## CONTENTS:

1  Documentation Contents  
   1.1  Introduction  ................................................................. 3  
   1.2  Backend installation  ....................................................... 6  
   1.3  Accessing SMARTER-backend  ............................................ 6  
   1.4  Backend Module Documentation  ....................................... 13  

2  Indices and tables  

Python Module Index  

Index  

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1  Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.2  Backend installation</td>
<td>6</td>
</tr>
<tr>
<td>1.3  Accessing SMARTER-backend</td>
<td>6</td>
</tr>
<tr>
<td>1.4  Backend Module Documentation</td>
<td>13</td>
</tr>
<tr>
<td>2  Indices and tables</td>
<td>29</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>31</td>
</tr>
<tr>
<td>Index</td>
<td>33</td>
</tr>
</tbody>
</table>
This documentation describes how to install and interact with the SMARTER database backend. Briefly, the SMARTER database created and maintained with the SMARTER-database project is made accessible to SMARTER partners using this API. This API is the same used by the SMARTER-frontend in order to access and browse SMARTER data using a web browser.

If you are partner of the SMARTER project, and member of WP4 group, you should have received the credentials to access the smarter API. If you aren’t a SMARTER partner, you cannot access to SMARTER data using API, however this API with the WEB frontend (and the genotypes) will be made available to the public at the end of the project. If you are member of WP4 SMARTER project but you don’t have the credentials yet, please send an email to the WP coordinators.

SMARTER-backend is a flask-API application developed on top of a MongoDB instance. The application works inside Docker containers managed with docker-compose. SMARTER WP4 users who need to subset or retrieve genotypes from the entire genotypes dataset need to retrieve variants and samples according their needs, in order to filter out the data they need using a PLINK command line.

Documentation is organized as following: in Introduction we describe what SMARTER-backend is and we provide general information. In Backend installation we describe how to install a local instance of SMARTER-backend. Then in Accessing SMARTER-backend we describe how to programmatically access to data or how to inspect data using applications like Postman or Talend Api Tester google-chrome extension.
1.1 Introduction

The SMARTER-backend is a REST API service developed on top of the SMARTER-database which provides methods to interact and access with SMARTER data using web resources. By using SMARTER-backend, you can develop softwares or scripts which get access to SMARTER data through web, without having a local instance of the SMARTER-database. In a similar way a web-server receives a request from a web-browser and returns text and images that could be rendered in a page, a REST API receives requests over the HTTP protocol and returns data that could be read and manipulated by user softwares.

1.1.1 API endpoints

SMARTER-backend receives requests over the internet and returns SMARTER data in JSON objects. There could be different data types that could be returned by SMARTER-backend, for example Breed, Variant, Sample or Dataset. In order to access to each different data type you have to make a request to the proper API endpoint, which is the mean from which the API can access the resources requested. An API endpoint is an URL, to which HTTP requests are submitted, and from which the response is thus expected. For example, to retrieve information on all the Breeds stored in SMARTER database, you should make a request to the breed endpoint which is:

https://webserver.ibba.cnr.it/smarter-api/breeds

Similarly, to have informations on Goat samples you have to make a request to a different endpoint, which is:

https://webserver.ibba.cnr.it/smarter-api/samples/goat

If you inspect the two previous URL, you may notice that these two endpoint have a prefix in common (https://webserver.ibba.cnr.it/smarter-api), while the last part of the URL changes relying on the data they provide. There are a few endpoints available by SMARTER-backend:
<table>
<thead>
<tr>
<th>Suffix</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/auth/login</td>
<td>Users</td>
<td>user authentication</td>
</tr>
<tr>
<td>/breeds</td>
<td>Breeds</td>
<td>returns a list of the available breeds</td>
</tr>
<tr>
<td>/datasets</td>
<td>Dataset</td>
<td>returns a list of the available datasets</td>
</tr>
<tr>
<td>/info</td>
<td>SmarterInfo</td>
<td>A dictionary of smarter information</td>
</tr>
<tr>
<td>/samples/sheep</td>
<td>SampleSheep</td>
<td>returns a list of the sheep samples</td>
</tr>
<tr>
<td>/samples/goat</td>
<td>SampleGoat</td>
<td>returns a list of goat samples</td>
</tr>
<tr>
<td>/samples.geojson/sheep</td>
<td>GeoJSON</td>
<td>return sheep samples in GeoJSON format</td>
</tr>
<tr>
<td>/samples.geojson/goat</td>
<td>GeoJSON</td>
<td>return goat samples in GeoJSON format</td>
</tr>
<tr>
<td>/supported-chips</td>
<td>SupportedChip</td>
<td>returns a list of chip which provide SNPs to the SMARTER dataset</td>
</tr>
<tr>
<td>/variants/sheep/OAR3</td>
<td>VariantSheep</td>
<td>returns a list of sheep SNPs in OAR3 assembly</td>
</tr>
<tr>
<td>/variants/sheep/OAR4</td>
<td>VariantSheep</td>
<td>returns a list of sheep SNPs in OAR4 assembly</td>
</tr>
<tr>
<td>/variants/goat/ARS1</td>
<td>VariantGoat</td>
<td>returns a list of goat SNPs in ARS1 assembly</td>
</tr>
<tr>
<td>/variants/goat/CHI1</td>
<td>VariantGoat</td>
<td>returns a list of goat SNPs in CHI1 assembly</td>
</tr>
</tbody>
</table>

So if you require to retrieve all the sheep SNPs in OAR3 assembly, you can append the suffix /variants/sheep/OAR3 to the common prefix https://webserver.ibba.cnr.it/smarter-api to obtain the final endpoint:

https://webserver.ibba.cnr.it/smarter-api/variants/sheep/OAR3

Every endpoints described provide a list of results, however you could retrieve a specific object by appending the proper ObjectId to the endpoint, for example:

https://webserver.ibba.cnr.it/smarter-api/datasets/604f75a61a08c53cebd09b5b

will retrieve the dataset with ObjectId 604f75a61a08c53cebd09b5b.

