve and collect information from databases like SNPchiMp, Ensembl or EVA. You will need also information from *Illumina* or *Affymetrix* Manifest files in order to deal with different types of genotype files. *Raw unprocessed files* and external *sources files* need to be placed in their proper folder: all data received by the SMARTER partners need to be placed in the `data/raw` folder in the SMARTER `$PROJECT_DIR` directory, in a `foreground` or `background` folder accordingly if data is produced in the context of SMARTER project or is available outside this project. External source files, like manifests, database dumps and other support files need to be placed in the `data/external` directory. Within this project external support files are organized by species (`GOA` and `SHE` for *goat* and *sheep* respectively) and by data source (ie, `SNPCHIMP`, `ILLUMINA` `AFFYMETRIX`, etc.). Those data files are not shipped with this github project, you need to ask to developer and to SMARTER WP4 coordinators to have access to this data.

Process raw data and create the final dataset

In order to process raw data, insert data into SMARTER database, generate the SMARTER ids an create the final genotype dataset files there are manly two steps that are managed using `make` command. In the first step, you will upload all the external information into the database: simply type (inside the `SMARTER-database` conda environment):

```
make initialize
```

to upload all the external information on *variants* in the database. This step is described in detail in the *Loading variants into database* section.

In the next step, you will process each sample by generating a *SMARTER ID*, and you will insert phenotypes and other sample related metadata into the SMARTER database. The final output of this step will be the generation of the final genotype files. Like before, simply type:

```
make data
```

Output data will be placed in a folders relying on the assembly version used, with all the genotypes in the same format and using the same reference system. Those folders will be placed in the `data/processed` folder. For more detailed information about all the process called within this step, please see *The Data Import Process* documentation. Last step in data generation is made available with:

```
make publish
```

which will pack your genotype files in order to be shared with other partners using the SMARTER FTP repository.

2.1.4 Database management through docker-compose

The SMARTER MongoDB docker-composed image in `database` folder does a *mount bind* of the `database/mongodb-home/` folder in which you can put files that could be inserted / retrieved from database. This means that you can place here a file to be imported into database or you can export a collection outside *SMARTER-database*. Here are described how to dump and restore a full *SMARTER-database* instance:

Restore SMARTER database from a *mongodump* file

In order to restore the SMARTER database from a dump file:

```
docker-compose run --rm --user mongodb mongo sh -c 'mongorestore --host mongo \
  --username="${MONGO_INITDB_ROOT_USERNAME}" \
  --password="${MONGO_INITDB_ROOT_PASSWORD}" --authenticationDatabase admin \
  --db=smarter --drop --preserveUUID --gzip \
  --archive=/home/mongodb/smarter.archive.gz'
```

After that, you can login through the *smarter* database by calling the mongodb client like this:

```
docker-compose run --rm --user mongodb mongo sh -c 'mongo --host mongo \
  --username="${MONGO_INITDB_ROOT_USERNAME}" --password="${MONGO_INITDB_ROOT_PASSWORD}" \
  --authenticationDatabase=admin smarter'
```

Dump SMARTER-database

In order to dump SMARTER database in a file:

```
docker-compose run --rm --user mongodb mongo sh -c 'mongodump --host mongo \
  --username="${MONGO_INITDB_ROOT_USERNAME}" \
  --password="${MONGO_INITDB_ROOT_PASSWORD}" --authenticationDatabase admin \
  --db=smarter --gzip --archive=/home/mongodb/smarter.archive.gz'
```

2.2 Loading variants into database

Table of Contents

When calling `make initialize` a series of steps are performed to initialize the MongoDB database by loading variants information. This process is independent from the *data generation* step, but those information are used to convert

genotype files to the same format and produce the final genotype dataset. Information on variants need to be loaded for both SMARTER species (*goat* and *sheep*) in order to do this data conversion. Some accessory information are also required, for example `rs_is`, since the same SNP can be called with different names.

2.2.1 The variant collections

Mainly the variations are modelled around Illumina variants described in their chips, since the majority of genotype files are produced using this technology. However, those data need to support also the Affymetrix manufacturer and even data produced from Whole Genome Sequencing (WGS). To accomplish this, variants have additional fields as described by *VariantSpecies* class, in order to support different sources of information. Variants derived from Affymetrix technology or by WGS are retrieved and replaced with the proper Illumina variants when possible, in order to make possible the comparison between samples derived from different technologies. When genotypes are processed during the *data import process*, variants are retrieved relying their names or attributes like genomic positions or `rs_id`, then genotypes are checked against database and converted with *Illumina TOP* coding convention.

2.2.2 About supported assemblies

Received genotype data could come from different chips which relies on different assemblies. Data generated long time ago could refer a very old or deprecated genome assembly, and even data generated with the same chip could have different positions since the probe mapping is a process continuously under revision. Considering this, we can't trust genomic positions when processing genotype files, the only thing stable in genotype files is the SNP *name*: this is the reason why we chose to use SNP names to update genomic positions. We upload evidences for different assemblies and chips in order to represent every SNP processed in genotype file, but we convert genomic positions and genotypes relying only on one evidence: this could introduce some errors maybe non present in latest assemblies, however this genome assembly is consistent between genotype files, and this let to compare genotypes across different dataset produced by different platforms.

When we first upload a SNP, we assign an initial *Location* object with a `version` and `imported_from` attributes in which track the genome assembly *version* and the *source* of information. This let us to further update the same location, if the assembly and the source is the same (for example, with a more recent manifest file) or store another *Location* object to manage a new genomic position from a different evidence. This let us also to switch from one assembly to another one, since all the available genomic locations are stored within the SNP itself.

At this time, the genome assemblies we support are `OAR3` for *sheep* and `ARS1` for *goat* genome: they are not the latest assembly versions, however they are supported by genome browser like Ensembl or UCSC. We plan to support more recent assemblies to facilitate the data sharing in the future.

2.2.3 Upload the supported chips

First step in database initialization is loading the supported chip into *SupportedChip* documents. You need to prepare a JSON file in which at least the chip name and the species is specified: This chip name will be assigned to *VariantSpecies* defined within this chips and also to *SampleSpecies* and *Dataset*. Here is an example of such JSON file:

```
[
    {
        "name": "IlluminaOvineSNP50",
        "species": "Sheep",
        "manifacturer": "illumina",
        "n_of_snps": 0
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "name": "WholeGenomeSequencing",
        "species": "Sheep"
    }
]
```

Next, you can upload the chip name using *import_snpchips.py*:

```
python src/data/import_snpchips.py --chip_file data/raw/chip_names.json
```

For more information, see *import_snpchips.py* manual page.

2.2.4 Import SNPs from manifest files

In order to define a *VariantSpecies* object, you need to load such SNP from a manifest file and specify the source of such location. After a SNP object is created, you can add additional location evidences, or update the same genomic location using a more recent manifest file. Since this database is modelled starting from Illumina chips, its better to define all the Illumina SNPs before: after that, if an Affymetrix chip has a correspondence with a SNP already present, the new location source can be integrated with the Illumina genotype. To upload SNP from an illumina manifest file, simply type:

```
python src/data/import_manifest.py --species_class sheep \
    --manifest data/external/SHE/ILLUMINA/ovinesnp50-genome-assembly-oar-v3-1.csv.gz \
    --chip_name IlluminaOvineSNP50 --version Oar_v3.1 --sender AGR_BS
```

where the `--species_class` must be one of *sheep* or *goat* and `--manifest`, `--chip_name` and `--version` need to specify the manifest file location, a *SupportedChip.name* already loaded into database and the assembly version. To upload data from an Affymetrix manifest file, there's another script:

```
python src/data/import_affymetrix.py --species_class sheep \
    --manifest data/external/SHE/AFFYMETRIX/Axiom_BGovis2_Annotation.r1.csv.gz \
    --chip_name AffymetrixAxiomBGovis2 --version Oar_v3.1
```

where the parameters required are similar to the Illumina import process. For more information see *import_manifest.py* and *import_affymetrix.py* manual pages.

2.2.5 Import locations from SNPchiMp

Another useful source of information come from the SNPchiMp database, which was a project in which SNPs belonging to Affymetrix or Illumina manufacturers where loaded with their genome alignment from dbSNP database: This lets to convert coordinates and genotypes between different genomic assemblies. Unfortunately, after dbSNP release `151` SNPs from animals like *sheep* and *goat* are not more managed by NCBI but were transferred to EBI EVA. This implies update importing script and update database like SNPchiMp. At the moment SNPchiMp data are the main data used from assemblies `OAR3`, `OAR4` and `CHI1`, while `ARS1` assembly is currently managed from manifest file (which is more recent than SNPchiMp). We plan to re-map the probes and to integrate data with EVA, in order to solve genomic locations for all the SNPs and having the latest evidences and *cross-reference* id like `rs_id`. To upload data from SNPchiMp, simply download the entire datafile for a certain assembly and chip. Then call the following program:

```
python src/data/import_snpchimp.py --species_class sheep \
    --snpchimp data/external/SHE/SNPCHIMP/SNPchimp_SHE_SNP50v1_oar3.1.csv.gz \
    --version Oar_v3.1
```

see *import_snpchimp.py* manual page for additional information.

2.2.6 Import locations from genome projects

The last source of evidence that is modelled by SMARTER-database comes from *sheep* and *goat* genome initiatives like Sheep HapMap or VarGoats, which can re-map chips on latest genome assemblies. However, this mapping process can have some issues (see here, here and here for example) so this source of evidence need to be revised with *sheep* and *goat* genomic projects. To upload this type of information in database, you can do as following:

```
python src/data/import_isgc.py \
    --datafile data/external/SHE/CONSORTIUM/OvineSNP50_B.csv_v3.1_pos_20190513.csv.gz \
    --version Oar_v3.1
python src/data/import_iggc.py \
    --datafile data/external/GOA/CONSORTIUM/capri4dbsnp-base-CHI-ARS-OAR-UMD.csv.gz \
            --version ARS1 --date "06 Mar 2018" --chrom_column ars1_chr --pos_column␣
↪ars1_pos \
    --strand_column ars1_strand
```

please, refer to *import_isgc.py* and *import_iggc.py* manual pages for additional information.

2.3 The Data Import Process

Table of Contents

When calling the `make data` step of SMARTER-database data generation, a series of steps are performed in order to process raw data and to generate the final dataset. This document tries to describe how the data import process works and how to add new data to the SMARTER-database.

To add a new dataset into SMARTER-database, you need to call the following scripts with specific option in `data` section of the `Makefile` file. The order in which those import scripts are called matters, since importing a sample into SMARTER-database means generating a unique `smarter_id`, which need to be stable, in order to track the same object when updating the database or in different releases. Scripts are written in order to be idempotent: calling the same script twice with the same parameters will produce the same final result.

2.3.1 Defining a new dataset

The data import process start by defining a dataset as a `.zip` archive, which could contain *genotype* or *phenotype* information (or other metadata). Dataset can also be classified as *foreground* or *background* respectively if they are generated in the context of the *SMARTER* project or before it. Accordingly to data source type and provenience, you have to define a record in the proper `.csv` file in `data/raw` folder, like the following:

```
#;File;Uploader;Size;Partner;Country;Species;Breed;N of Individuals;Gene Array;Chip Name
3;ADAPTmap_genotypeTOP_20161201.zip;smarterdatabase-admin;43.68MB;AUTH;36 Countries;Goat;
↪144 breeds;4653;Genotyping data in plink binary format;IlluminaGoatSNP50
```

Next, dataset need to be imported by calling `src/data/import_datasets.py` with the proper dataset type and input file, like the following example:

```
python src/data/import_datasets.py \
    --types genotypes background \
    data/raw/genotypes-bg.csv
```

This command will add this dataset as a new *Dataset* object into the SMARTER-database and will unpack its content in a folder with the *MongoDB* `ObjectID` inside the `data/interim` folder. This let you to analyze and process the dataset content using the SMARTER-database `src` code. For more information, see the *import_datasets.py* help.

2.3.2 Exploring data with Jupyter Lab

Before importing genotypes and samples into the SMARTER-database, there is an additional *data exploration* step using Jupyter Lab: this step requires manual intervention to understand if data could be imported as it is or if some fixing steps are required. Common issues in datasets could be having different breeds with the same code, or using different codes to specify the same breeds. There could be the case where sample names within the genotype file are different from the ones used in metadata: in all those cases, you have to define a new metadata file where there will be a correspondence between the used value and the value to be inserted into the SMARTER-database.

In this data exploration step, you could check also the coding format of genotypes, by calling the proper *SmarterMixin* derived class. Metadata can also be integrated with external data sources, which can be used to fix some stuff related to metadata. Start Jupyter Lab (in an activated conda environment) with:

```
jupyter lab
```

then create a new notebook according your needs. Those notebook can also be used after the data ingestion to produce reports about the SMARTER-database status. Please, see the notebook section in the Cookiecutter Data Science project for more information.

