AnnotationSets 1.4.0

Clinical data is supported in *File, Sample, Cohort, Individual* and *Family* in a field called *annotationSets*. Any of these entities will be able to perform the same operations described below apart from their own particular features.

In this document, we will be referring to *annotationSets* and *annotations* (the field names used in OpenCGA to store any clinical data or any other user-defined free data model).

Clinical data ingestion

Create or remove a whole

In order to add new clinical information to one entity, the user will need to call to the main */update* web service of the entity (*files/{file}/update, samples/{sample}/update*, etc.). These web services accept a list of *annotationSets* using the format described above. By default, any time the user sends a list of *a nnotationSets* without adding any other parameter, those *annotationSets* containing the clinical information will be added to the already existing *annotation Sets* the entity might contain (if any). However, this behaviour can be altered by changing the value of the ***annotationSetsAction*** query parameter. This parameter accepts 3 possible action values:

- **ADD:** This is the default behavior, even if the query parameter is not sent. Any *annotationSets* sent through the main body of the POST operation will be added to the already existing ones (if any).
- **SET:** Replace the current existing *annotationSets* (if any) of the entity being updated by the ones sent through the main body of the POST operation.
- **REMOVE:** Remove the list of *annotationSets* sent through the main body of the POST operation if they already exist in the entity being updated. In this case, the only field that is necessary and therefore taken into account, will be the *annotationSet* ***id*****.**

Updating some values of already stored clinical data

In order to update only a few values (*annotations*) of an already stored *annotationSet,* users will need to call to a new web service *.../annotationSets/{annotationSetId}/annotations/update* present for the different 5 supported entities. This web service accepts a map of key-values that will generally contain the name of the annotation being updated and the new value to be stored. At the moment, there are 5 different actions supported by the ***action*** query parameter:

- **ADD:** Default behavior if the query parameter is not provided. Adds the new value to the annotation. If it already existed, the value will be replaced.
- **SET:** ⚠This action might be really harmful. It will set the *annotations* provided in the body of the POST operation and will remove any other *anno tations* the *annotationSet* might have had stored.
- **REPLACE:** Replace the value of an already existing *annotation.* Similar to the ADD action but, in this case, If the annotation did not exist, it will not set the new value !!
- **REMOVE:** Empty the values of some stored *annotations*. To perform this action, the map of the body will need to contain the key 'remove' and a comma separated list containing the annotations to be removed. Example: {"remove": "member.address,member.age"}
- **RESET**: Reset the values of the *annotations* defined to their default values defined in the *variables* of the *variableSet*. To perform this actiont, he map of the body will need to contain the key 'reset' and a comma separated list containing the annotations to be reset. Example: {"reset": "member.address"}

Querying by clinical data

The 5 supported entities mentioned above have their own */search* web service. Among all the different unique fields those entities can be queried for, there is an additional *annotation* field to perform queries over the clinical data stored. There are mainly three different kinds of filtering that can be performed:

- Filter by ***variableSet***: Users might want to filter all the entities that have been *annotated* (have values) using one user-defined *variableSet*. * Follow Filtering by variableSet and annotationSet section defined below to see the supported operations.
- Filter by ***annotationSet****: Users might want to filter all the entities that have been annotated* (have values) for one particular *annotationSet.* * Follow Filtering by variableSet and annotationSet section defined below to see the supported operations.
- Filter by ***annotation***: Users are also allowed to filter by any of the clinical data values. Example:

  Let's imagine that for the above described *Individual* data model, we want to look for any *Individual* whose gender has been defined as FEMALE, older than 30 and living in London. To do this query, we would need to write something like:

  ```
  1. annotation: individual_private_details:age>30;individual_private_details:gender=FEMALE;
  individual_private_details:address.city=London
  or
  2. annotation: age>30;gender=FEMALE;address.city=London
  ```

  - The first option to search, though longer, should never fail as long as there exist a *variableSet* in the *study* containing the *variables* that are being queried. Basically, we are telling OpenCGA to look for any *Individual* matching those values but, at the same time, we are giving OpenCGA information of where the *variables* the user want to look for have been defined (the *variableSet* that defines those *variables*). A general way of seeing this query would have the following format: **[[{annotationSetId}@]{*variableSetId*}:]{*variable*}{*operator*}{*value*}**, where operator can be any of =, == or != for any data type, plus >, >=, <, <= for numeric variables.
  - However, OpenCGA also allows performing the query using the shorter way as seen in the second line in which users can omit specifying the *variableSet* where the *variables* were defined. In this case, OpenCGA will look for all the *VariableSets* that might have defined these *var iables* and, as long as those variables have only been defined in one *VariableSet*, the query will be performed. Otherwise, OpenCGA will raise an error because it will not know the real scope of the query.