**Warning:** Please note that ObjectId could change over time, since they rely on the time they are added into database. If you require a particular data, you should use the proper API endpoint by providing the appropriate parameters as arguments, for example `file=<file name>` to retrieve the dataset relying on provided file name.

### 1.1.2 HTTP Verbs

An endpoint can act differently relying on the HTTP Verb used when making a request. For SMARTER-backend, only two HTTP Verbs (or methods) are currently supported, GET and POST:

<table>
<thead>
<tr>
<th>Http Verb</th>
<th>CRUD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Read</td>
<td>Retrieve a list of objects or a single object</td>
</tr>
<tr>
<td>POST</td>
<td>Create</td>
<td>Create a new auth token / add a payload to a query</td>
</tr>
</tbody>
</table>

More precisely, SMARTER data through the SMARTER-backend are read-only and GET and POST method are allowed in order to retrieve any type of SMARTER data object from the API. POST method are required during the authentication step, and to submit a complex queries by providing a payload to certain endpoints.
1.1.3 Status codes

SMARTER-backend API uses standard response status code to show the outcome of each HTTP request. Briefly, replies with a status code like 2xx are successful requests, 4xx codes means errors in client side (you are using the API in the wrong way) and 5xx means errors on the server side (you should get in touch with API maintainer and describe what went wrong). You can find a complete reference on HTTP status codes here.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 Ok</td>
<td>A request completed with success</td>
</tr>
<tr>
<td>400 Bad request</td>
<td>The request was malformed. The response body will include an error providing further information</td>
</tr>
<tr>
<td>401 UnAuthorized</td>
<td>Request lacks of the required authorization header or token is expired</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>Requested object or endpoint doesn’t exist</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>The server encountered an unexpected condition which prevented it from fulfilling the request</td>
</tr>
</tbody>
</table>

1.1.4 JWT Authentication

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed (https://jwt.io/introduction)

SMARTER data aren’t available to the public yet, only SMARTER WP4 should have the credentials to use the SMARTER-backend API. If you are a WP4 member and you don’t have the required credentials, please get in touch with WP4 group leaders. In order to get access to SMARTER data, you are required to generate a JWT token by providing your credentials as a POST request to the API authentication endpoint, which is:

https://webserver.ibba.cnr.it/smarter-api/auth/login

If you credentials are valid, you will receive a JSON object with your generated token and an expires date. By default a token will be valid for 7 days after it was generated. You need to add this token as an Authorization header to all of your API requests. The format is Authorization: Bearer <your token>. In the Accessing SMARTER-backend section you will find furter instructions on how to generate your token and how you could interact with the SMARTER-backend API.

Danger: Please don’t share your API credentials or the generated token with anyone, this includes also code that could be shared using github: credentials or generated tokens should be never stored in your code
1.2 Backend installation


1.3 Accessing SMARTER-backend

1.3.1 Authentication

In the following sections you will find examples on how to interact with the SMARTER-backend in different ways. Regardless the method your prefer, you are required to generate a JWT Token before accessing data. This token need to be added to each header request made to any SMARTER-backend endpoint, with the following format:

```
Authorization: Bearer <your token>
```

Please remember that the token is valid only for 7 days, then you will need to generate a new JWT token in order to access SMARTER metadata.

1.3.2 Query parameters

The API endpoints described in the Introduction return by default all the SMARTER objects they managed. However it’s possible to filter out the results returned using GET parameters (also called URL parameters or query strings) which are usually name-value pairs, separated by an = sign. For example, by submitting a request to:

```
https://webserver.ibba.cnr.it/smarter-api/breeds
```

You will get all the Breeds stored in smarter database, but you can filter out the results by species by passing species=Goat parameter:

```
https://webserver.ibba.cnr.it/smarter-api/breeds?species=Goat
```

Please note that parameters are not part of the API endpoint. The question mark is used as a separator, and divide the endpoint from the GET parameter. You can provide multiple parameters by joining them with the & character, for example:

```
https://webserver.ibba.cnr.it/smarter-api/breeds?species=Goat&search=land
```

will retrieve all the SMARTER goat breeds which have land in ther name (Landrace, for example, but also Rangeland)

Hint: The page https://webserver.ibba.cnr.it/smarter-api/docs/ describes every API endpoints with their own set of parameters, see the proper endpoint documentation to have a list of the allowed parameters and what they do.
1.3.3 Pagination

API queries could returned thousands of results, so to improve performance and lower traffic between client and server, all the API endpoints implements pagination. This means that each query returns a limited set of results, but it returns also the total number of objects with informations useful to collect the next batch of objects. For example, if you analyze the breed API response while search for all the objects (https://webserver.ibba.cnr.it/smarter-api/breeds), you will see a reply like this:

```
{
    "items": [
        ...
    ],
    "next": "/smarter-api/breeds?size=10&page=2",
    "page": 1,
    "pages": 26,
    "prev": null,
    "size": 10,
    "total": 257
}
```

Where in the `items` array there will be size Breed objects (default 10, omitted here to better describe the response); in the `next` attribute there will be the URLs to be used to get the next batch of objects, if you get the next page, you will get a `prev` attribute for the previous page; The `total` stands for the total number of breed objects and the `page` stands for the current batch page number. By default, the behaviour is to display 10 results per page, however you could change this by setting a different page size with a get parameter, for example `size=20`.