2.3.3 Adding breeds to the database

Before upload samples into SMARTER-database, you have to register a *Breed* first: If the dataset have one or few breeds, you could define a new breed object by calling *add_breed.py* like this:

```
python src/data/add_breed.py --species_class sheep \
    --name Texel --code TEX --alias TEXEL_UY \
    --dataset TEXEL_INIA_UY.zip
```

where the `--species_class` parameter specifies the source species (*goat* or *sheep*), `--name` and `--code` specify the breed *name* and *code* used in the SMARTER-database respectively, the `--alias` specifies the FID (the *code*) used in the genotype file and the `--dataset` parameter specifies the dataset sources of the sample we want to add. If you have

to manage many different breeds in the same dataset, it's better to create breeds from a metadata file. In such case, you can create your new breeds with a different script:

```
python src/data/import_breeds.py --species_class Sheep \
    --src_dataset=ovine_SNP50HapMap_data.zip \
    --datafile ovine_SNP50HapMap_data/kijas2012_dataset_fix.xlsx \
    --code_column code --breed_column Breed \
    --fid_column Breed --country_column country
```

in such case, we will have a `--src_dataset` and `--dst_dataset` which let to specify the dataset where the metadata information are retrieved (using the `--datafile` option) and the dataset where these information will be applied: parameters like these can be provided to other import scripts which rely on a metadata file and one or two distinct datasets. The other parameters let to specify which columns of the metadata file will be used when defining a new breed. See *import_breeds.py* documentation for more information.

Note: Breed `name` and `code` are unique in the same species (enforced by MongoDB): if you have the same breed in two different dataset, you need to call those command twice: first time you will create a new *`Breed`* object with the alias used in the first dataset. Every other call on the same breed, will update the same object to support also the new alias in the other dataset.

2.3.4 Adding samples to the database

Samples can be added in two ways: the first is when converting data from genotype files, the second is by processing metadata information. The first approach should be used when you have a single breed in the whole genotype file, and the breed `code` in the genotype file have already a *`Breed`* instance in the SMARTER-database: this is the simplest data file, when data belongs to the same country and breed. With this situation, you could create samples while processing the genotype file simply by adding the `--create-samples` flag to the appropriate importing script (for more information, see *Processing PLINK-like files*, *Processing Illumina report files* and *Processing Affymetrix files* sections)

The second approach need to be used when you have different breeds in you genotype file, or when there are additional information that can't be derived from the genotype file, like the country of origin, the sample name or the breed codes which could have different values respect to the values stored in the genotype file. Other scenarios could be *Illumina report* or *Affymetrix report* files which don't track the FID or other types of information outside sample names and genotypes. Another case is when your genotype files contains more samples than metadata file, for example, when you want to track in SMARTER-database only a few samples: in all these cases, samples need to be created **before** processing genotypes, using the *import_samples.py* script:

```
python src/data/import_samples.py --src_dataset Affymetrix_data_Plate_652_660.zip \
    --datafile Affymetrix_data_Plate_652_660/Uruguay_Corriedale_ID_GenotypedAnimals_fix.
↪xlsx \
    --code_all CRR --id_column "Sample Name" \
    --chip_name AffymetrixAxiomOviCan --country_all Uruguay \
    --alias_column "Sample Filename"
```

like *import_breeds.py*, we have `--src_dataset` and `--datafile` to indicate where our metadata file is located; if our genotype file is located in the same dataset of metadata, we can omit the `--dst_dataset` parameter. Breed codes and country can be set to the same values with the `--code_all` or `--country_all` parameters, or can be read from metadata file like the following example:

```
python src/data/import_samples.py --src_dataset greece_foreground_sheep.zip \
    --dst_dataset AUTH_OVN50KV2_CHIOS_FRIZARTA.zip \
    --datafile greece_foreground_sheep/AUTH_OVN50KV2_CHIOS_FRIZARTA.xlsx \
```

(continues on next page)

(continued from previous page)

```
    --code_column breed_code --id_column sample_name \
    --chip_name IlluminaOvineSNP50 --country_column Country
```

Please, look at *import_samples.py* help page to have more info about the sample creation process.

Note: Samples are always related to their source dataset, so you could have more samples with the same `original_id` in SMARTER-database. However, samples need to be unique in the same dataset, otherwise the genotype conversion step will not work. If your dataset contains two or more samples with the same `original_id`, you could specify an additional column (like the `alias`) to identify your samples within genotype files

---

2.3.5 Processing genotype files

Genotype data is not added into the SMARTER-database, however this data is validated *with* SMARTER-database, which track information on SNPs: in fact, genotype data could be produced long time ago and with different technologies, so assemblies don't match and genotype calls need to be standardized in order to be compared. This is particularly true when genotypes are referred according genomic sequence: since the chip probes could be aligned to the *forward/reverse* strands, the same SNPs could have different genotypes in different assembly versions. In such way, variants need to be converted in order to compare datasets produced in different times with different approaches. To accomplish this, variants need to be loaded into database from manifest, and supplementary information need to be added into the smarter database: all those steps are managed through `Makefile` by calling:

```
make initialize
```

before importing datasets into the SMARTER-database. For more information, see the *Loading variants into database* section of this documentation.

Converting genotypes to Illumina TOP

All the received genotypes are converted in **illumina TOP** format: this coding convention was introduced by Illumina and its main features is that SNP orientation is determined from the sequence around the SNP itself. This seems complex but has the advantage that the **SNPs remains the same even if the SNP database or the genome assembly changes**. In detail, illumina defines as unambiguous a SNP with only one of A or T calls: SNPs like A/G or A/C will be TOP snps; SNP with T/C and T/G are BOTTOM SNPs. All the other ambiguous cases are determined using the sequence walking method: starting from the SNPs itself, take a letter after and before and check if the resulting pair is ambiguous or not. If the pair is unambiguous, you can classify in TOP/BOTTOM. If the pair is ambiguous take the second letter after and before the SNP and check the resulting pair. This will be done until we can assign a TOP/BOTTOM coding to the SNP.

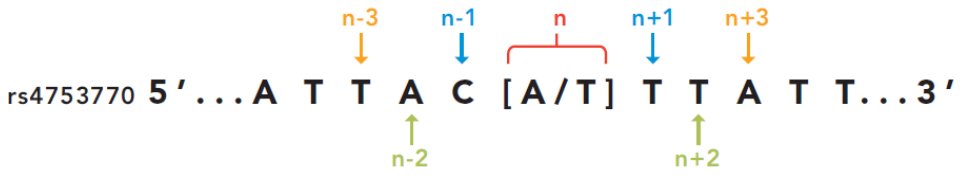

Fig. 1: Credits: Illumina technical notes

In this example A/T is ambiguous even if is composed by A and T. The first couple taken (T/C) is unambiguous so we can say that this example SNP is in BOTTOM orientation. SMARTER genotypes are converted into Illumina TOP: this means that if a SNP is already in TOP coding will be used as it is, but all the other cases need to be converted into illumina TOP. The following is an example of coding conversion for `DU186191_327.1` SNP:

Table 1: DU186191_327.1 A/G (unambiguous SNP)

smarter_id	source version	source coding	source genotype	top genotype
UYOA-CRR-000003890	OAR4	forward	T C	A G
UYOA-CRL-000000382	OAR3	A/B	A B	A G
NAOA-ADP-000001020	OAR3	top	G A	G A
GROA-CHI-000004137	OAR3	forward	T T	A A

In the previous example is easy to convert a SNP into illumina TOP: assumed that the TOP genotype is `A/G`, every time a letter `T` or `C` is found it need to be reversed. But how the snp `OAR1_103790218.1` can be converted?

Table 2: OAR1_103790218.1 C/G (ambiguous SNP)

smarter_id	source version	source coding	source genotype	top genotype
UYOA-CRR-000003890	OAR4	forward	G G	G G
UYOA-CRL-000000382	OAR3	A/B	B B	G G
NAOA-ADP-000001020	OAR3	top	G G	G G
GROA-CHI-000004137	OAR3	forward	C C	G G

This case is more complex since the `C` call is the complement of `G`, so you can't determine the coding of this genotype. The only way to determine the genotype coding of this SNP is to check the coding of the other SNPs in the same dataset. The other source of information required is the orientation of the probe to the reference genome. Consider samples `UYOA-CRR-000003890` and `GROA-CHI-000004137`: they have the same TOP genotype since the probe is aligned to different strands in `OAR3` and `OAR4` assemblies, so only one genotype need to be reversed to get a TOP genotype. All the information about SNP position and strand orientation are stored in *`Variants`* and *`Locations`* objects, and can be accessed using the proper methods. The genotype conversion is managed by the proper *`SmarterMixin`* derived class method, called by the proper importing script.

So why convert genotypes into illumina TOP? Because illumina TOP SNPs are identical in different genome assemblies, and this means that if you have a new genome version you don't need to convert the genotype, you will need only to update the genomic positions of the SNPs. For such reason, each genotype importing script has a `--coding` option with let you to specify the genotype coding of the source file. Source coding will be checked against SMARTER-database *`variant`* information in order to be converted in Illumina TOP coding.

To read more about illumina TOP/BOTTOM coding convention, please see illumina technical notes documentation and also Simple guidelines for identifying top/bottom (TOP/BOT) strand and A/B allele and How to interpret DNA strand and allele information for Infinium genotyping array data.

Processing PLINK-like files

Genotypes provided as PLINK files (both *text* or *binary*) can be imported using the *import_from_plink.py* script, like in the following example:

```
python src/data/import_from_plink.py --bfile AUTH_OVN50KV2_CHIOS_FRIZARTA/AUTH_OVN50KV2_
↪CHI_FRI \
    --dataset AUTH_OVN50KV2_CHIOS_FRIZARTA.zip --coding forward \
    --chip_name IlluminaOvineSNP50 --assembly OAR3
```

The `--bfile`/`--file` options (mutually exclusive) let you to specify a file prefix (like PLINK does) for a binary/text file respectively. The `--dataset` option lets to specify which dataset contains the genotype file; `--coding` option lets

to specify the source coding (if the provided coding does not match with database data, the import process will fail). The `--assembly` parameter will be the destination assembly version of the converted genotypes. There are also other parameter, for example when you have source genotypes with *rs_id* or when the source assembly is different from the destination assembly. For a full list os such options, take a look to *import_from_plink.py* help page.

Processing Illumina report files

Genotypes provided as Illumina reports need to be processed using another script:

```
python src/data/import_from_illumina.py --report JCM2357_UGY_FinalReport1.txt \
    --snpfile OvineHDSNPList.txt --dataset CREOLE_INIA_UY.zip --breed_code CRL \
    --chip_name IlluminaOvineHDSNP --assembly OAR3 --create_samples
```

In this case the Illumina report file need to be specified with the `--report` option, while the SNPs information file need to be specified with the `--snpfile` option. This command, like *import_from_plink.py* and *import_from_affymetrix.py* let to create samples while reading from genotypes using the `--create_samples` flag. Since illumina report files doesn't track information about FID, breed codes need to be specified using `--breed_code` parameter only for one breed samples file: files with multiple breeds can't be imported like this, samples need to be created before with *import_samples.py* in order to retrieve the correct information from SMARTER-database. Please see *import_from_illumina.py* manual pages to get other information regarding this program.

Processing Affymetrix files

Affymetrix genotypes can be provided using reports format or PLINK like format (which lacks of some columns unlike standard PLINK files). Even in this case, there will be a proper script to call and custom parameters to specify:

```
python src/data/import_from_affymetrix.py \
    --prefix Affymetrix_data_Plate_652_660/Affymetrix_data_Plate_652/Affymetrix_data_
↪Plate_652 \
    --dataset Affymetrix_data_Plate_652_660.zip --breed_code CRR --chip_name␣
↪AffymetrixAxiomOviCan \
    --assembly OAR3 --sample_field alias --src_version Oar_v4.0 --src_imported_from␣
↪affymetrix
```

In this example, the `--prefix` parameter means load data from a PLINK-like file. The other input source type could be specified with the `--report` option. Other parameters are already been described with other import script, with the exception of `--sample_field`, which let to search samples using a different attribute, and the source of the assembly (both `--src_version` and `--src_imported_from`) which is required to convert genotypes into Illumina TOP. For other information, please see the *import_from_affymetrix.py* help page.

