Data as a Service(DaaS) APIs Technical Documentation

Innoplexus exposes its data ocean of Life Science as set of web APIs which can be consumed on demand basis over an HTTP(s) network. Following is detailed document explaining it's technical aspects.

NB: All APIs are RESTful in nature, and general REST terminologies are used in this document keeping generic sense in mind. More information about REST terminologies can be found here

Common API Features

- All APIs are resource-based and RESTful, where each resource class is a *Noun* and its documents are the *Resources*
- A resource class is an isolated collection of life sciences data. Example: Publication, Congress, Clinical Trial. List of resource classes supported till date can be found here.
- No support for cross resource class queries.
- HTTP Verb is always GET. Request input and context is built as a combination of various query parameters available for the given API
- All APIs are secured by `api_key` which is sent by query parameter
- All the APIs return the response in JSON format
- A positive response comes with a `200 OK` status code and resultant documents in a key named `data`
- A negative response comes with a `4xx/5xx` status code and error details in a key named `error`; error document contains a key `message` with human-readable error message

Resource Class Specific APIs

All the resource classes functions are categorised under 5 major buckets as mentioned below:

- Fetch resource by ID a.k.a **Resource View**
- Fetch paginated list of resources against filters a.k.a. **Index View**
- Simple Text Search Query (along with filters) a.k.a **Index View** with query
- Fetch count of resources with filters and/or query a.k.a **Index View Count**
- Aggregation (on filtered and queried data set) a.k.a **Aggregation**

Following sections will describe each API in details. An example of a positive and negative query will be given for `congress` resource class. For other resource classes, just replace the text for the respective resource class.

Resource View

An API to fetch a single resource.

API Structure:

`api.innoplexus.de/api/v1/{{resource-class}}/resource/{{primary-key-id}}`

Request Input:

- **`primary-key-id:`** A string which is the Primary Key ID/Unique ID of the required document in the respective resource class. The primary key is a configurable property of an resource class. It defaults to `innoplexus_id`
- **`fields:`** A list of strings which represent the attributes of document to be projected. List of fields for each resource class is described in separate sections below

Response:

- Positive:
    - Status Code: 200
    - Body: Requested document with projected fields
- Negative:
    - Status Code: 404
    - Body: Message saying "Resource Not Found"

Example:

Fetch congress with primary key ID "`9e4fd2c6284266029a70bc41cc2ef33d776f849ae7decc0a8c2e7324`"

Request URL:

```
api.innoplexus.de/api/v1/congress/resource/9e4fd2c6284266029a70bc41cc2ef33d776f849ae7decc0a8c2e7324?api
_key={{api_key}}
```

Response: `200 OK`

```
{
    "data": {
        "innoplexus_id": "9e4fd2c6284266029a70bc41cc2ef33d776f849ae7decc0a8c2e7324",
        "created_at": 1410201000,
        "last_updated": 1509714268,
        "abstract": "Abstract Unavailable",
        "congress_name": "International Conference and Exhibition on Neurology and Therapeutics",
        "congress_date": "2014-09-09",
        "title": "Systemic Varicella Zoster Infection Causing Cerebral Venous Thrombosis and
Revealing Prothrombotic State",
        "congress_id": "fbe07f493855151a@#@eab5f840cf59cc04",
        "source_url": "http://www.neurologyconference.com/2014/scientific-program/",
        "congress_type": "",
        "session_type": "",
        "source": "month_wise_data",
        "authors": [
            {
                "normalize_name": "Ayman Mahmoud Alboudi",
                "source_url": "",
                "weightage_logic": "",
                "kol_name": "Ayman Mahmoud Alboudi",
                "affiliation": [
                    "Rashid Hospital"
                ],
                "designation": [],
                "country": "",
                "kol_title": [],
                "kol_education": [],
                "author_id": "206db48cc1f8497da5bc6fac6461c66d",
                "address": "",
                "email": [],
                "phone_fax": []
            }
        ],
        "congress_venue": {
            "city": "Philadelphia",
            "country": "USA"
        },
        "classification": [],
        "highlighted_keywords": []
    }
}
```

Index View

An API to fetch multiple resources of a given resource class, which can be filtered by passing a variety of query parameters. This API fetches paginated list of documents.

API Structure:

`{{host}}/api/v1/{{resource-class}}/`

Possible Query Params(Request Inputs):

1. **`size`**: This is an integer value which mentions the total number of documents per page. The number defaults to 10.