**Warning:** Please remember that pagination helps to better manage resources, don’t try to retrieve all the results for a single query request: there could be a size limit or you can have issues in retrieve / process the results.

1.3.4 Examples

Here we list some patterns on how to interact with the SMARTER-backend, feel free to follow the method you prefer:

**Accessing data using Third party softwares**

Accessing data using Python


Accessing data using R

Here are some examples on how to interact with SMARTER-backend API using R. You will need to set up some utility functions in order to save your time by avoiding repeating stuff.

Importing R libraries

First of all, let’s start with importing some R libraries (maybe you will need to install some of them first):

```r
library(httr)
library(jsonlite)
library(getPass)
library(dplyr)
```

httr is required to send requests and get response from SMARTER-backend API; jsonlite is required to parse JSON output, which is the default format of the API response. getPass is not strictly required, it will prompt for our credentials in order to not store them in our code. dplyr is useful to manage dataframes, for example when they have different columns (like response from SMARTER-backend)

Generate a JWT token with R

As stated in our Authentication section of this guide, you need to generate a JWT token in order to get full access to smarter metadata. Here is an utility function to request a token by providing your credentials with a POST HTTP method:

```r
base_url <- "https://webserver.ibba.cnr.it"

get_smarter_token <-
  function(username = readline(prompt = "Username ? "),
           password = getPass::getPass("Password ? ")) {
    auth_url <-
      http::modify_url(base_url, path = "/smarter-api/auth/login")

    resp <-
      POST(
        auth_url,
        body = list(username = username, password = password),
        encode = "json"
      )

    # this will read a JSON by default

    # this will read a JSON by default
```

(continues on next page)
data <- httr::content(resp)

# returning only the token as a string
return(data$token)

}  

token <- get_smarter_token()

base_url is defined for simplicity in order to make all our request to the same server. The `get_smarter_token` function requires `user` and `password` as parameters. The `readline` and `getPass::getPass` functions used as default values are not strictly required, we use them in order to not write credentials in our code: the function will prompt for those values if not provided during function call. The token string is parsed and written into `token` variable: This is the value we need to add to each requests `header`

**Hint:** Rstudio has a dedicated section on Securing Credentials. We recommend to follow their guidelines.

### Deal with data and pagination

Next, before starting query SMARTER-backend, we can define more utility functions (as suggested by Best practices for API packages) in order to deal with pagination and API errors. We will read our data with `jsonlite` package in order to `flatten` our results (read nested object and add them as columns in the resulting dataframe):

```r
read_url <- function(url, token, query = list()) {
# in this request, we add the token to the request header section
resp <-
    GET(url, query = query, add_headers(Authorization = paste("Bearer", token)))

# check errors: SMARTER-backend is supposed to return JSON objects
if (http_type(resp) != "application/json") {
    stop("API did not return json", call. = FALSE)
}

# parse a JSON response. fromJSON to flatten results
parsed <-
    jsonlite::fromJSON(
        content(resp, "text", encoding = "utf-8"),
        flatten = TRUE
    )

# deal with API errors: not "200 Ok" status
if (http_error(resp)) {
    stop(
        sprintf("SMARTER API returned an error [%s]: '%s'",
            status_code(resp),
            parsed$message
        ),
        call. = FALSE
    )
}
```

(continues on next page)
Our functions will take an `url` parameter, which will be our API endpoint, the `token` that will be added in the header request as described in *Authentication* section of our documentation and `query`, which will be a list of parameters that will enhance our queries as described in *Query parameters*.

**Read data with R**

Next we can try to read data from our API by defining custom functions around the desired endpoint. This function will call the functions previously defined and will return all the results in a *dataframe*. Here’s a sample function to deal with datasets objects by querying the *datasets* endpoint:

```r
get_smarter_datasets <- function(token, query=list()) {
  url <-
    modify_url(base_url, path = "/smarter-api/datasets")
  data <- get_smarter_data(url, token, query)

  # returning only the results dataframe
  data$results
}
```

(continues on next page)
all_datasets <- get_smarter_datasets(token)

By calling the defined get_smarter_datasets function and providing a valid token as parameter you will retrieve all datasets and you will store them in the all_datasets dataframe. Similarly, to deal with the Breed endpoint you could define the get_smarter_breeds function:

```r
get_smarter_breeds <- function(token, query = list()) {
  # setting the URL endpoint
  url <- httr::modify_url(base_url, path = "/smarter-api/breeds")

  # reading our data
  data <- get_smarter_data(url, token, query)

  # returning only the results dataframe
  data$results
}
goat_breeds <-
  get_smarter_breeds(token, query = list(species = "Goat"))
```

get_smarter_breeds and get_smarter_datasets functions can be used to return all the SMARTER datasets and breeds. However you can pass additional parameters to the endpoint using the query parameter (which need to be a list). For example, you could retrieve all the genotypes datasets using the type parameter:

```r
genotypes_datasets <- get_smarter_datasets(token, query = list(type="genotypes"))
```