2.3.6 Adding metadata information

Next step in the data import pipeline is importing metadata into SMARTER-database: those data can't be provided in the final genotype file, and so will be made available through the SMARTER-backend with the help of the r-smarter-api R package and SMARTER-frontend. There are two main scripts to import metadata: *import_metadata.py* and *import_phenotypes.py*. *import_metadata.py* should be used to import GPS coordinates and other generic metadata fields, while *import_phenotypes.py* should be used to import phenotypes. Both two scripts can be used to apply information to all the samples belonging to the same breed or to each sample belonging to the same datasets, relying on metadata defined for each breed group or each distinct sample. For example, to load data with GPS coordinates and additional columns you can call *import_metadata.py* like this:

```
python src/data/import_metadata.py --src_dataset "High density genotypes of French Sheep␣
↪populations.zip" \
    --datafile Populations_infos_fix.xlsx --breed_column "Population Name" \
    --latitude_column Latitude --longitude_column Longitude --metadata_column Link \
    --metadata_column POP_GROUP_CODE --metadata_column POP_GROUP_NAME
```

In this example, metadata are applied by breed using the `--breed_column`. Parameters like `--src_dataset`/ `--dst_dataset` and `--dataset` have the same behavior described in *import_samples.py*. All the additional metadata column can be loaded by calling multiple times the `--metadata_column` parameter by providing the desired column in metadata file. Similarly, this applies also for *import_phenotypes.py* as described in the following example:

```
python src/data/import_phenotypes.py --src_dataset ADAPTmap_phenotype_20161201.zip \
--dst_dataset ADAPTmap_genotypeTOP_20161201.zip \
--datafile ADAPTmap_phenotype_20161201/ADAPTmap_InfoSample_20161201_fix.xlsx --id_column␣
↪ADAPTmap_code \
--chest_girth_column ChestGirth --height_column Height --length_column Length \
--additional_column FAMACHA --additional_column WidthOfPinBones
```

This time, phenotype metadata are loaded for each sample, as described by the `--id_column` parameter. Then there are parameters which describe a single phenotype trait, like `--height_column` or `--length_column`, while additional phenotype traits not described by the *Phenotype* class, can be loaded with the `--additional_column` parameter, which can be specified multiple times.

2.3.7 Merging datasets together

Last step of data import is merging all the processed genotype files into one dataset for species/assemblies. You can do it by calling *merge_datasets.py* like this:

```
python src/data/merge_datasets.py --species_class sheep --assembly OAR3
```

This script will search all processed genotype files for the same species/assembly and will merge all the genotypes in one file. The final genotype will be placed in a new directory with the same name of the desired assembly under the `data/processed` directory.

2.4 Commands

Table of Contents

- *src/data/import_from_plink.py*
- *src/data/import_iggc.py*
- *src/data/import_isgc.py*
- *src/data/import_manifest.py*
- *src/data/import_metadata.py*
- *src/data/import_multiple_phenotypes.py*
- *src/data/import_phenotypes.py*
- *src/data/import_samples.py*
- *src/data/import_snpchimp.py*
- *src/data/import_snpchips.py*
- *src/data/merge_datasets.py*
- *src/data/SNPconvert.py*
- *src/data/update_db_status.py*

Here are the scripts called during data import by the `make initialize` and `make data` commands. For more information, see *The Data Import Process* and *Loading variants into database* documentation sections.

2.4.1 src/data/add_breed.py

Add or update a breed into SMARTER database

```
src/data/add_breed.py [OPTIONS]
```

Options

--species_class <species_class>

Required The generic species of this breed (Sheep or Goat)

Options

Sheep | Goat

--name <name>

Required The breed name

--code <code>

Required The breed code

--alias <alias>

The FID used as a breed code in genotype file

--dataset <dataset>

Required The raw dataset file name (zip archive)

2.4.2 src/data/import_affymetrix.py

Load SNP data from Affymetrix manifest file into SMARTER-database

```
src/data/import_affymetrix.py [OPTIONS]
```

Options

--species_class <species_class>
Required

--manifest <manifest>
Required

--chip_name <chip_name>
Required

--version <version>
Required

2.4.3 src/data/import_breeds.py

Import breeds from metadata file into SMARTER-database

```
src/data/import_breeds.py [OPTIONS]
```

Options

--species_class <species_class>
Required The generic species of this breed (Sheep or Goat)

Options
Sheep | Goat

--src_dataset <src_dataset>
Required The raw dataset file name (zip archive) in which search datafile

--dst_dataset <dst_dataset>
The raw dataset file name (zip archive) in which define breeds (def. the 'src_dataset')

--datafile <datafile>
Required The metadata file in which search for information

--code_column <code_column>
The name of the breed code column in metadata table

--breed_column <breed_column>
The name of the breed column in metadata table

--fid_column <fid_column>
The name of the FID column used in genotype file

--country_column <country_column>
The name of the country column in metadata table

2.4.4 src/data/import_datasets.py

Import a dataset stored in `data/raw` folder into the *smarter* database and unpack file contents into `data/interim` subfolder

INPUT_FILEPATH: The CSV dataset description file

```
src/data/import_datasets.py [OPTIONS] INPUT_FILEPATH
```

Options

--types <types>

Required 2 argument types (ex. genotypes background, phenotypes foreground, etc)

Arguments

INPUT_FILEPATH

Required argument

2.4.5 src/data/import_dbsnp.py

```
src/data/import_dbsnp.py [OPTIONS]
```

Options

--species_class <species_class>

Required The generic species of dbSNP data (Sheep or Goat)

--input_dir <input_dir>

Required The directory with dbSNP input (XML) files

--pattern <pattern>

The directory with dbSNP input (XML) files

Default

*.gz

--sender <sender>

Required The SNP sender (ex. AGR_BS, IGGC)

--version <version>

Required The assembly version

--imported_from <imported_from>

The source of this data

2.4.6 src/data/import_from_affymetrix.py

Read genotype data from affymetrix files and convert it to the desidered assembly version using Illumina TOP coding

```
src/data/import_from_affymetrix.py [OPTIONS]
```

Options

--prefix <prefix>
: File prefix for map and ped files (like plink does)

--report <report>
: Affymetrix report path

--dataset <dataset>
: **Required** The raw dataset file name (zip archive)

--coding <coding>
: Affymetrix coding format

  Default
  : affymetrix

  Options
  : ab | affymetrix

--breed_code <breed_code>
: A breed code to be assigned on all samples while creating samples

--chip_name <chip_name>
: **Required** The SMARTER SupportedChip name

--assembly <assembly>
: **Required** Destination assembly of the converted genotypes

--create_samples
: Create a new SampleSheep or SampleGoat object if doesn't exist

--sample_field <sample_field>
: Search samples using this attribute

--search_field <search_field>
: search variants using this field

  Default
  : probeset_id

--src_version <src_version>
: **Required** Source assembly version

--src_imported_from <src_imported_from>
: **Required** Source assembly imported_from

--max_samples <max_samples>
: Limit import to first samples (only valid for affymetrix report)

--skip_coordinate_check
: Skip coordinate check (only valid for affymetrix report)

2.4.7 src/data/import_from_illumina.py

Read genotype data from an Illumina report file and convert it to the desidered assembly version using Illumina TOP coding

```
src/data/import_from_illumina.py [OPTIONS]
```

Options

--dataset <dataset>

Required The raw dataset file name (zip archive)

--snpfile <snpfile>

Required The illumina SNPlist file

--report <report>

Required The illumina report file

--coding <coding>

Illumina coding format

Default
ab

Options
ab

--breed_code <breed_code>

Assign this FID to every sample in illumina report

--chip_name <chip_name>

Required The SMARTER SupportedChip name

--assembly <assembly>

Required Destination assembly of the converted genotypes

--create_samples

Create a new SampleSheep or SampleGoat object if doesn't exist

2.4.8 src/data/import_from_plink.py

Read genotype data from a PLINK file (text or binary) and convert it to the desidered assembly version using Illumina TOP coding

```
src/data/import_from_plink.py [OPTIONS]
```

Options

--file <file_>
PLINK text file prefix

--bfile <bfile>
PLINK binary file prefix

--dataset <dataset>
Required The raw dataset file name (zip archive)

--coding <coding>
Genotype coding format

Default
top

Options
top | forward | ab | affymetrix | illumina

--chip_name <chip_name>
Required The SMARTER SupportedChip name

--assembly <assembly>
Required Destination assembly of the converted genotypes

--create_samples
Create a new SampleSheep or SampleGoat object if doesn't exist

--sample_field <sample_field>
Search samples using this attribute

--search_field <search_field>
search variants using this field

Default
name

--search_by_positions
search variants using their positions

--src_version <src_version>
Source assembly version

--src_imported_from <src_imported_from>
Source assembly imported_from

--ignore_coding_errors
set SNP as missing when there are coding errors (no more CodingException)

2.4.9 src/data/import_iggc.py

Read data from Goat genome project and add a new location type for variants

```
src/data/import_iggc.py [OPTIONS]
```

Options

--datafile <datafile>

Required

--version <version>

Required

--force_update

Force location update

--date <date>

A date string

--entry_column <entry_column>

Entry name column in datafile (the SNP name)

Default

locus_name

--chrom_column <chrom_column>

Required Chromosome column in datafile

--pos_column <pos_column>

Required Position column in datafile

--strand_column <strand_column>

Required Strand column in datafile

--sequence_column <sequence_column>

Sequence column in datafile

Default

sequence

--rs_column <rs_column>

rsID column in datafile

Default

rs_

2.4.10 src/data/import_isgc.py

Read data from Sheep genome project and add a new location type for variants

```
src/data/import_isgc.py [OPTIONS]
```

Options

--datafile <datafile>

Required

--version <version>

Required

--force_update

Force location update

--date <date>

A date string

--entry_column <entry_column>

Entry name column in datafile (the SNP name)

Default
entry

--chrom_column <chrom_column>

Chromosome column in datafile

Default
chrom

--pos_column <pos_column>

Position column in datafile

Default
pos

--alleles_column <alleles_column>

Alleles column in datafile

Default
alleles

2.4.11 src/data/import_manifest.py

Load SNP data from Illumina manifest file into SMARTER-database

```
src/data/import_manifest.py [OPTIONS]
```

Options

--species_class <species_class>
  Required

--manifest <manifest>
  Required

--chip_name <chip_name>
  Required

--version <version>
  Required

--sender <sender>
  Required

2.4.12 src/data/import_metadata.py

Read data from metadata file and add it to SMARTER-database samples

```
src/data/import_metadata.py [OPTIONS]
```

Options

--src_dataset <src_dataset>
  Required The raw dataset file name (zip archive) in which search datafile

--dst_dataset <dst_dataset>
  The raw dataset file name (zip archive) in which add metadata(def. the 'src_dataset')

--datafile <datafile>
  Required

--sheet_name <sheet_name>
  pandas 'sheet_name' option

--breed_column <breed_column>
  The breed column

--id_column <id_column>
  The original_id column

--alias_column <alias_column>
  The alias column

--latitude_column <latitude_column>

--longitude_column <longitude_column>

--sex_column <sex_column>
  Sex column in src datafile

--notes_column <notes_column>
  The notes field in metadata

--metadata_column <metadata_column>
Metadata column to track. Could be specified multiple times

--species_column <species_column>
Species column in src datafile

--na_values <na_values>
pandas NA values

2.4.13 src/data/import_multiple_phenotypes.py

Read multiple data for the same sample from phenotype file and add it to SMARTER-database samples

```
src/data/import_multiple_phenotypes.py [OPTIONS]
```

Options

--src_dataset <src_dataset>
Required The raw dataset file name (zip archive) in which search datafile

--dst_dataset <dst_dataset>
The raw dataset file name (zip archive) in which add metadata(def. the 'src_dataset')

--datafile <datafile>
Required

--sheet_name <sheet_name>
pandas 'sheet_name' option

--breed_column <breed_column>
The breed column

--id_column <id_column>
The original_id column

--alias_column <alias_column>
An alias for original_id

--column <columns>
Required Column to track. Could be specified multiple times

--na_values <na_values>
pandas NA values

2.4.14 src/data/import_phenotypes.py

Read data from phenotype file and add it to SMARTER-database samples

```
src/data/import_phenotypes.py [OPTIONS]
```

Options

--src_dataset <src_dataset>
: **Required** The raw dataset file name (zip archive) in which search datafile

--dst_dataset <dst_dataset>
: The raw dataset file name (zip archive) in which add metadata(def. the 'src_dataset')

--datafile <datafile>
: **Required**

--sheet_name <sheet_name>
: pandas 'sheet_name' option

--breed_column <breed_column>
: The breed column

--id_column <id_column>
: The original_id column

--alias_column <alias_column>
: An alias for original_id

--purpose_column <purpose_column>

--chest_girth_column <chest_girth_column>

--height_column <height_column>

--length_column <length_column>

--additional_column <additional_column>
: Additional column to track. Could be specified multiple times