2. **`page`**: The page number of the result set to be displayed. The default value is 1.
3. **`filters`**: List of clauses connected with `AND/OR.` Ref Filter Syntax Description
4. **`query`**: Text value specifying a search phrase
5. **`sort`**: List of fields followed by ASC/DESC for sorting values by ascending or descending respectively.
6. **`fields:`** A list of strings which represent the attributes of document to be projected

Response:

- Positive**:**
  - Status Code: 200
  - Body:
    - **`documents:`** List of resulting documents with size as mentioned in request.
    - **`total:`** Total number of documents matching the query
    - **`next_page`**: Link to next page of results
    - **`prev_page`**: Link to previous page of results

- Negative:
  - Status Code:
    - **400**: A malformed request - syntax/semantic error etc, wrong query parameters, typo errors etc.
    - **422**: An unprocessable request, request might be as per specifications
    - **401**: Unauthorized request (wrong or expired API Key)
    - **5xx**: Server side error, please try after some-time; inform Devs if error persists
  - Body:
    - Standard error response, with possible indications of syntax error in filter grammar

Possible Functionalities:

- Filter clause
  - A filter clause is a comparison of a attribute(s) against value(s)
  - Supported comparison operators:
    - Equals (**`EQ`**): Match a single attribute against single value
    - Not Equals (**NE**): Negative match
    - Range (**`GT/GTE/LT/LTE`**): If the value lies within a given range
    - Exists (**`EXISTS`**): Is the given attribute exists in the resource
    - Multi Value Match (**`IN`**): Single Attribute, multiple values
    - Prefix (**`STARTS_WITH`**): If the value starts with a given string
    - Word Prefix (**`WORD_STARTS_WITH`**): If words in a textual value starts with a given string
  - Clauses can be connected with logical operators -
    - `AND`
    - `OR`

- Clause can be prioritised using round braces i.e “`()`”
- **`Boost:`** `Filters can be appended with “boost <float>” for e.g.`
    - `(x eq “y” boost 3.0 or a eq “b” boost 4.0) this will alter the order in which document appears according to their boost score.`
- **`Inner_projection:`** `Similarly filter can also be appended with inner_projection for e.g.`
    - `X eq “y*” inner_projec, this will mach regular-expression for y rather that match to string “y*”.`
    - `X eq “y*” or a eq “*b”`

- Only text search
    - A search query is also possible by passing a text containing few phrases.
    - Our proprietary **NLTK-Parser** *(Natural Language Toolkit)* service breaks them down into concepts and performs search on relevant fields
    - Phrases should be connected by `‘AND’` and negated by `‘NOT’`
    - This is useful when the user has no knowledge of attributes of an resource class

- Order of sorting
    - Order of resultant documents can be influenced by mentioning which fields to sort on
    - A search query can be re-ordered based on some custom functions, to alter the score of a matching document
    - For `publications`, we add a field factor on `` `impact_factor` ``, for others we add a field factor on `` `created_at` ``

Example:

Fetch list of congress projecting attributes `` `innoplexus_id` ``, `` `congress_name` `` and `` `source` `` whose source are `` `web_pdf` ``

Request URL:

`api.inoplexus.de/api/v1/congress/?api_key={{api_key}}&filters=source EQ "web_pdf"&fields=innoplexus_id,congress_name,source`

Response: `200 OK`

```
{
    "data": {
        "documents": [
            {
                "innoplexus_id": "60f6c338c35c1632fe9fe4e0299251767a4cd9ac82cbf52cb3f6220a",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "145372c512275648526eceec0c21920497f46a71c211d9b01ed764e6",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "0ad7f50291be6046bb3f2027fd003359edc65749640359d45567ce6e",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "4416157deb118699e6c388985d76d4efbd6f1f8d61238cf96473732f",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "5a27e8fe33e08ffe4dbd80666829947c62880e3d12d02bd22782d264",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "e0e089db73da02ad3dcd34db503dbd351db008fc9a2646d09c716054",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "206d47d53457b77443feea9d638e4b12e2128f8862d17ff95ef2d44a",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "5fcf372ed2cc0ac34e1536b7cd419e3df5db096afb6607a8aa735a1d",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "5f7603fbaea40d535b84c297ffcd825a9c28ce4532568ce7259a539d",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            },
            {
                "innoplexus_id": "74f0670c10b7df0a3de5910d6765f2c136e4154d46bd5b00280e393c",
                "congress_name": "British Thoracic Society Annual Winter Meeting",
                "source": "web_pdf"
            }
        ],
        "total": 263861,
        "next_page":
"api.innoplexus.de/api/v1/congress/?api_key={{api_key}}&filters=source%20EQ%20%22web_pdf%22&fields=in
noplexus_id,congress_name,source&page=2&size=10",
        "prev_page": null
    }
}
```

Index View Count

An API to the count of results from a index view filter query.