Since query accepts list, you can specify the same parameter multiple times (if the endpoints supports this type of query, see api docs to get more information). For example, if you need only the foreground genotypes, you can select dataset like this:

```r
foreground_genotypes_datasets <- get_smarter_datasets(
  token, query = list(type="genotypes", type="foreground"))
```

You can add other parameters to refine your query, for example if you want to select only the Goat breeds, you can specify species = "Goat" in the query parameter. If you need also to search for the land term in the breed name, you will call the same function by adding a new parameter:

```r
search_goat_breeds <-
  get_smarter_breeds(token, query = list(
    species = "Goat", search = "land")
)
```

search_goat_breeds will be a dataframe with the same results of the query URL:

```r
https://webserver.ibba.cnr.it/smarter-api/breeds?species=Goat&search=land
```

We can select only the column we need by subsetting dataframe columns, or using dplyr select:

```r
search_goat_breeds <- search_goat_breeds %>% select(name, code)
```

Breed code and names can be used to get from samples from the proper endpoint. Let’s define another function that could be used for sheep and goat samples endpoints relying on parameters:
get_smarter_samples <- function(token, species, query = list()) {
  # mind that species is lowercase in endpoint url
  species <- tolower(species)

  url <-
    modify_url(base_url, path = sprintf("/smarter-api/samples/%s", species))

  data <- get_smarter_data(url, token, query)

  # returning only the results dataframe
  data$results
}

landrace_samples <- get_smarter_samples(
  token,
  species = "Goat",
  query = list(breed_code = "LNR")
)

As for the breed example, we can refine our query, for example by selecting Landrace goat samples which have a locations (GPS coordinates) and phenotypes defined (mind to the double _ in locations__exists and phenotype__exists):

selected_landrace_samples <- get_smarter_samples(  
  token,  
  species = "Goat",  
  query = list(    
    breed_code = "LNR",    
    locations__exists = TRUE,    
    phenotype__exists = TRUE)  
)

As before we can select the smarter_id columns, to have a list of our samples in order to subset the full genotype file using plink:

selected_landrace_samples %>% select(smarter_id)

The same could be applied on variants endpoins in order to get information on variants. In the following example we will select the goat variants on chromosome 1 within 1-1000000 position in ARS1 assembly:

get_smarter_variations <- function(token, species, query = list()) {
  # mind that species is lowercase in endpoint url
  species <- tolower(species)
  assembly <- toupper(assembly)

  url <-
    modify_url(base_url, path = sprintf("/smarter-api/variants/%s/%s", assembly))

  data <- get_smarter_data(url, token, query)

  # returning only the results dataframe
  data$results
}
selected_goat_variations <- get_smarter_variations(
    token,
    species = "Goat",
    assembly = "ARS1",
    query = list(
        size = 100,
        region = "1:1-1000000"
    )
)

**Hint:** We are planning to simplify the variants response by returning a SNP list of the selected SNPs only, in order to be used when subsetting a genotype file using plink

**Warning:** Be careful when using the variants endpoints: getting all the variants will takes a lot of time and could fill all your available memory. Avoid to request all variants in your R session, unless you know what you are doing

### 1.4 Backend Module Documentation

#### 1.4.1 common package

**common.views module**

Created on Mon May 24 16:34:05 2021

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

This module is an attempt to define class based views like the django ones

```python
exception common.views.ImproperlyConfigured
    Bases: Exception
class common.views.ListView
    Bases: flask_restful.Resource
        endpoint = None
        get_context_data()
        get_queryset()
        model = None
        object_list = None
        order_by = None
        page = 1
        parse_args() → list
        parser = <flask_restful.reqparse.RequestParser object>
```
queryset = None
size = 10
class common.views.ModelView
    Bases: flask_restful.Resource
    get_object(pk, queryset=None)
        Return the object the view is displaying. Require self.queryset and a pk or slug argument in the URLconf. Subclasses can override this to return any object.

    get_queryset()
        Return the QuerySet that will be used to look up the object. This method is called by the default implementation of get_object() and may not be called if get_object() is overridden.

model = None
queryset = None

1.4.2 database package

database.db module

Created on Fri May 21 18:09:08 2021
@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>
database.db.initialize_db(app)

database.models module

Created on Fri May 21 18:09:30 2021
@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>
class database.models.Breed(*args, **values)
    Bases: flask_mongoengine.Document

exception DoesNotExist
    Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned

aliases
    A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

    If using with ReferenceFields see: Many to Many with ListFields

    Note: Required means it cannot be empty - as the default for ListFields is []

code
    A unicode string field.

id
    A field wrapper around MongoDB’s ObjectIds.
n_individuals
    32-bit integer field.

name
    A unicode string field.

objects
    The default QuerySet Manager.
    Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality.
    Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.
    The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

species
    A unicode string field.

class database.models.BreedAlias(*args, **kwargs)
    Bases: mongoengine.document.EmbeddedDocument
    country
        A unicode string field.