--na_values <na_values>
: pandas NA values

2.4.15 src/data/import_samples.py

Generate samples from a metadata file

```
src/data/import_samples.py [OPTIONS]
```

Options

--src_dataset <src_dataset>
: **Required** The raw dataset file name (zip archive) in which search datafile

--dst_dataset <dst_dataset>
: The raw dataset file name (zip archive) in which define samples(def. the 'src_dataset')

--datafile <datafile>
: **Required** The metadata file in which search for information

--code_column <code_column>
Code column in src datafile (ie FID)

--code_all <code_all>
Code applied to all items in datafile

--country_column <country_column>
Country column in src datafile

--country_all <country_all>
Country applied to all items in datafile

--species_column <species_column>
Species column in src datafile

--species_all <species_all>
Species applied to all items in datafile

--id_column <id_column>
Required The 'original_id' column to place in smarter database

--sex_column <sex_column>
Sex column in src datafile

--chip_name <chip_name>
Required The SMARTER SupportedChip name

--alias_column <alias_column>
An alias for original_id

--skip_missing_alias
Don't import samples with no alias

2.4.16 src/data/import_snpchimp.py

Import data from SNPchiMp dump tables

```
src/data/import_snpchimp.py [OPTIONS]
```

Options

--species_class <species_class>
Required

--snpchimp <snpchimp>
Required

--version <version>
Required

2.4.17 src/data/import_snpchips.py

Upload chips into *src.features.smarterdb.SupportedChip* objects

```
src/data/import_snpchips.py [OPTIONS]
```

Options

--chip_file <chip_file>

Required The chip description JSON file

2.4.18 src/data/merge_datasets.py

Search for processed genotype files for a certain species in `data/processed` folder and then call PLINK to join all genotypes in the same dataset

```
src/data/merge_datasets.py [OPTIONS]
```

Options

--species_class <species_class>

Required Search processed genotypes belonging to this species ('Sheep'or 'Goat')

--assembly <assembly>

Required Search processed genotypes belonging to this assembly

2.4.19 src/data/SNPconvert.py

Convert a PLINK/Illumina report file in a SMARTER-like ouput file, without inserting data in SMARTER-database. Useful to convert data relying on SMARTER-database for private datasets (data which cannot be included in SMARTER-database)

```
src/data/SNPconvert.py [OPTIONS]
```

Options

--file <file_>

PLINK text file prefix

--bfile <bfile>

PLINK binary file prefix

--report <report>

The illumina report file

--snpfile <snpfile>

The illumina SNPlist file

--coding <coding>
Illumina coding format

Default
top

Options
top | forward | ab

--assembly <assembly>
Required Destination assembly of the converted genotypes

--species <species>
Required The SMARTER assembly species (Goat or Sheep)

--results_dir <results_dir>
Required Where results will be saved

--chip_name <chip_name>
The SMARTER SupportedChip name

--search_field <search_field>
search variants using this field

Default
name

--search_by_positions
search variants using their positions

--src_version <src_version>
Source assembly version

--src_imported_from <src_imported_from>
Source assembly imported_from

--ignore_coding_errors
set SNP as missing when there are coding errors (no more CodingException)

2.4.20 src/data/update_db_status.py

Update SMARTER database statuses

```
src/data/update_db_status.py [OPTIONS]
```

2.5 Submodules

2.5.1 src.features.affymetrix

Created on Wed Jun 9 16:39:57 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

src.features.affymetrix.**read_Manifest**(*path: Path*, *delimiter: str = ','*) → namedtuple

Open an affymetrix manifest file and yields records as namedtuple. Add an additional column for manifacured date (when SNP is recorded in datafile)

Parameters

- **path** (*Path*) – The position of manifest file.
- **delimiter** (*str, optional*) – field delimiter. The default is ",".

Yields

record (*collections.namedtuple*) – A single SNP record from manifest.

src.features.affymetrix.**read_affymetrixRow**(*path: Path*, *delimiter='\t'*) → namedtuple

Open an affymetrix report file and yields namedtuple. Add two additional columns for the number of SNPs and samples in each returned record

Parameters

- **path** (*Path*) – The path of report file.
- **delimiter** (*str, optional*) – Fields delimiter. The default is " ".

Yields

record (*collections.namedtuple*) – A single record (a SNP over all samples + affymetrix information)

src.features.affymetrix.**search_manifactured_date**(*header: list*) → Optional[datetime]

Grep manifactured date from affymetrix header

Parameters

header (*list*) – affymetrix header section

Returns

a datetime object

Return type

datetime.datetime

src.features.affymetrix.**search_n_samples**(*header: list*) → int

Grep number of samples in affymetrix reportfile

Parameters

header (*list*) – affymetrix header section

Returns

the number of samples in file

Return type

int

src.features.affymetrix.**search_n_snps**(*header: list*) → int

Grep number of SNPs in affymetrix reportfile

Parameters

header (*list*) – affymetrix header section

Returns

the number of SNPs in file

Return type

int

2.5.2 src.data.common

Created on Fri Apr 30 15:13:32 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

Common stuff for smarter scripts

class src.data.common.**AssemblyConf**(*version*, *imported_from*)

Bases: tuple

imported_from

Alias for field number 1

version

Alias for field number 0

src.data.common.**deal_with_datasets**(*src_dataset: str*, *dst_dataset: str*, *datafile: str*) → [<class 'src.features.smarterdb.Dataset'>, <class 'src.features.smarterdb.Dataset'>, <class 'pathlib.Path'>]

Check source and destination dataset with its content

src.data.common.**deal_with_sex_and_alias**(*sex_column: str*, *alias_column: str*, *row: Series*)

Deal with sex and alias parameters

Parameters

- **sex_column** (*str*) – The sex column label.
- **alias_column** (*str*) – The alias column label.
- **row** (*Series*) – A row of metadata file.

Returns

- **sex** (*SEX*) – A SEX instance.
- **alias** (*str*) – The alias read from metadata table could be None.

src.data.common.**fetch_and_check_dataset**(*archive: str*, *contents: list[str]*) → [<class 'src.features.smarterdb.Dataset'>, list[pathlib.Path]]

Common operations on dataset: fetch a dataset by file (submitted archive), check that working dir exists and required file contents is in dataset. Test and get full path of required files

Parameters

- **archive** (*str*) – the dataset archive (file)
- **contents** (*list*) – a list of files which beed to be defined in dataset

Returns
a dataset instance list[Path]: a list of Path of required files

Return type
Dataset

src.data.common.**get_sample_species**(*species: str*) → Union[*SampleSheep*, *SampleGoat*]

Get a species name in input. It return the proper SampleSpecies class

Parameters
species (*str*) – the species name

Returns
a SampleSpecies class

Return type
Union[*SampleSheep*, *SampleGoat*]

src.data.common.**get_variant_species**(*species: str*) → Union[*VariantSheep*, *VariantGoat*]

Get a species name in input. It return the proper VariantSpecies class

Parameters
species (*str*) – the species name

Returns
a VariantSpecies class

Return type
Union[*VariantSheep*, *VariantGoat*]

src.data.common.**new_variant**(*variant: Union[VariantSheep, VariantGoat]*, *location: Location*)

src.data.common.**pandas_open**(*datapath: Path*, ***kwargs*) → DataFrame

Open an excel or csv file with pandas and returns a dataframe

Parameters

- **datapath** (*Path*) – the path of the file
- **kwargs** (*dict*) – additional pandas options

Returns
file content as a pandas dataframe

Return type
pd.DataFrame

src.data.common.**update_affymetrix_record**(*variant: Union[VariantSheep, VariantGoat]*, *record: Union[VariantSheep, VariantGoat]*) → [typing.Union[src.features.smarterdb.VariantSheep, src.features.smarterdb.VariantGoat], <class 'bool'>]

src.data.common.**update_chip_name**(*variant: Union[VariantSheep, VariantGoat]*, *record: Union[VariantSheep, VariantGoat]*) → [typing.Union[src.features.smarterdb.VariantSheep, src.features.smarterdb.VariantGoat], <class 'bool'>]

src.data.common.**update_location**(*location: Location*, *variant: Union[VariantSheep, VariantGoat]*, *force_update: bool = False*) → [typing.Union[src.features.smarterdb.VariantSheep, src.features.smarterdb.VariantGoat], <class 'bool'>]

Check provided Location with variant Locations: append new object or update a Location object if more recent than the data stored in database

Parameters

- **location** (*Location*) – The location to test against the database.
- **variant** (*Union[VariantSheep, VariantGoat]*) – The variant to test.
- **force_update** (*bool, optional*) – Force location update. The default is False.

Returns
A list with the updated VariantSpecie object and a boolean value which is True if the location was updated

Return type
[Union[*VariantSheep*, *VariantGoat*], bool]

src.data.common.**update_probesets**(*variant_attr: list[*src.features.smarterdb.Probeset*]*, *record_attr: list[*src.features.smarterdb.Probeset*]*) → bool

Update probeset relying on object references

src.data.common.**update_rs_id**(*variant: Union[*VariantSheep*,* VariantGoat*]*, *record: Union[*VariantSheep*,* VariantGoat*]*) → [typing.Union[src.features.smarterdb.VariantSheep, src.features.smarterdb.VariantGoat], <class 'bool'>]

src.data.common.**update_sequence**(*variant: Union[*VariantSheep*,* VariantGoat*]*, *record: Union[*VariantSheep*,* VariantGoat*]*) → [typing.Union[src.features.smarterdb.VariantSheep, src.features.smarterdb.VariantGoat], <class 'bool'>]

src.data.common.**update_variant**(*qs: QuerySet*, *variant: Union[*VariantSheep*,* VariantGoat*]*, *location:* Location) → bool

Update an existing variant (if necessary)

2.5.3 src.features.dbsnp

Created on Wed Mar 3 10:54:15 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

class src.features.dbsnp.**DBSNP**(*path*, *elem*)

Bases: object

__init__(*path*, *elem*)

ass_component(*elem*)

ass_maploc(*elem*)

ass_snpstat(*elem*)

classmethod clean_tag(*tag: str*)

config = {('ExchangeSet', 'Rs', 'Assembly'): 'record_assembly', ('ExchangeSet', 'Rs', 'Assembly', 'Component'): 'ass_component', ('ExchangeSet', 'Rs', 'Assembly', 'Component', 'MapLoc'): 'ass_maploc', ('ExchangeSet', 'Rs', 'Assembly', 'SnpStat'): 'ass_snpstat', ('ExchangeSet', 'Rs', 'Create'): 'record_create', ('ExchangeSet', 'Rs', 'Sequence'): 'record_exemplar', ('ExchangeSet', 'Rs', 'Sequence', 'Observed'): 'record_observed', ('ExchangeSet', 'Rs', 'Ss'): 'record_ss', ('ExchangeSet', 'Rs', 'Ss', 'Sequence', 'Observed'): 'ss_observed', ('ExchangeSet', 'Rs', 'Ss', 'Sequence', 'Seq3'): 'ss_seq3', ('ExchangeSet', 'Rs', 'Ss', 'Sequence', 'Seq5'): 'ss_seq5', ('ExchangeSet', 'Rs', 'Update'): 'record_update'}

process_element(*path*, *elem*)

record_assembly(*elem*)

record_create(*elem*)

record_exemplar(*elem*)

record_observed(*elem*)

record_ss(*elem*)

record_update(*elem*)

recurse_children(*path: list*, *elem: Element*)

ss_observed(*elem*)

ss_seq3(*elem*)

ss_seq5(*elem*)

to_dict()

src.features.dbsnp.**process_rs_elem**(*elem: Element*)

src.features.dbsnp.**read_dbSNP**(*path: str*)

src.features.dbsnp.**search_chip_snps**(*snp*, *handle='AGR_BS'*)

2.5.4 src.features.illumina

Created on Mon Feb 8 17:15:26 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

class src.features.illumina.**IlluSNP**(*sequence=None*, *max_iter=10*)

Bases: object

__init__(*sequence=None*, *max_iter=10*)

Define a IlluSNP class

findSNP(*sequence*)

Find snp (eg [A/G] in sequence (0-based)

fromSequence(*sequence*, *max_iter=10*)

Define a IlluSNP from a sequence

isUnambiguous(*snp*)

Return True if snp is unambiguous

toTop()

Convert a BOT sequence into TOP

exception src.features.illumina.**IlluSNPException**(*value=None*)

Bases: Exception

Base exception class for IlluSNP

__init__(*value=None*)

src.features.illumina.**read_Manifest**(*path: str*, *size=2048*, *skip=0*, *delimiter=None*)

src.features.illumina.**read_illuminaRow**(*path: str*, *size=2048*)

src.features.illumina.**read_snpList**(*path: str*, *size=2048*, *skip=0*, *delimiter=None*)

src.features.illumina.**read_snpMap**(*path: str*, *size=2048*, *skip=0*, *delimiter=None*)

src.features.illumina.**search_manifactured_date**(*header: list*) → Optional[datetime]

Grep manifactured date from illumina header

Parameters
header (*list*) – the illumina header skipped lines

Returns
a datetime object

Return type
datetime.datetime

src.features.illumina.**skip_lines**(*handle*, *skip*) → Tuple[int, list]

src.features.illumina.**skip_until_section**(*handle*, *section*) → Tuple[int, list]

Ignore lines until a precise sections

src.features.illumina.**sniff_file**(*handle*, *size*, *position=0*)

2.5.5 src.features.plinkio

Table of Contents

AffyPlinkIO

class src.features.plinkio.**AffyPlinkIO**(*prefix: Optional[str] = None*, *mapfile: Optional[str] = None*, *pedfile: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

Bases: *FakePedMixin*, *TextPlinkIO*

a new class for affymetrix plink files, which are slightly different from plink text files

get_samples() → list

Get samples from genotype files

Returns
The sample list.