Filtering by *variableSet* and *annotationSet*

Filtering by any of these fields can be a bit tricky depending on the amount of *annotationSets* stored for a particular entry. This can be better explained with the following example. Let's say we have only 4 *Individuals* stored in OpenCGA, and they contain the following *annotationSets,* each A, B, C and D corresponding to different *variableSets* A, B, C and D respectively.

Individual 1 :	{ A, B }
Individual 2 :	{ B }
Individual 3:	{ C, D }
Individual 4:	{ }

In this case, the operators =, == and != are also supported, though they might give unexpected results to the user. For this reason, we have also added === and !== operators to support any possible query operation. An example containing the results that would be obtained is shown in the table below:

Operator	Value looked for	Individuals returned	Explanation
=, ==	B	1, 2	Fetch all the individuals containing *annotationSet* or *variableSet* B
===	B	2	Fetch all the individuals that only contains *annotationSet* or *variableSet* B
!=	B	1, 3, 4	Fetch all the individuals that doesn't only contain *annotationSet* or *variableSet* B. Individuals containing B plus any other *annotationSet* or *variableSet* will be returned.
!==	B	3, 4	Fetch all the individuals that have never been annotated using *annotationSet* or *variableSet* B.

Project the *annotation* fields to return

Annotations, as well as any other field from the data models can be included or excluded from the final JSON the user will get. However, because *annotations* contain custom data models that are not completely under OpenCGA's control, a set of reserved prefixes have been defined as explained below:

- Include/exclude specific **annotations**: If we need to project some specific annotations only, users will need to add the prefixes "annotationSets.annotations" or "annotation" to the field to be projected. Example: If after running a query we only want to include the full_name and the hpo variables defined in the Individual VariableSet, users will need to write

```
include: annotation.full_name,annotation.hpo
```

or

```
include: annotationSets.annotation.full_name,annotationSets.annotation.hpo
```

- Include/exclude specific **annotationSets**: Let's imagine that we have several *annotationSets* defined such as in the examples of Individual1 and Individual3. If we only want to project the annotations of one specific *annotationSet*, users will need to use the prefixes "annotationSets.id" or "annotationSet" to the *annotationSet* id to be projected. Example: To include only the *annotations* from the *annotationSets* B and D, we will need to write:

```
include: annotationSets.id.B,annotationSets.id.D
```

or

```
include: annotationSet.B,annotationSet.D
```

- Include/exclude specific **variableSets**: Let's say that for some entries the user have created several *annotationSets* using the same *variableSet* and the user wants to fetch only those instead of getting other *annotationSets*. To do so, users will need to use the prefixes "annotationSets.variableSetId" or "variableSet". Example: Let's imagine that we have another Individual that contains 2 *annotationSets* (a and b) using the template defined in the *variableSet* X and another *annotationSet* (c) annotating the *variableSet* Y. If the user is only interested in getting the annotationSets "a" and "b", we will need to write:

```
include: annotationSets.variableSetId.X
```

or

```
include: variableSet.B
```

Flatten annotations

Additionally, the different */info* and */search* web services have a new query parameter called *flattenAnnotations*. That field is a simple boolean to indicate whether the *annotations* should be returned flattened or not. Let's imagine we have the following annotationSet:

```
{
  "id": "annotation_set_id",
  "variableSetId": "individual_private_details",
  "annotations": {
    "full_name": "John Smith",
    "age": 60,
    "gender": "MALE",
    "address": {
                "city": "United States",
        "zip": "99501"
    }
  }
}
```

The same result with *flattenAnnotations* set to true would be:

```
{
  "id": "annotation_set_id",
  "variableSetId": "individual_private_details",
  "annotations": {
    "full_name": "John Smith",
    "age": 60,
    "gender": "MALE",
    "address.city": "United States",
    "address.zip": "99501"
  }
}
```

GroupBy

- We can put something like the following the 'fields' field: annotation:29:pedigreeAnnotation:Population to group by Population