dataset
    A reference to a document that will be automatically dereferenced on access (lazily).
    Note this means you will get a database I/O access everytime you access this field. This is necessary
    because the field returns a Document which precise type can depend of the value of the _cls field present
    in the document in database. In short, using this type of field can lead to poor performances (especially if
    you access this field only to retrieve it pk field which is already known before dereference). To solve this
    you should consider using the LazyReferenceField.
    Use the reverse_delete_rule to handle what should happen if the document the field is referencing is deleted.
    EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an InvalidDocumentError
    will be raised if trying to set on one of these Document / Field types.
    The options are:
    • DO NOTHING (0) - don’t do anything (default).
    • NULLIFY (1) - Updates the reference to null.
    • CASCADE (2) - Deletes the documents associated with the reference.
    • DENY (3) - Prevent the deletion of the reference object.
    • PULL (4) - Pull the reference from a ListField of references
    Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

    class Org(Document):
        owner = ReferenceField('User')

    class User(Document):
        org = ReferenceField('Org', reverse_delete_rule=CASCADE)

    User.register_delete_rule(Org, 'owner', DENY)

fid
    A unicode string field.
class database.models.Consequence(*args, **kwargs)
    Bases: mongoengine.document.EmbeddedDocument

class database.models.Country(*args, **values)
    Bases: flask_mongoengine.Document
    A helper class to deal with countries object. Each record is created after data import, when database status is updated

    exception DoesNotExist
        Bases: mongoengine.errors.DoesNotExist

    exception MultipleObjectsReturned
        Bases: mongoengine.errors.MultipleObjectsReturned

    alpha_2
        A unicode string field.

    alpha_3
        A unicode string field.

    id
        A field wrapper around MongoDB’s ObjectIds.

    name
        A unicode string field.

    numeric
        32-bit integer field.

    objects
        The default QuerySet Manager.

        Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

        The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

    official_name
        A unicode string field.

    species
        A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

        If using with ReferenceFields see: Many to Many with ListFields

    Note: Required means it cannot be empty - as the default for ListFields is []

class database.models.Dataset(*args, **values)
    Bases: flask_mongoengine.Document
    Describe a dataset instance with fields owned by data types

    exception DoesNotExist
        Bases: mongoengine.errors.DoesNotExist

    exception MultipleObjectsReturned
        Bases: mongoengine.errors.MultipleObjectsReturned
**breed**
A unicode string field.

**chip_name**
A unicode string field.

**contents**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields

---

**country**
A unicode string field.

**doi**
The publication DOI of this dataset

**file**
A unicode string field.

**gene_array**
A unicode string field.

**id**
A field wrapper around MongoDB’s ObjectIds.

**n_of_individuals**
32-bit integer field.

**n_of_records**
32-bit integer field.

**objects**
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

**partner**
A unicode string field.

**size_**
A unicode string field.

**species**
A unicode string field.

**trait**
A unicode string field.

**type_**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields
**Class** database.models.Location(*args, **kwargs)

- **affymetrix_ab**
  - A unicode string field.
- **alleles**
  - A unicode string field.
- **chrom**
  - A unicode string field.
- **consequences**
  - A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.
  
  If using with ReferenceFields see: Many to Many with ListFields

**Note:** Required means it cannot be empty - as the default for ListFields is []

- **date**
  - Datetime field.
  
  Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil’s parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.
  
  Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

  **Note:** Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use ComplexDateTimeField if you need accurate microsecond support.

- **illumina**
  - A unicode string field.
- **illumina_forward**
  - A unicode string field.
- **illumina_strand**
  - A unicode string field.
- **property illumina_top**
  - Return genotype in illumina top format

- **imported_from**
  - A unicode string field.

- **position**
  - 32-bit integer field.

- **ss_id**
  - A unicode string field.

- **strand**
  - A unicode string field.
version
   A unicode string field.

class database.models.Phenotype(*args, **kwargs)
   Bases: mongoengine.document.DynamicEmbeddedDocument
   A class to deal with Phenotype. A dynamic document and not a generic DictField since that there can be attributes
   which could be enforced to have certain values. All other attributes could be set without any assumptions
   
   chest_girth
      Floating point number field.
   height
      Floating point number field.
   length
      Floating point number field.
   purpose
      A unicode string field.

class database.models.Probeset(*args, **kwargs)
   Bases: mongoengine.document.EmbeddedDocument
   
   chip_name
      A unicode string field.
   probeset_id
      A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the
      database.

      If using with ReferenceFields see: Many to Many with ListFields

      Note: Required means it cannot be empty - as the default for ListFields is []


class database.models.SAMPLETYPE(value)
   Bases: enum.Enum
   An enumeration.
   
   BACKGROUND = 'background'
   FOREGROUND = 'foreground'

class database.models.SEX(value)
   Bases: bytes, enum.Enum
   An enumeration.
   