Return type
list

read_pedfile(*breed: Optional[str] = None*, *dataset: Optional[Dataset] = None*, **args*, ***kwargs*)

Open pedfile for reading return iterator

breed
[str, optional] A breed to be assigned to all samples, or use the sample breed stored in database if not provided. The default is None.

dataset
[Dataset, optional] A dataset in which search for sample breed identifier

Yields
line (*list*) – A ped line read as a list.

AffyReportIO

class src.features.plinkio.**AffyReportIO**(*report: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

Bases: *FakePedMixin*, *SmarterMixin*

In this type of file there are both genotypes and informations. Moreover genotypes are *transposed*, traking SNP for all samples in a simple line

__init__(*report: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

delimiter = '\t'

fetch_coordinates(*src_assembly: AssemblyConf*, *dst_assembly: Optional[AssemblyConf] = None*, *search_field: str = 'name'*, *chip_name: Optional[str] = None*, *skip_check: bool = False*)

Search for variants in smarter database. Check if the provided A/B information is equal to the database content

Parameters

- **src_assembly** (*AssemblyConf*) – the source data assembly version
- **dst_assembly** (*AssemblyConf*) – the destination data assembly version
- **search_field** (*str*) – search variant by field (def. “name”)
- **chip_name** (*str*) – limit search to this chip_name
- **skip_check** (*bool*) – skipp coordinate check

get_samples() → list

Get samples from genotype files

Returns
The sample list.

Return type
list

header = []

n_samples = None

peddata = []

read_peddata(*breed: Optional[str] = None*, *dataset: Optional[Dataset] = None*, *sample_field: str = 'original_id'*, **args*, ***kwargs*)

Yields over genotype record.

Parameters

- **breed** (*str, optional*) – A breed to be assigned to all samples, or use the sample breed stored in database if not provided. The default is None.
- **dataset** (*Dataset, optional*) – A dataset in which search for sample breed identifier
- **sample_field** (*str, optional*) – Search samples using this field. The default is "original_id".

Yields

line (*list*) – A ped line read as a list.

read_reportfile(*n_samples: Optional[int] = None*, **args*, ***kwargs*)

Read reportfile once and generate mapdata and pedata, with genotype informations by sample.

Parameters

n_samples (*int, optional*) – Limit to N samples. Useful when there are different number of samples from reported in file. The default is None (read number of samples from reportfile).

Return type

None.

report = None

warn_missing_cols = True

BinaryPlinkIO

class src.features.plinkio.**BinaryPlinkIO**(*prefix: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

Bases: *SmarterMixin*

__init__(*prefix: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

get_samples() → list

Get samples from genotype files

Returns

The sample list.

Return type

list

plink_file = None

property prefix

read_mapfile()

Read map data and track informations in memory. Useful to process data files

read_pedfile(**args*, ***kwargs*)

Open pedfile for reading return iterator

MapRecord

class src.features.plinkio.**MapRecord**(*chrom: str*, *name: str*, *cm: float*, *position: int*)

Bases: object

__init__(*chrom: str*, *name: str*, *cm: float*, *position: int*) → None

chrom: str

cm: float

name: str

position: int

FakePedMixin

class src.features.plinkio.**FakePedMixin**

Bases: object

Class which override SmarterMixin when creating a PED file from a non-plink file format. In this case the FID is already correct and I don't need to look for dataset aliases

search_breed(*fid*, **args*, ***kwargs*)

Get breed relying on provided FID and species class attribute

search_fid(*sample_name: str*, *dataset:* Dataset, *sample_field: str = 'original_id'*) → str

Determine FID from smarter SampleSpecies breed

Parameters

- **sample_name** (*str*) – The sample name.
- **dataset** (Dataset) – The dataset where the sample comes from.
- **sample_field** (*str*) – The field use to search sample name

Returns
fid – The FID used in the generated .ped file

Return type
str

IlluminaReportIO

class src.features.plinkio.**IlluminaReportIO**(*snpfile: Optional[str] = None*, *report: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

Bases: *FakePedMixin*, *SmarterMixin*

__init__(*snpfile: Optional[str] = None*, *report: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

get_samples() → list

Get samples from genotype files

Returns
The sample list.

Return type
list

read_reportfile(*breed: Optional[str] = None*, *dataset: Optional[*Dataset*] = None*, **args*, ***kwargs*)

Open and read an illumina report file. Returns iterator

Parameters

- **breed** (*str, optional*) – A breed to be assigned to all samples, or use the sample breed stored in database if not provided. The default is None.
- **dataset** (Dataset, *optional*) – A dataset in which search for sample breed identifier

Raises
IlluminaReportException – Raised when SNPs index doesn't match snpfile.

Yields
line (*list*) – A ped line read as a list.

read_snpfile()

Read snp data and track informations in memory. Useful to process data files

report = None

snpfile = None

SmarterMixin

class src.features.plinkio.**SmarterMixin**

Bases: object

Common features of a Smarter related dataset file

SampleSpecies = None

VariantSpecies = None

chip_name = None

dst_locations = []

fetch_coordinates(*src_assembly: AssemblyConf*, *dst_assembly: Optional[AssemblyConf] = None*, *search_field: str = 'name'*, *chip_name: Optional[str] = None*, **args*, ***kwargs*)

Search for variants in smarter database

Parameters

- **src_assembly** (AssemblyConf) – the source data assembly version
- **dst_assembly** (AssemblyConf) – the destination data assembly version
- **search_field** (*str*) – search variant by field (def. "name")
- **chip_name** (*str*) – limit search to this chip_name

fetch_coordinates_by_positions(*src_assembly: AssemblyConf*, *dst_assembly: Optional[AssemblyConf] = None*)

Search for variant in smarter database relying on positions

Parameters

- **src_assembly** (AssemblyConf) – the source data assembly version.

- **dst_assembly** (AssemblyConf, *optional*) – the destination data assembly version. The default is None.

Return type
None.

filtered = {}

get_or_create_sample(*line: list*, *dataset:* Dataset, *breed:* Breed, *sample_field: str = 'original_id'*, *create_sample: bool = False*) → Union[*SampleSheep*, *SampleGoat*]

Get a sample from database or create a new one (if create_sample parameter flag is provided)

Parameters

- **line** (*list*) – A ped line as a list.
- **dataset** (Dataset) – The dataset object this sample belongs to.
- **breed** (Breed) – The Breed object of such sample.
- **sample_field** (*str, optional*) – Search sample name within this field. The default is "original_id".
- **create_sample** (*bool, optional*) – Create a sample if not found in database. The default is False.

Raises
SmarterDBException – Raised if more than one sample is retrieved.

Returns
sample – A SampleSheep or SampleGoat object for a Sample object found or created. None if no sample is found and create_sample if False.

Return type
Union[*SampleSheep*, *SampleGoat*]

make_query_args(*src_assembly: AssemblyConf*, *dst_assembly: AssemblyConf*)

Generate args to select variants from database

make_query_kwargs(*search_field: str*, *record:* MapRecord, *chip_name: str*)

Generate kwargs to select variants from database

mapdata = []

read_genotype_method = None

search_breed(*fid*, *dataset*, **args*, ***kwargs*)

Get breed relying aliases and dataset

search_country(*dataset:* Dataset, *breed:* Breed)

skip_index(*idx*)

Skip a certain SNP reling on its position

property species

src_locations = []

update_mapfile(*outputfile: str*)

update_pedfile(*outputfile: str*, *dataset: Dataset*, *coding: str*, *create_samples: bool = False*, *sample_field: str = 'original_id'*, *ignore_coding_errors: bool = False*, **args*, ***kwargs*)

Write a new pedfile relying on illumina_top genotypes and coordinates stored in smarter database

Parameters

- **outputfile** (*str*) – write ped to this path (overwrite if exists)
- **dataset** (*Dataset*) – the dataset we are converting
- **coding** (*str*) – the source coding (could be 'top', 'ab', 'forward')
- **create_samples** (*bool*) – create samples if not exist (useful to create samples directly from ped file)
- **sample_field** (*str*) – search samples using this attribute (def. 'original_id')
- **ignore_coding_errors** (*bool*) – ignore coding related errors (no more exceptions when genotypes don't match)

variants_name = []

TextPlinkIO

class src.features.plinkio.**TextPlinkIO**(*prefix: Optional[str] = None*, *mapfile: Optional[str] = None*, *pedfile: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

Bases: *SmarterMixin*

__init__(*prefix: Optional[str] = None*, *mapfile: Optional[str] = None*, *pedfile: Optional[str] = None*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*)

get_samples() → list

Get samples from genotype files

Returns
The sample list.

Return type
list

mapfile = None

pedfile = None

read_mapfile()

Read map data and track informations in memory. Useful to process data files

read_pedfile(**args*, ***kwargs*)

Open pedfile for reading return iterator

plink_binary_exists

plinkio.**plink_binary_exists**()

Test if plink binary files exists

2.5.6 src.features.smarterdb

Created on Tue Feb 23 16:21:35 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

Classes:

Breed(*args, **values)	

BreedAlias(*args, **kwargs)	Required to describe the breed and code used in a certain dataset in order to resolve the final breed to be used in SMARTER-database
Consequence(*args, **kwargs)	A class to manage SNP consequences.
Counter(*args, **values)	A class to deal with counter collection (created when initializing smarter database) and used to define SMARTER IDs
Country([name])	A helper class to deal with countries object.
Dataset(*args, **values)	Describe a dataset instace with fields owned by data types
Location(*args, **kwargs)	A class to deal with a SNP location (ie position in an assembly for a certain chip or data source)
Phenotype(*args, **kwargs)	A class to deal with phenotypes.
Probeset(*args, **kwargs)	A class to deal with different affymetrix probesets
SAMPLETYPE(value)	A simple Enum object to define sample type (`background` or `foreground`)
SEX(value)	An enum object to manage Sample sex in the same way as plink does
SampleGoat(*args, **values)	A class specific for Goat samples
SampleSheep(*args, **values)	A class specific for Sheep samples
SampleSpecies(*args, **values)	A generic class used to manage Goat or Sheep samples
SmarterInfo(*args, **values)	A class to track database status informations
SupportedChip(*args, **values)	A class to deal with SMARTER-database managed chips
VariantGoat(*args, **values)	A class to deal with Goat variations (SNP)
VariantSheep(*args, **values)	A class to deal with Sheep variations (SNP)
VariantSpecies(*args, **values)	Generic class to deal with Variant (SNP) objects

Exceptions:

SmarterDBException

Functions:

complement(genotype)	Return reverse complement for a base call
getNextSequenceValue(sequence_name, mongodb)	Read from *Counter* collection and determine the next sequence number to be used for the SMARTER ID
getSmarterId(species_class, country, breed)	Generate a new SMARTER ID object using the internal counter collections
get_or_create_breed(species_class, name, code)	Get a Breed instance or create a new one (or update a breed adding a new *BreedAlias*)
get_or_create_sample(SampleSpecies, ...[, ...])	Get or create a sample providing attributes (search for original_id in provided dataset
get_sample_type(dataset)	test if foreground or background dataset
global_connection([database_name])	Establish a connection to the SMARTER database.

class src.features.smarterdb.**Breed**(**args*, ***values*)

Bases: Document

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

aliases	A list of *BreedAlias* objects.
code	The breed code
id	A field wrapper around MongoDB's ObjectIds.
n_individuals	How many samples are the same breed
name	The breed name
objects([q_obj])	
species	The breed species.

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

aliases

A list of *BreedAlias* objects. Required to determine the SMARTER-database breed from the genotype file (which can use a different breed name or code)

code

The breed code

id

A field wrapper around MongoDB's ObjectIds.

n_individuals

How many samples are the same breed

name

The breed name

objects(*q_obj=None*, ***query*) = []

species

The breed species. Should be one of Goat or Sheep

class src.features.smarterdb.**BreedAlias**(**args*, ***kwargs*)

Bases: EmbeddedDocument

Required to describe the breed and code used in a certain dataset in order to resolve the final breed to be used in SMARTER-database

Attributes:

country	The country of the breed in the dataset.
dataset	The dataset ObjectID in which this BreedAlias is used
fid	The breed Family ID used in genotype file

country

The country of the breed in the dataset. Used in multi country datasets

dataset

The dataset ObjectID in which this BreedAlias is used

fid

The breed Family ID used in genotype file

class src.features.smarterdb.**Consequence**(**args*, ***kwargs*)

Bases: EmbeddedDocument

A class to manage SNP consequences. Not yet implemented

class src.features.smarterdb.**Counter**(**args*, ***values*)

Bases: Document

A class to deal with counter collection (created when initializing smarter database) and used to define SMARTER IDs

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

id	A unicode string field.
objects([q_obj])	
sequence_value	32-bit integer field.

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A unicode string field.

objects(*q_obj=None*, ***query*) = []

sequence_value

32-bit integer field.

class src.features.smarterdb.**Country**(*name: Optional[str] = None*, **args*, ***kwargs*)

Bases: Document

A helper class to deal with countries object. Each record is created after data import, when database status is updated

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Methods:

__init__([name])	Initialise a document or an embedded document.

Attributes:

alpha_2	Country 2 letter code (used to derive SMARTER IDs)
alpha_3	Country 3 letter code
id	A field wrapper around MongoDB's ObjectIds.
name	The Country name
numeric	The country numeric code
objects([q_obj])	
official_name	Country ufficial name
species	The sample species find within this country

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

__init__(*name: Optional[str] = None*, **args*, ***kwargs*)

Initialise a document or an embedded document.

Parameters

- **values** – A dictionary of keys and values for the document. It may contain additional reserved keywords, e.g. "__auto_convert".

- **__auto_convert** – If True, supplied values will be converted to Python-type values via each field's *to_python* method.
- **_created** – Indicates whether this is a brand new document or whether it's already been persisted before. Defaults to true.

alpha_2

Country 2 letter code (used to derive SMARTER IDs)

alpha_3

Country 3 letter code

id

A field wrapper around MongoDB's ObjectIds.

name

The Country name

numeric

The country numeric code

objects(*q_obj=None*, ***query*) = []

official_name

Country official name

species

The sample species find within this country

class src.features.smarterdb.**Dataset**(**args*, ***values*)

Bases: Document

Describe a dataset instace with fields owned by data types

Miscellaneous:

DoesNotExist

MultipleObjectsReturned

Attributes:

breed	The breed of the dataset.
chip_name	The *SupportedChip.name* attribute of the technology used
contents	Dataset contents as a list
country	The country where the data come from.
doi	The publication DOI of this dataset
file	The source dataset file
gene_array	The technology used to generate data specified by the partner
id	A field wrapper around MongoDB's ObjectIds.
n_of_individuals	Number of individual in the dataset
n_of_records	Number of the record in the phenotype file
objects([q_obj])	

partner	The partner which owns the dataset
result_dir	returns the locations of dataset processed directory.
*size_*	The file size
species	The species of the data.
trait	Trait described in phenotype file
*type_*	Dataset type.
uploader	The partner which upload this dataset
working_dir	returns the locations of dataset working directory.

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

breed

The breed of the dataset. Could have many values

chip_name

The *SupportedChip.name* attribute of the technology used

contents

Dataset contents as a list

country

The country where the data come from. Could have many values

doi

The publication DOI of this dataset

file

The source dataset file

gene_array

The technology used to generate data specified by the partner

id

A field wrapper around MongoDB's ObjectIds.

n_of_individuals

Number of individual in the dataset

n_of_records

Number of the record in the phenotype file

objects(*q_obj=None*, ***query*) = []

partner

The partner which owns the dataset

property result_dir: PosixPath

returns the locations of dataset processed directory. Could exists or not

Returns
a subdirectory in /data/processed/

Return type
pathlib.PosixPath

size_

The file size

species

The species of the data. Could be 'Sheep' or 'Goat'

trait

Trait described in phenotype file

type_

Dataset type. Need to be one from `['genotypes', 'phenotypes]` and one from `['background', 'foreground']`

uploader

The partner which upload this dataset

property working_dir: PosixPath

returns the locations of dataset working directory. Could exists or not

Returns
a subdirectory in /data/interim/

Return type
pathlib.PosixPath

class src.features.smarterdb.**Location**(**args*, ***kwargs*)

Bases: EmbeddedDocument

A class to deal with a SNP location (ie position in an assembly for a certain chip or data source)

Methods:

__init__(*args, **kwargs)	Initialise a document or an embedded document.
ab2top(genotype[, missing])	Convert an illumina ab SNP in a illumina top snp
affy2top(genotype[, missing])	Convert an affymetrix SNP in a illumina top snp
forward2top(genotype[, missing])	Convert an illumina forward SNP in a illumina top snp
illumina2top(genotype[, missing])	Convert an illumina SNP in a illumina top snp
is_ab(genotype[, missing])	Return True if genotype is compatible with illumina AB coding
is_affymetrix(genotype[, missing])	Return True if genotype is compatible with affymetrix coding
is_forward(genotype[, missing])	Return True if genotype is compatible with illumina FORWARD coding
is_illumina(genotype[, missing])	Return True if genotype is compatible with illumina coding (as it's recorded in manifest)
is_top(genotype[, missing])	Return True if genotype is compatible with illumina TOP coding

Attributes:

affymetrix_ab	The SNP code read as it is from affymetrix data
alleles	The dbSNP alleles of such SNP
chrom	The chromosome where this SNP is located
consequences	A list of SNP consequences (not yet implemented)
date	Track manifactured date or when this data was last updated
illumina	The SNP code read as it is from illumina data
illumina_forward	The SNP code in illumina forward coding
illumina_strand	The probe orientation in alignment
illumina_top	Return genotype in illumina top format
imported_from	The source of the SNP data
position	The SNP position
ss_id	The SNP subission ID
strand	The strand orientation in aligment
version	The assembly version where this SNP is placed

__init__(**args*, ***kwargs*)

Initialise a document or an embedded document.

Parameters

- **values** – A dictionary of keys and values for the document. It may contain additional reserved keywords, e.g. "__auto_convert".
- **__auto_convert** – If True, supplied values will be converted to Python-type values via each field's *to_python* method.
- **_created** – Indicates whether this is a brand new document or whether it's already been persisted before. Defaults to true.

ab2top(*genotype: list*, *missing: list = ['0', '-']*) → list

Convert an illumina ab SNP in a illumina top snp

Parameters

- **genotype** (*list*) – a list of two alleles (ex ['A','B'])

- **missing** (*list*) – a list of missing allele strings (def [“0”, “-“])

Returns
The genotype in top format

Return type
list

affy2top(*genotype: list*, *missing: list = ['0', '-']*) → list

Convert an affymetrix SNP in a illumina top snp

Parameters

- **genotype** (*list*) – a list of two alleles (ex [‘A’,’C’])
- **missing** (*list*) – a list of missing allele strings (def [“0”, “-“])

Returns
The genotype in top format

Return type
list

affymetrix_ab

The SNP code read as it is from affymetrix data

alleles

The dbSNP alleles of such SNP

chrom

The chromosome where this SNP is located

consequences

A list of SNP consequences (not yet implemented)

date

Track manifactured date or when this data was last updated

forward2top(*genotype: list*, *missing: list = ['0', '-']*) → list

Convert an illumina forward SNP in a illumina top snp

Parameters

- **genotype** (*list*) – a list of two alleles (ex [‘A’,’C’])
- **missing** (*list*) – a list of missing allele strings (def [“0”, “-“])

Returns
The genotype in top format

Return type
list

illumina

The SNP code read as it is from illumina data

illumina2top(*genotype: list*, *missing: list = ['0', '-']*) → list

Convert an illumina SNP in a illumina top snp

Parameters

- **genotype** (*list*) – a list of two alleles (ex [‘A’,’C’])
- **missing** (*list*) – a list of missing allele strings (def [“0”, “-“])

Returns
The genotype in top format

Return type
list

`illumina_forward`
The SNP code in illumina forward coding

`illumina_strand`
The probe orientation in alignment

`property illumina_top`
Return genotype in illumina top format

`imported_from`
The source of the SNP data

`is_ab`(*genotype: list*, *missing: list = ['0', '-']*) → bool
Return True if genotype is compatible with illumina AB coding

Parameters
- **genotype** (*list*) – a list of two alleles (ex ['A','B'])
- **missing** (*list*) – a list of missing allele strings (def ["0", "-"])

Returns
True if in AB coding

Return type
bool

`is_affymetrix`(*genotype: list*, *missing: list = ['0', '-']*) → bool
Return True if genotype is compatible with affymetrix coding

Parameters
- **genotype** (*list*) – a list of two alleles (ex ['A','C'])
- **missing** (*list*) – a list of missing allele strings (def ["0", "-"])

Returns
True if in affymetrix AB coding

Return type
bool

`is_forward`(*genotype: list*, *missing: list = ['0', '-']*) → bool
Return True if genotype is compatible with illumina FORWARD coding

Parameters
- **genotype** (*list*) – a list of two alleles (ex ['A','C'])
- **missing** (*list*) – a list of missing allele strings (def ["0", "-"])

Returns
True if in forward coding

Return type
bool

is_illumina(*genotype: list*, *missing: list = ['0', '-']*) → bool

Return True if genotype is compatible with illumina coding (as it's recorded in manifest)

Parameters

- **genotype** (*list*) – a list of two alleles (ex ['A','C'])
- **missing** (*list*) – a list of missing allele strings (def ["0", "-"])

Returns
True if in affymetrix AB coding

Return type
bool

is_top(*genotype: list*, *missing: list = ['0', '-']*) → bool

Return True if genotype is compatible with illumina TOP coding

Parameters

- **genotype** (*list*) – a list of two alleles (ex ['A','C'])
- **missing** (*list*) – a list of missing allele strings (def ["0", "-"])

Returns
True if in top coding

Return type
bool

position

The SNP position

ss_id

The SNP subission ID

strand

The strand orientation in aligment

version

The assembly version where this SNP is placed

class src.features.smarterdb.**Phenotype**(**args*, ***kwargs*)

Bases: DynamicEmbeddedDocument

A class to deal with phenotypes. This is a dynamic document and not a generic DictField since there can be attributes which could be enforced to have certain values. All other attributes could be set without any assumptions

Attributes:

chest_girth	Floating point number field.
height	Floating point number field.
length	Floating point number field.
purpose	A unicode string field.

chest_girth

Floating point number field.

height

Floating point number field.

length

Floating point number field.

purpose

A unicode string field.

class src.features.smarterdb.**Probeset**(**args*, ***kwargs*)

Bases: EmbeddedDocument

A class to deal with different affymetrix probesets

Attributes:

chip_name	the chip name where this affymetrix probeset comes from
probeset_id	A list probeset assigned to the same SNP

chip_name

the chip name where this affymetrix probeset comes from

probeset_id

A list probeset assigned to the same SNP

class src.features.smarterdb.**SAMPLETYPE**(*value*)

Bases: Enum

A simple Enum object to define sample type (background or foreground)

Attributes:

BACKGROUND
FOREGROUND

BACKGROUND = 'background'

FOREGROUND = 'foreground'

class src.features.smarterdb.**SEX**(*value*)

Bases: bytes, Enum

An enum object to manage Sample sex in the same way as plink does

Attributes:

FEMALE
MALE
UNKNOWN

Methods:

from_string(value)	Get proper type relying on input string

FEMALE = 2

MALE = 1

UNKNOWN = 0

classmethod from_string(*value: str*)

Get proper type relying on input string

Parameters
value (*str*) – required sex as string

Returns
A sex instance (MALE, FEMALE, UNKNOWN)

Return type
SEX

class src.features.smarterdb.**SampleGoat**(**args*, ***values*)

Bases: *SampleSpecies*

A class specific for Goat samples

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

father_id	The father (SIRE) of this animal.
id	A field wrapper around MongoDB's ObjectIds.
mother_id	The mother (DAM) of this animal.
objects([q_obj])	
species	The species name.
species_class	The generic specie class

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

father_id

The father (SIRE) of this animal. Is a reference to another SampleGoat instance

id

A field wrapper around MongoDB's ObjectIds.