   FEMALE = 2
   MALE = 1
   UNKNOWN = 0

class database.models.SampleGoat(*args, **values)
   Bases: database.models.SampleSpecies

   exception DoesNotExist
      Bases: mongoengine.errors.DoesNotExist

   exception MultipleObjectsReturned
      Bases: mongoengine.errors.MultipleObjectsReturned
father_id
A really lazy reference to a document. Unlike the ReferenceField it will not be automatically (lazily) dereferenced on access. Instead, access will return a LazyReference class instance, allowing access to pk or manual dereference by using fetch() method.

id
A field wrapper around MongoDB’s ObjectIds.

mother_id
A really lazy reference to a document. Unlike the ReferenceField it will not be automatically (lazily) dereferenced on access. Instead, access will return a LazyReference class instance, allowing access to pk or manual dereference by using fetch() method.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

species
A unicode string field.

species_class = 'Goat'
class database.models.SampleSheep(*args, **values)
    Bases: database.models.SampleSpecies

exception DoesNotExist
    Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned

father_id
A really lazy reference to a document. Unlike the ReferenceField it will not be automatically (lazily) dereferenced on access. Instead, access will return a LazyReference class instance, allowing access to pk or manual dereference by using fetch() method.

id
A field wrapper around MongoDB’s ObjectIds.

mother_id
A really lazy reference to a document. Unlike the ReferenceField it will not be automatically (lazily) dereferenced on access. Instead, access will return a LazyReference class instance, allowing access to pk or manual dereference by using fetch() method.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

species
A unicode string field.
species_class = 'Sheep'

class database.models.SampleSpecies(*args, **values):
    Bases: flask_mongoengine.Document

    alias
        A unicode string field.

    breed
        A unicode string field.

    breed_code
        A unicode string field.

    chip_name
        A unicode string field.

    country
        A unicode string field.

    dataset
        A reference to a document that will be automatically dereferenced on access (lazily).

        Note this means you will get a database I/O access every time you access this field. This is necessary because the field returns a Document which precise type can depend of the value of the _cls field present in the document in database. In short, using this type of field can lead to poor performances (especially if you access this field only to retrieve it pk field which is already known before dereference). To solve this you should consider using the LazyReferenceField.

        Use the reverse_delete_rule to handle what should happen if the document the field is referencing is deleted. EmbeddedDocuments, DictFields and MapFields does not support reverse_delete_rule and an InvalidDocumentError will be raised if trying to set on one of these Document / Field types.

        The options are:
        • DO NOTHING (0) - don’t do anything (default).
        • NULLIFY (1) - Updates the reference to null.
        • CASCADE (2) - Deletes the documents associated with the reference.
        • DENY (3) - Prevent the deletion of the reference object.
        • PULL (4) - Pull the reference from a ListField of references

        Alternative syntax for registering delete rules (useful when implementing bi-directional delete rules)

        class Org(Document):
            owner = ReferenceField('User')

        class User(Document):
            org = ReferenceField('Org', reverse_delete_rule=CASCADE)

        User.register_delete_rule(Org, 'owner', DENY)

    locations
        A GeoJSON field storing a list of Points.

        The data is represented as:

        ```
        {'type': 'MultiPoint',
         'coordinates': [[x1, y1], [x2, y2]]
        ```
You can either pass a dict with the full information or a list to set the value.

Requires mongodb >= 2.6

**metadata**
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

**Note:** Required means it cannot be empty - as the default for DictFields is {}

**original_id**
A unicode string field.

**phenotype**
An embedded document field - with a declared document_type. Only valid values are subclasses of EmbeddedDocument.

**sex**
Enumeration Field. Values are stored underneath as is, so it will only work with simple types (str, int, etc) that are bson encodable

Example usage:

```python
class Status(Enum):
    NEW = 'new'
    ONGOING = 'ongoing'
    DONE = 'done'

class ModelWithEnum(Document):
    status = EnumField(Status, default=Status.NEW)

ModelWithEnum(status='done')
ModelWithEnum(status=Status.DONE)
```

Enum fields can be searched using enum or its value:

```python
ModelWithEnum.objects(status='new').count()
ModelWithEnum.objects(status=Status.NEW).count()
```

The values can be restricted to a subset of the enum by using the `choices` parameter:

```python
class ModelWithEnum(Document):
    status = EnumField(Status, choices=[Status.NEW, Status.DONE])
```

**smarter_id**
A unicode string field.

**species_class** = None

**type_**
Enumeration Field. Values are stored underneath as is, so it will only work with simple types (str, int, etc) that are bson encodable

Example usage:

```python
class Status(Enum):
    NEW = 'new'
```

(continues on next page)
ongoing = 'ongoing'
DONE = 'done'

class ModelWithEnum(Document):
    status = EnumField(Status, default=Status.NEW)

ModelWithEnum(status='done')
ModelWithEnum(status=Status.DONE)

Enum fields can be searched using enum or its value:

ModelWithEnum.objects(status='new').count()
ModelWithEnum.objects(status=Status.NEW).count()

The values can be restricted to a subset of the enum by using the choices parameter:

class ModelWithEnum(Document):
    status = EnumField(Status, choices=[Status.NEW, Status.DONE])

exception database.models.SmarterDBException
    Bases: Exception

class database.models.SmarterInfo(*args, **values)
    Bases: flask_mongoengine.Document

        A class to track database status informations

exception DoesNotExist
    Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
    Bases: mongoengine.errors.MultipleObjectsReturned

id
    A unicode string field.

last_updated
    Datetime field.

        Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil’s parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

        Note: To default the field to the current datetime, use: DateTimeField(default=datetime.utcnow)

        Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use ComplexDateTimeField if you need accurate microsecond support.

objects
    The default QuerySet Manager.

        Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality.
        Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

        The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.
plink_specie_opt
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

version
A unicode string field.

working_assemblies
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.

Note: Required means it cannot be empty - as the default for DictFields is {}

class database.models.SupportedChip(*args, **values)
    Bases: flask_mongoengine.Document

    exception DoesNotExist
        Bases: mongoengine.errors.DoesNotExist

    exception MultipleObjectsReturned
        Bases: mongoengine.errors.MultipleObjectsReturned

    id
        A field wrapper around MongoDB’s ObjectIds.

    manufacturer
        A unicode string field.

    n_of_snps
        32-bit integer field.

    name
        A unicode string field.

    objects
        The default QuerySet Manager.

        Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

        The method function should return a QuerySet, probably the same one that was passed in, but modified in some way.

    species
        A unicode string field.

class database.models.User(*args, **values)
    Bases: flask_mongoengine.Document

    exception DoesNotExist
        Bases: mongoengine.errors.DoesNotExist

    exception MultipleObjectsReturned
        Bases: mongoengine.errors.MultipleObjectsReturned

    check_password(password)
id
A field wrapper around MongoDB’s ObjectIds.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet`, probably the same one that was passed in, but modified in some way.

password
A unicode string field.

username
A unicode string field.

class database.models.VariantGoat(*args, **values)
Bases: database.models.VariantSpecies

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

id
A field wrapper around MongoDB’s ObjectIds.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet`, probably the same one that was passed in, but modified in some way.

class database.models.VariantSheep(*args, **values)
Bases: database.models.VariantSpecies

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

id
A field wrapper around MongoDB’s ObjectIds.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet`, probably the same one that was passed in, but modified in some way.
**class database.models.VariantSpecies(*args, **values)**

**Bases:** flask_mongoengine.Document

**affy_snp_id**
A unicode string field.

**chip_name**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields

*Note:* Required means it cannot be empty - as the default for ListFields is []

**cust_id**
A unicode string field.

**illumina_top**
A unicode string field.

**locations**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields

*Note:* Required means it cannot be empty - as the default for ListFields is []

**name**
A unicode string field.

**probesets**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields

*Note:* Required means it cannot be empty - as the default for ListFields is []

**rs_id**
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: Many to Many with ListFields

*Note:* Required means it cannot be empty - as the default for ListFields is []

**sender**
A unicode string field.

**sequence**
A dictionary field that wraps a standard Python dictionary. This is similar to an embedded document, but the structure is not defined.
Note: Required means it cannot be empty - as the default for DictFields is {}

to_mongo(*args, **kwargs)
    Override flask-mongoengine method
database.models.complement(genotype: str)
INDICES AND TABLES

• genindex
• modindex
• search
c
  common, 13
  common.views, 13

d
  database, 14
  database.db, 14
  database.models, 14
| A | affy_snp_id (database.models.VariantSpecies attribute), 26 |
| B | affymetrix_ab (database.models.Location attribute), 18 |
| | alias (database.models.SampleSpecies attribute), 21 |
| | aliases (database.models.BreedSpecies attribute), 14 |
| | alleles (database.models.Location attribute), 18 |
| | alpha_2 (database.models.Country attribute), 16 |
| | alpha_3 (database.models.Country attribute), 16 |
| C | BACKGROUND (database.models.SAMPLETYPE attribute), 19 |
| | check_password() (database.models.User method), 24 |
| | chest_girth (database.models.Phenotype attribute), 19 |
| | chip_name (database.models.Dataset attribute), 17 |
| | chip_name (database.models.Probeset attribute), 19 |
| | chrom (database.models.Location attribute), 18 |
| | code (database.models.Breed attribute), 14 |
| | common module, 13 |
| | complement() (in module database.models), 27 |
| | Consequence (class in database.models), 15 |
| | consequences (database.models.Location attribute), 18 |
| | contents (database.models.Dataset attribute), 17 |
| | Country (class in database.models), 16 |
| | country (database.models.BreedSpecies attribute), 15 |
| | country (database.models.Dataset attribute), 17 |
| | country (database.models.SampleSpecies attribute), 21 |
| | Country.DoesNotExist, 16 |
| | Country.MultipleObjectsReturned, 16 |
| | cust_id (database.models.VariantSpecies attribute), 26 |
| D | database module, 14 |
| | database.db module, 14 |
| | dataset (database.models.BreedSpecies attribute), 21 |
| | dataset (database.models.Dataset attribute), 17 |
| | Dataset (class in database.models), 16 |
| | Dataset.DoesNotExist, 16 |
| | Dataset.MultipleObjectsReturned, 16 |
| | date (database.models.Location attribute), 18 |
| | doi (database.models.Dataset attribute), 17 |
| E | endpoint (common.views.ListView attribute), 13 |
| F | father_id (database.models.SampleGoat attribute), 19 |
| | father_id (database.models.SampleSheep attribute), 20 |
| | FEMALE (database.models.SEX attribute), 19 |
| | fid (database.models.BreedSpecies attribute), 15 |
| | file (database.models.Dataset attribute), 17 |
| | FOREGROUND (database.models.SAMPLETYPE attribute), 19 |
| G | gene_array (database.models.Dataset attribute), 17 |
| | get_context_data() (common.views.ListView method), 13 |
| | get_object() (common.views.ModelView method), 14 |
| | get_queryset() (common.views.ListView method), 13 |
Index