mother_id

The mother (DAM) of this animal. Is a reference to another SampleGoat instance

objects(*q_obj=None*, ***query*) = []

species

The species name. Could be something different from `Capra hircus`

species_class = 'Goat'

The generic specie class

class src.features.smarterdb.**SampleSheep**(**args*, ***values*)

Bases: *SampleSpecies*

A class specific for Sheep samples

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

father_id	The father (SIRE) of this animal.
id	A field wrapper around MongoDB's ObjectIds.
mother_id	The mother (DAM) of this animal.
objects([q_obj])	
species	The species name.
species_class	The generic specie class

exception DoesNotExist

Bases: `DoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

father_id

The father (SIRE) of this animal. Is a reference to another SampleSheep instance

id

A field wrapper around MongoDB's ObjectIds.

mother_id

The mother (DAM) of this animal. Is a reference to another SampleSheep instance

objects(*q_obj=None*, ***query*) = []

species

The species name. Could be something different from `Ovis aries`

species_class = 'Sheep'

The generic specie class

class src.features.smarterdb.**SampleSpecies**(**args*, ***values*)

Bases: Document

A generic class used to manage Goat or Sheep samples

Attributes:

alias	This is a sample alias, mainly the name used in the genotype file, which can be different from the name specified in the metadata file
breed	The breed full name
breed_code	The breed code
chip_name	The chip name used to define this sample
country	Where this samples comes from
dataset	The dataset where this sample come from
locations	The sample GPS location as a Point (X, Y -> longitude, latitude).
metadata	Additional metadata (not managed via ORM)
original_id	The sample original ID in the source dataset
phenotype	A *Phenotype* instance
sex	A *SEX* instance.
smarter_id	A SMARTER unique and stable identifier
species_class	A generic species (Sheep or Goat).
*type_*	A *SAMPLETYPE* instance (ie, `background` or `foreground`

Methods:

save(*args, **kwargs)	Custom save method.

alias

This is a sample alias, mainly the name used in the genotype file, which can be different from the name specified in the metadata file

breed

The breed full name

breed_code

The breed code

chip_name

The chip name used to define this sample

country

Where this samples comes from

dataset

The dataset where this sample come from

locations

The sample GPS location as a Point (X, Y -> longitude, latitude). Mind that a location is specified in latitude and longitude coordinates. Specifying coordinates header in general is useful to avoid errors

metadata

Additional metadata (not managed via ORM)

original_id

The sample original ID in the source dataset

phenotype

A *Phenotype* instance

save(**args*, ***kwargs*)

Custom save method. Deal with smarter_id before save

sex

A *SEX* instance. Store sex like plink does

smarter_id

A SMARTER unique and stable identifier

species_class = None

A generic species (Sheep or Goat). Used to determine specific methods and to identify the proper data from the database

type_

A *SAMPLETYPE* instance (ie, background or foreground

exception src.features.smarterdb.**SmarterDBException**

Bases: Exception

class src.features.smarterdb.**SmarterInfo**(**args*, ***values*)

Bases: Document

A class to track database status informations

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

id	A unicode string field.
last_updated	When the SMARTER-database was updated for the last time
objects([q_obj])	
plink_specie_opt	The plink parameters used to generate the final genotype dataset
version	The SMARTER-database version
working_assemblies	A dictionary in which managed assemblies are tracked

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A unicode string field.

last_updated

When the SMARTER-database was updated for the last time

objects(*q_obj=None*, ***query*) = []

plink_specie_opt

The plink parameters used to generate the final genotype dataset

version

The SMARTER-database version

working_assemblies

A dictionary in which managed assemblies are tracked

class src.features.smarterdb.**SupportedChip**(**args*, ***values*)

Bases: Document

A class to deal with SMARTER-database managed chips

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

id	A field wrapper around MongoDB's ObjectIds.
manifacturer	Who created the chip
n_of_snps	How many SNPs are described within this chip
name	The chip identifier
objects([q_obj])	
species	The species for which a chip is defined

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A field wrapper around MongoDB's ObjectIds.

manifacturer

Who created the chip

n_of_snps

How many SNPs are described within this chip

name

The chip identifier

objects(*q_obj=None*, ***query*) = []

species

The species for which a chip is defined

class src.features.smarterdb.**VariantGoat**(**args*, ***values*)

Bases: *VariantSpecies*

A class to deal with Goat variations (SNP)

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

id	A field wrapper around MongoDB's ObjectIds.
objects([q_obj])	

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A field wrapper around MongoDB's ObjectIds.

objects(*q_obj=None*, ***query*) = []

class src.features.smarterdb.**VariantSheep**(**args*, ***values*)

Bases: *VariantSpecies*

A class to deal with Sheep variations (SNP)

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Attributes:

id	A field wrapper around MongoDB's ObjectIds.
objects([q_obj])	

exception DoesNotExist

Bases: DoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A field wrapper around MongoDB's ObjectIds.

objects(*q_obj=None*, ***query*) = []

class src.features.smarterdb.**VariantSpecies**(**args*, ***values*)

Bases: Document

Generic class to deal with Variant (SNP) objects

Attributes:

affy_snp_id	The affymetrix SNP id
chip_name	The chip names where this SNP could be found
cust_id	The affymetrix customer id (which is the illumina name)
illumina_top	Illumina TOP variant (which is the same indipendently by locations)
locations	A list of *Location* objects
name	The name of the SNPs.
probesets	A list of *Probeset* objects
rs_id	The SNP rsID
sender	Who provide this SNP probe
sequence	A dictionary where keys are chip_name, and values are their probe sequences

Methods:

get_location(version[, imported_from])	Returns location for assembly version and imported source
get_location_index(version[, imported_from])	Returns location index for assembly version and imported source
save(*args, **kwargs)	Custom save method.

affy_snp_id

The affymetrix SNP id

chip_name

The chip names where this SNP could be found

cust_id

The affymetrix customer id (which is the illumina name)

get_location(*version: str*, *imported_from='SNPchiMp v.3'*)

Returns location for assembly version and imported source

Parameters

- **version** (*str*) – assembly version (ex: 'Oar_v3.1')
- **imported_from** (*str*) – coordinates source (ex: 'SNPchiMp v.3')

Returns

the genomic coordinates

Return type
Location

`get_location_index`(*version: str*, *imported_from='SNPchiMp v.3'*)

Returns location index for assembly version and imported source

Parameters

- **version** (*str*) – assembly version (ex: 'Oar_v3.1')
- **imported_from** (*str*) – coordinates source (ex: 'SNPchiMp v.3')

Returns
the index of the location requested

Return type
int

`illumina_top`

Illumina TOP variant (which is the same indipendently by locations)

`locations`

A list of *Location* objects

`name`

The name of the SNPs. Could be illumina name or affyemtrix name

`probesets`

A list of *Probeset* objects

`rs_id`

The SNP rsID

`save`(**args*, ***kwargs*)

Custom save method. Deal with variant name before save

`sender`

Who provide this SNP probe

`sequence`

A dictionary where keys are chip_name, and values are their probe sequences

src.features.smarterdb.**complement**(*genotype: str*) → str

Return reverse complement for a base call

Parameters
genotype (*str*) – A base call (one from *A*, *T*, *G*, *C*).

Returns
result – The reverse complement of the base call.

Return type
str

src.features.smarterdb.**getNextSequenceValue**(*sequence_name: str*, *mongodb: Database*)

Read from *Counter* collection and determine the next sequence number to be used for the SMARTER ID

src.features.smarterdb.**getSmarterId**(*species_class: str*, *country: str*, *breed: str*) → str

Generate a new SMARTER ID object using the internal counter collections

Parameters

- **species_class** (*str*) – The class of the species (should be 'Goat' or 'Sheep').
- **country** (*str*) – The country name of the sample.
- **breed** (*str*) – The breed name of the sample.

Raises
SmarterDBException – Raised when passing a wrong species or no one.

Returns
A new smarter_id.

Return type
str

src.features.smarterdb.**get_or_create_breed**(*species_class: str*, *name: str*, *code: str*, *aliases: list = []*) → [<class 'src.features.smarterdb.Breed'>, <class 'bool'>]

Get a Breed instance or create a new one (or update a breed adding a new *BreedAlias*)

Parameters

- **species_class** (*str*) – The class of the species (should be 'Goat' or 'Sheep')
- **name** (*str*) – The breed full name.
- **code** (*str*) – The breed code (unique in Sheep and Goats collections).
- **aliases** (*list, optional*) – A list of *BreedAlias* objects. The default is [].

Raises
SmarterDBException – Raised if the breed is not Unique.

Returns

- **breed** (*Breed*) – A *Breed* instance.
- **modified** (*bool*) – True is breed is created (or alias updated).

src.features.smarterdb.**get_or_create_sample**(*SampleSpecies: Union[*SampleGoat, SampleSheep*]*, *original_id: str*, *dataset:* Dataset, *type_: str*, *breed:* Breed, *country: str*, *species: Optional[str] = None*, *chip_name: Optional[str] = None*, *sex: Optional[*SEX*] = None*, *alias: Optional[str] = None*) → list[Union[*src.features.smarterdb.SampleGoat*, *src.features.smarterdb.SampleSheep*], bool]

Get or create a sample providing attributes (search for original_id in provided dataset

Parameters

- **SampleSpecies** (*Union[SampleGoat, SampleSheep]*) – the class required for insert/update.
- **original_id** (*str*) – the original_id in the dataset.
- **dataset** (Dataset) – the dataset instance used to register sample.
- **type** (*str*) – sample type. "background" or "foreground" are the only values accepted
- **breed** (Breed) – a *Breed* instance.
- **country** (*str*) – the country where the sample comes from.
- **species** (*str, optional*) – The sample species. If None, the default *species_class* attribute will be used
- **chip_name** (*str, optional*) – the chip name. The default is None.

- **sex** (*SEX, optional*) – A *SEX* instance. The default is None.
- **alias** (*str, optional*) – an original_id alias. Could be the name used in the genotype file, which could be different from the original_id. The default is None.

Raises
SmarterDBException – Raised multiple samples are returned (should never happen).

Returns

- *Union[SampleGoat, SampleSheep]* – a SampleSpecies instance.
- **created** (*bool*) – True is sample is created.

src.features.smarterdb.**get_sample_type**(*dataset:* Dataset)

test if foreground or background dataset

Parameters
dataset (Dataset) – the dataset instance used to register sample

Returns
sample type ("background" or "foreground")

Return type
str

src.features.smarterdb.**global_connection**(*database_name: str = 'smarter'*) → MongoClient

Establish a connection to the SMARTER database. Reads environment parameters using `load_dotenv()`, returns a MongoClient object.

Parameters
database_name (*str, optional*) – The smarter database. The default is 'smarter'.

Returns
CLIENT – a mongoclient instance.

Return type
MongoClient

2.5.7 src.features.snpchimp

Created on Tue Feb 16 16:42:36 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

src.features.snpchimp.**clean_chrom**(*chrom: str*)

Return 0 if chrom is 99 (unmapped for snpchimp)

Parameters
chrom (*str*) – the (SNPchiMp) chromsome

Returns
0 if chrom == 99 else chrom

Return type
str

src.features.snpchimp.**read_snpChimp**(*path: str, size=2048*)

2.5.8 src.features.utils

Created on Mon Mar 15 14:13:51 2021

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

class src.features.utils.**TqdmToLogger**(*logger*, *level=None*)

Bases: StringIO

Output stream for TQDM which will output to logger module instead of the StdOut.

__init__(*logger*, *level=None*)

buf = ''

flush()

Flush write buffers, if applicable.

This is not implemented for read-only and non-blocking streams.

level = None

logger = None

write(*buf*)

Write string to file.

Returns the number of characters written, which is always equal to the length of the string.

class src.features.utils.**UnknownCountry**

Bases: object

Deal with unknown country

__init__()

src.features.utils.**camelCase**(*string: str*) → str

Convert a string into camel case

Parameters
string (*str*) – the string to convert

Returns
the camel case version of the string

Return type
str

src.features.utils.**find_duplicates**(*header: list*) → list

Find duplicate columns in list. Returns index to remove after the first occurence

Parameters
header (*list*) – a list like the header read from a CSV file

Returns
a list of index (numeric)

Return type
list

src.features.utils.**get_interim_dir**() → PosixPath

Return smarter data temporary dir

Returns
the smarter data temporary dir

Return type
pathlib.PosixPath

src.features.utils.**get_processed_dir**() → PosixPath

Return smarter data processed dir (final processed data)

Returns
the smarter data final processed dir

Return type
pathlib.PosixPath

src.features.utils.**get_project_dir**() → PosixPath

Return smarter project dir (which are three levels upper from the module in which this function is stored)

Returns
the smarter project base dir

Return type
pathlib.PosixPath

src.features.utils.**get_raw_dir**() → PosixPath

Return smarter data raw dir

Returns
the smarter data raw directory

Return type
pathlib.PosixPath

src.features.utils.**sanitize**(*word: str*, *chars=['.', ',', '-', '/', '#']*, *check_mongoengine=True*) → str

Sanitize a word by removing unwanted characters and lowercase it.