H
height (database.models.Pheno-ty attribute), 19

I
id (database.models.Breed attribute), 14
id (database.models.Country attribute), 16
id (database.models.Dataset attribute), 17
id (database.models.SampleSheep attribute), 20
id (database.models.SampleGoat attribute), 20
id (database.models.SmarterInfo attribute), 23
id (database.models.SupportedChip attribute), 24
id (database.models.User attribute), 24
id (database.models.VariantGoat attribute), 25
id (database.models.VariantSheep attribute), 25
illumina (database.models.Location attribute), 18
illumina_forward (database.models.Location attribute), 18
illumina_strand (database.models.Location attribute), 18
illumina_top (database.models.Location property), 18
illumina_top (database.models.VariantSpecies attribute), 26
imported_from (database.models.Location attribute), 18
ImproperlyConfigured, 13
initialize_db() (in module database.db), 14

L
last_updated (database.models.SmarterInfo attribute), 23
length (database.models.Pheno-ty attribute), 19
ListView (class in common.views), 13
Location (class in database.models), 18
locations (database.models.SampleSpecies attribute), 21
locations (database.models.VariantSpecies attribute), 26

M
MALE (database.models.SEX attribute), 19
manufacturer (database.models.SupportedChip attribute), 24
metadata (database.models.SupportedChip attribute), 22
model (common.views.ListView attribute), 13
model (common.views.ModelView attribute), 14
ModelView (class in common.views), 14
module
    common, 13
    common.views, 13
    database, 14
    database.db, 14

database.models, 14
mother_id (database.models.SampleGoat attribute), 20
mother_id (database.models.SampleSheep attribute), 20

N
n_individuals (database.models.Breed attribute), 14
n_of_individuals (database.models.Dataset attribute), 17
n_of_records (database.models.Dataset attribute), 17
n_of_snps (database.models.SupportedChip attribute), 24
name (database.models.Breed attribute), 15
name (database.models.Country attribute), 16
name (database.models.SupportedChip attribute), 24
name (database.models.VariantSpecies attribute), 26
numeric (database.models.Country attribute), 16

O
object_list (common.views.ListView attribute), 13
objects (database.models.Breed attribute), 15
objects (database.models.Country attribute), 16
objects (database.models.Dataset attribute), 17
objects (database.models.SampleGoat attribute), 20
objects (database.models.SampleSheep attribute), 20
objects (database.models.SmarterInfo attribute), 23
objects (database.models.SupportedChip attribute), 24
objects (database.models.User attribute), 25
objects (database.models.VariantGoat attribute), 25
objects (database.models.VariantSheep attribute), 25
official_name (database.models.Country attribute), 16
order_by (common.views.ListView attribute), 13
original_id (database.models.SampleSpecies attribute), 22

P
page (common.views.ListView attribute), 13
parse_args() (common.views.ListView method), 13
parser (common.views.ListView attribute), 13
partner (database.models.Dataset attribute), 17
password (database.models.User attribute), 25
Phenotype (class in database.models), 19
phenotype (database.models.SampleSpecies attribute), 22
plink_species_opt (database.models.SmarterInfo attribute), 23
position (database.models.Location attribute), 18
Probeset (class in database.models), 19
probeset_id (database.models.Probeset attribute), 19
probesets (database.models.VariantSpecies attribute), 26
purpose (database.models.Pheno-ty attribute), 19

Q
queryset (common.views.ListView attribute), 13
queryset (common.views.ModelView attribute), 14
R
rs_id (database.models.VariantSpecies attribute), 26
S
SampleGoat (class in database.models), 19
SampleGoat.DoesNotExist, 19
SampleGoat.MultipleObjectsReturned, 19
SampleSheep (class in database.models), 20
SampleSheep.DoesNotExist, 20
SampleSheep.MultipleObjectsReturned, 20
SampleSpecies (class in database.models), 21
SAMPLETYPE (class in database.models), 19
sender (database.models.VariantSpecies attribute), 26
sequence (database.models.VariantSpecies attribute), 26
SEX (class in database.models), 19
sex (database.models.SampleSpecies attribute), 22
size (common.views.ListView attribute), 14
size_ (database.models.Dataset attribute), 17
smarter_id (database.models.SampleSpecies attribute), 22
SmarterDBException, 23
SmarterInfo (class in database.models), 23
SmarterInfo.DoesNotExist, 23
SmarterInfo.MultipleObjectsReturned, 23
species (database.models.Breed attribute), 15
species (database.models.Country attribute), 16
species (database.models.Dataset attribute), 17
species (database.models.SampleGoat attribute), 20
species (database.models.SampleSheep attribute), 20
species (database.models.SupportedChip attribute), 24
species_class (database.models.SampleGoat attribute), 20
species_class (database.models.SampleSheep attribute), 20
species_class (database.models.SampleSpecies attribute), 22
ss_id (database.models.Location attribute), 18
strand (database.models.Location attribute), 18
SupportedChip (class in database.models), 24
SupportedChip.DoesNotExist, 24
SupportedChip.MultipleObjectsReturned, 24
T
to_mongo() (database.models.VariantSpecies method), 27
trait (database.models.Dataset attribute), 17
type_ (database.models.Dataset attribute), 17
type_ (database.models.SampleSpecies attribute), 22
U
UNKNOWN (database.models.SEX attribute), 19