Parameters

- **word** (*str*) – the word to sanitize
- **chars** (*list*) – a list of characters to remove
- **check_mongoengine** (*bool*) – true to add '_' after a mongoengine reserved word

Returns
the sanitized word

Return type
str

src.features.utils.**skip_comments**(*handle: ~_io.TextIOWrapper*, *comment_char='#'*) -> (<*class 'int'*>, <*class 'list'*>)

Ignore comments lines from a open file handle. Return the stream position immediately after the comments and all the comment lines in a list.

Parameters

- **handle** (*io.TextIOWrapper*) – An open file handle.
- **comment_char** (*TYPE, optional*) – The comment character used in file. The default is "#".

Returns
The stream position after the comments and the ignored lines as a list.

Return type
(int, list)

src.features.utils.**text_or_gzip_open**(*path: str*, *mode: Optional[str] = None*) → TextIOWrapper
Open a file which can be compressed or not. Returns file handle

2.6 History

2.6.1 TODO

- Check chromosomes in *Variants locations*: mind to **scaffold**, **null**, and **non-autosomal** chromosomes for *Goat* and *Sheep*
- Rename objects (use names in a consistent way, ex *TOP*, *BOT*)
- Release a *smarter* coordinate version with information on every variant defined in database (which will be used as reference)
- Map affymetrix snps in *OARV3* coordinates
- Check if `rs_id` is still valid or not (with EVA)
- Manage python packages with poetry
- Rename `manifacturer` into `manufacturer`

2.6.2 0.4.9 (2023-09-27)

- Load phenotypes for *Fosses, Provencale* goat breeds
- Add sex for *Fosses, Provencale* goat breeds
- Add *sex* while importing metadata
- Load multiple phenotypes for *Boutsko foreground* sheeps
- Add multiple phenotypes as a list (103)
- Update *datasets* metadata
- Update dependencies

2.6.3 0.4.8 (2023-06-28)

- Capitalize `species_class` parameter in `src.data.import_breeds.py`
- Generate output files for *OARV4* and *CHIR1* (#87)
- Import data from *dbSNP152* (#15)
- Import data from *IGGC* (#18)
- Split `import_consortium.py` in `import_isgc.py` and `import_iggc.py` to import data from Sheep and Goat genome *consortia* respectively
- Force data update when importing from consortium

- Track date when importing from consortium
- Determine `illumina_top` data directly from variant for Sheep when importing from *consortium* data
- Uniform *note* metadata field (add a *note* parameters in import metadata)
- Import data from *Cortellari et al 2021* (https://doi.org/10.1038/s41598-021-89900-2)
- Import data from *Burren et al 2016* (https://doi.org/10.1111/age.12476)
- Revise illumina A/B genotype tracking
- Import from Illumina report with only 3 columns in SNP list file
- Update dependencies

2.6.4 0.4.7 (2022-12-23)

- Import background data from *Gaouar et al 2017* (https://doi.org/10.1038/hdy.2016.86)
- Import from plink with illumina coding (as specified in manifest: not *top* nor *forward*)
- Import background data from *Belabdi et al 2019* (https://doi.org/10.1038/s41598-019-44137-y)
- Import background data from *Ciani et al 2020* (https://doi.org/10.1186/s12711-020-00545-7)
- Import background data from *Barbato et al 2017* (https://doi.org/10.1038/s41598-017-07382-7)
- Update species for *european mouflon*
- Support species update with `import_metadata.py`
- Import 18 *welsh* breed as background genotypes
- Rename two *welsh* breeds
- Model *doi* in datasets
- Upgrade CI workflows to `actions/cache@v3`
- Add `SNPconvert.py` script
- Import genotypes of other WPs coming from Uruguay
- Deal with affymetrix report with less SNPs than declared
- Add an option to skip coordinate check when importing affymetrix report
- Import from affymetrix a limited number of samples
- Skip sample creation when there's no alias
- Support for missing columns in affymetrix report files
- Support *invalid python names* in `src.features.affymetrix.read_affymetrixRow`
- Update requirements
- Deal with missing files in `import_datasets.py`
- Update Uruguay metadata locations
- Move *Galway* sheep to *Ireland* country (Ovine HapMap)

2.6.5 0.4.6 (2022-09-26)

- Update requirements
- Read from affymetrix A/B reportfile
- Import latest Uruguayan data (#65)
- Configure database connection (#66)
- Update sex in ped file if there are information in database
- Enable continuous integration for documentation (ReadTheDocs)
- Update documentation
- Track full species information in Sample (support for multi-species sheep and goats)
- Updated *isheep* exploration notebooks
- Deal with *unknown* countries and species
- Fix issues related on *alias* when creating samples or adding metadata
- Fetch variants using positions
- Import from plink using *genomic coordinates*
- Import *50K*, *600K* and *WGS isheep* datasets (#47)
- Fix issue in `src.features.plinkio.plink_binary_exists`
- Code refactoring in `src.features.plinkio`
- Import data from Sheep HapMap V2

2.6.6 0.4.5 (2022-06-14)

- Update requirements
- Import data from Hungary (#53)
- Create a new sample when having the same `original_id` in dataset but for a different breed
- `illumina_top` is an attribute of variant, and is set when the first location is loaded.
- Check variants data before update (#56)
- Simplified `import_affymetrix` script
- Import custom affymetrix chips (*Oar_v3.1*)
- Support *source* and *destination* assemblies when importing from *plink* or *affymetrix* source files
- Deal with spaces in filenames while importing from plink
- Add `affy_snp_id` primary key
- Update `import_affymetrix.py` script
- Import data from Spain (#52)
- Fix *20220503* dataset breed and *churra* chip name
- Track manifest probe `sequence``s by ``chip_name`
- Track `probeset_id` by `chip_name`

- Search for affymetrix `probeset_id` in the proper `chip_name` while importing samples
- Track multiple `rs_id`
- Fetch *churra* coordinates by `rs_id` and `probeset_id` and filter out unmanaged SNPs
- If `src_dataset` and `dst_dataset` are equals, provide only `src_dataset`

2.6.7 0.4.4 (2022-02-28)

- Model location with `MultiPointField`
- Describe smarter metadata
- Import sweden goat metadata
- Import latest 290 samples greek dataset
- Fix issue with greek samples name (`B273` converted into `B273A`)
- Add latest 19 sheep greek samples
- Add a country collection
- Update dependencies

2.6.8 0.4.3 (2021-11-11)

- Add 270 *Frizarta* background samples
- Import from ab plink and support multiple missing letters
- Track database status and constants
- Add *foreground/background* type attribute in `SampleSpecies`
- Update dependencies
- Add make rule to pack results and make checksum
- Move greek foreground metadata to a custom phenotypes dataset
- Update greek foreground metadata
- Import phenotypes from Uruguay
- Import phenotypes using alias
- Allow phenotypes for ambiguous sex animals
- Import french goat foreground dataset
- Pin `plinkio` to support *extra-chroms* in plink binary files
- Import 5 Sweden Sheep background genotypes
- Force *half-missing* SNPs to be MISSING
- Add the README.txt.ftp
- Bug fixed in importing multibreed reportfile (setting FID properly in output)

2.6.9 0.4.2 (2021-08-27)

- Set nullable `ListField` for sample *locations* and variant *consequences*
- Capitalize phenotype values (ie *milk* -> *Milk*)
- Import greek *chios-mytilini-boutsko* sheep dataset
- Track multiple location for sample (deal with transhumant breeds)
- Import greek *skopelios-eghoria* goat dataset
- Use sample data to deal with multi breeds illumina row files
- Determine fid from database with IlluminaReportIO
- Import greek *frizarta-chios-pelagonia* sheep dataset
- Import greek *frizarta-chios* sheep dataset
- Import sweden foreground goat dataset
- Update *ADAPTmap* breed names and phenotypes import
- Check that breed exists while inserting phenotype data
- Import french foreground sheep dataset
- Use `elemMatch` in projection in `plinkio.SmarterMixin.fetch_coordinates` (ex: `VariantSheep.objects.fields(elemMatch__locations={"imported_from": "SNPchiMp v.3", "version": "Oar_v4.0"})`)
- Use `elemMatch` to search a SNP within the desired coordinate systems in `plinkio.SmarterMixin.fetch_coordinates`
- Skip SNPchimp indels when importing from SNPchimp
- Skip illumina indels when reading from manifest

2.6.10 0.4.1 (2021-09-08)

- Add `chip_name` in Dataset (database value, not user value)
- Skip `null` fields when importing datasets
- Import uruguay sheep affymetrix data
- Import from affymetrix dataset
- Rely on original affymetrix coordinate system to determine illumina top alleles
- Search samples *aliases* while importing genotypes
- Clearly state when creating samples (ignore samples if not defined in database)
- Track sample aliases for `original_id`
- Import samples from file by providing *country* and *breeds* values as parameters
- Import sheep coordinates from genome project
- Security updates
- Fix github Workflow

2.6.11 0.4.0 (2021-06-18)

- `dbSNP` feature library refactor
- fix linter issues
- Transform *affymetrix* unmapped chrom to `0`
- Transform *SNPchiMp* unmapped chroms to `0`
- ignore affymetrix insertions and deletions
- join affymetrix data with illumina relying on `cust_id`
- define `illumina_top` from affymetrix flanking sequences
- load data from affymetrix manifest
- calculate *illumina_top* from affymetrix sequence
- Test import data from *snpchimp*
- Import `OARV4` coordinates
- `data/common` module refactoring
- Fix bug in importing dataset order
- Model affymetrix fields
- Read from affymetrix manifest file
- Track illumina manufactured date

2.6.12 0.3.1 (2021-06-11)

- Upgrade dependencies
- Enable continuous integration
  - Github Workflow
  - Coverage

2.6.13 0.3.0 (2021-05-19)

- Deal with multi-sheets `.xlsx` documents
- Import phenotypes (from a *source* dataset to a *destination* dataset)
- Define phenotype attribute as a `mongoengine.DynamicDocument` field
- Import metadata or phenotype *by breeds* or *by samples*
- Import metadata (from a *source* dataset to a *destination* dataset)
- Forcing `plink` **chrom** options when converting in binary formats
- import data from *ADAPTmap* project
  - Import goat breeds (from a *source* dataset to a *destination* dataset)
  - Import goat data from plink files
  - Import goat metadata

- Import goat data from manifest and snpchimp
- configure `mongodb-express` credentials
- Add Goat Related tables
  - add `variantGoat` collection
  - add `sampleGoat` collection

2.6.14 0.2.3 (2021-05-03)

- Unset ped columns if relationship can't be derived from data (ex. *brazilian BSI*)
- Deal with geographical coordinates
- Add features to samples (relying on metadata file)

2.6.15 0.2.2 (2021-04-29)

- Breed name should be a unique key within species
- make rule to clean-up `interim` data
- skip already processed file from import
- Deal with `mother_id` and `father_id` (search for `smarter_id` in database)
- Deal with multi-countries dataset
  - track country in aliases while importing breeds from dataset

2.6.16 0.2.1 (2021-04-22)

- Track `chip_name` with samples
- Deal with binary plink files
- Search breed by *aliases* used in `dataset`:
  - match *fid* with breed *aliases* in `dataset`
  - store *aliases* by `dataset`
- Add breeds from `.xlsx` files

2.6.17 0.2.0 (2021-04-15)

- Merge multiple files per dataset
- Import from an *illumina report* file
- Deal with *AB* allele coding
- Deal with plink text files using modules
- Fix *SNPchiMp* data import
- Determine `illumina_top` coding as a *property* relying on database data
- Support multi-manifest upload (extend database with *HD* chip)

- Deal with compressed manifest
- Add breeds with *CLI*
- Check coordinates format relying on *DRM*
- Test stuff with `mongomock`

2.6.18 0.1.0 (2021-03-29)

- Start with project documentation
- Explore background datasets
- Merge plink binary files
- Convert from `forward` to `illumina_top` coordinates
- Convert to plink binary format
- Manage database credentials
- Import samples into `smarter` database while fixing coordinates and genotypes
- Configure tox and sphinx environments
- Model breeds in `smarter` database
- Import *datasets* into database
- Read from *dbSNP xml dump* file
- Import *SNPchiMp* data into `smarter` database
- Import *Illumina manifest* data into database
- Model objects with `mongoengine`
- Model *smarter ids*
- Configure environments, requirements and dependencies

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W