API Structure:

`api.innoplexus.de/api/v1/{{resource-class}}/count`

Request:

Same as index view

Response:

`count` in **`data`** containing integer value

Aggregation

An API to aggregate metric or create buckets of data, and inner buckets of data points. This functionality can be performed on filtered data set as obtained from Index view. Nesting of buckets is limited till 3rd level

API Structure:

`api.innoplexus.de/api/v1/{{resource-class}}/`

Possible Query Params(Request Inputs):

1. **`filters`** and **`query`** as mentioned in Index view for limiting scope of aggregations
2. **`aggs`**: Mention the aggregation query. Aggregation query language is explained here

Response:

- Positive**:**
  - Status Code: 200
  - Body:
    - **`` `aggregation_name`: ``** List of resultant documents with size as mentioned in request

        - **`total_count`:** Total number of resources on which the aggregation was performed
        - **`bucket_response`**: Contains list of buckets in a key named `buckets`, which can contain nested aggregation buckets or metrics
            - **`key`:** Name of bucket/Value of aggregated data point
            - **`key_as_string`:** A readable name for integers, like date epochs
            - **`doc_count`:** Count of resources aggregated in the bucket
            - **`inner_aggregation_response`:** A bucket nested inside the given one. Has similar structure.
        - **`metric_response`**: Contains a metric value aggregated on a numerical field.
            - **`value`**: Aggregated value
- Negative:
    - Status Code:
        - 400: For a malformed request
        - 401: Unauthorized request (wrong or expired API Key)
        - 5xx: Server side error, please try after some-time; inform Devs if error persists
    - Body:
        - Standard error response, with possible indications of syntax error in `aggs` grammar

Possible Functionalities:

Aggregations can be broadly divided in two types, each having multiple subtypes

- Bucket:
    - Grouping documents into buckets based on values of a given attributes.
    - Supported sub-types of bucket aggregations are:
        - **Terms**: Create buckets of distinct values of a given attribute (field). This makes sense when applied on non textual fields, which have definite values.
        - **Significant Terms:** Creating buckets and selecting the ones which have a major change than the usual doc count.
        - **Histogram:** Creates buckets on a numerical field, where the values of the buckets are calculated on the run time.
        - **Date Histogram:** Creates histogram on epoch field.
    - Only a bucket type aggregation supports further nesting. A nested aggregation works on the scope of a parent aggregation.

- Metric:
    - Analyse documents basis any numeric type field and fetch metrics like count, avg, max, min etc
    - Restricted to numeric fields
    - Supported sub-types of metric aggregation:
        - `Min`
        - `Max`
        - `Sum`
        - `Avg`

Filter Syntax Description

	Category	Description	Syntax Description	Simple Example
1	Boolean clauses	We support "and", "or" & "not" in our filter: 1. And & Or are binary operators used in infix notation. 2. Not is a unary operator used in a prefix notation. 3. A Clause can also be separated by a pair of parenthesis. 4. Or a clause can be an operation-expression (see row below for more clear description) 5. Boost operator is used to multiply boost factor of the clause it is applied to, by default boost factor is 1.0 for all clause. 6. Inner projection: Using these fields will generate a larger query over all of fields's property in focus. **Notice:**Applying more than 1 field-properties will result in application of the property most closer to the attributes	1. AND: <Clause> AND <Clause> 2. Or: <Clause> OR <Clause> 3. Not: NOT <Clause> 4. Parenthesis: (<Clause>) 5. <Clause> BOOST <float>, here <float> is a float-number like 1.0,2.1,3.2 etc. 6. <Boolean clause> using <field_properties> disambiguation: <Boolean clause> using exact using summay is same as <Boolean clause> using exact	1. x eq "a" **and** y eq "b" 2. x gt "a" **or** y lt "b" 3. **not** x start_swith "a" 4. **(** x eq "a" **)** 5. x in ["a"] **boost 3.0** 6. a **using synonyms** Here x,y is attributes of data-schema and a,b are filter-values.
2	Comparison expression (Clause)	An operation is a basic comparison/querying operation	An operation expression is of form: 1. <key> <op_string> <value> 2. <key> <op_array> <value>	Op_string: 1. x **eq** "1" 2. name **starts_with** "man" Op_array:

				1. x **in** ["a", "b", "c"]
3.	field-properties	There are 3 field-properties: synonyms, raw and summary.	It goes as follows: 1. <Boolean clause> using (exact | summary | synonyms)	1. x eq "some_thing" **using exact** 2. (x eq "some_thing" and y lt "other_thing") **using summary** (x eq "some_thing" **using synonyms**) using exact
4.	Key	A key is a string of alphanumeric characters. If key = all, then the corresponding action of key will be applied to all fields.	In regular expression : [a-zA-Z0-9_.]+	1. like_this 2. int2String 3. Phase
5	value	A value is a string of character inside double-quotes("). Any double-quotes inside the string should come with escape character i.e. \"		1. "like this" 2. "\" example of string\" inside a string"
6	op_string	These are set of operation which take a single key and value as arguments. These are case insensitive.	1. Eq : Equal to 2. Ne : Not equal to 3. Gte: Greater than or equal to 4. Gt : Greater than 5. Lt : Less than 6. Lte: Less than or equal to 7. exist 8. starts_with 9. word_starts_with	Example of these occurring in expressions: 1. x **eq** "y" 2. x **ne** "y" 3. x **gte** "y" 4. x **gt** "y" 5. x **lt** "y" 6. x **lte** "y" 7. x **exists** " y" 8. x **starts_with** "y" 9. x **word_starts_with** "y"

7.	op_array	Are operation which have different argument specification.	In operator: <key> in <array>	1. x **in** ["a", "b", "c"]
8.	array	An array is a comma separated list of values inside box brackets.		["a", "b", "c"]

Aggregation Syntax Description

Notice: Some of the terms will be used from the above table.

"?" after a syntax term denotes optional syntax term.

	Category	Description	Syntax Description	Simple Description
1	Aggregation	The syntax of aggregation is a list of pattern as in the right column where at the end of expression we can add another such expression implying a nested query.	Group_by <aggregation_type> <aggregation_field> <Parameter_list> <aggregation>?	1. group_by term some.Field 2. group_by histogram phase limit "5" group_by avg some_field
2	Aggregation_type	*Bucket aggregations* are used to cluster/aggregate data points according to values. *Metric aggregation:* The aggregations in this family compute metrics based on values extracted in one way or another from the documents that are being aggregated.	*Bucket Aggregation:* 1. term 2. significant_term 3. histogram 4. date_histogram *Metric Aggregation:* 1. sum 2. avg 3. min 4. max 5. stats	1. term 2. significant_term 3. histogram 4. date_histogram 5. sum 6. avg 7. min 8. max 9. stats

3	Aggregation_field	A string containing Alpha-numeric characters along with “_”and “.”, is the name of field we want to aggregate.	Syntactically it is same as <key>.	1. like_this 2. int2String 3. Phase
4	Parameter_list	It is a space separated list of Parameter operations.	<Parameter_operation>* Here "*" (is kleene closure) is zero or more occurrence of parameter_operation.	1. having ["a", "b", "c"] missing "asd" limit "10" with interval eq "some_range_fo rmat"

5	Parameter_operation	List of operations available for aggregation: 1. With : with is used to set some special restricted parameters. 2. Having : Passes for data-points which have the values from input array. 3. not_having : Passes for data-points which does not have the values from input array. 4. Limit: Is the size of bucket. 5. Order_by: For sorting bucket. 6. missing: This creates bucket of all documents in the current document that are missing a field value. 7. using (...) : applies field-property to aggregation fields for e.g. group_by term x using synonyms, will aggregate on	<value> & <array> is same as above table. Fix keys : 1. With <Restricted_parameter_list> 2. Having <array> 3. Not_having <array> 4. Order_by <name> <asc|desc>?: *Disambiguation:* asc: ascending order desc: descending order If not provided by default we take desc as order. 5. Missing <value> 6. Limit <value> 7. USING <field_properties>	1. **with** min_doc_count eq "0" 2. **having** ["a", "b", "c"] 3. **not_having** ["a", "b", "c"] 4. **order_by** publish_date asc 5. **missing** "asd" 6. **limit** "10" 7. **using** exact

		all synonyms of x.		
6	Restricted_parameter_list	List of restricted parameters.	<restricted_keys> eq <value>	1. min_doc_count eq "2" 2. interval eq "yyyy" 3. format eq "asd" 4. chi_square eq "true"
7	Restricted_keys	Type: 1. min_doc_count : integer 2. interval: string 3. format : string 4. chi_square: boolean (true/false)	1. min_doc_count 2. interval 3. format 4. chi_square	1. min_doc_count 2. interval 3. format 4. chi_square

Resource Classes

The following is the list of resource class supported through Innoplexus DaaS. The name goes exactly into the `{{resource-class}}` placeholder of all resource class specific APIs. The schema of each resource class will be discussed in later sections.

Name
`publication`
`congress`
`guideline`
`drug`
`clinical_trial`
`author`

Each resource class' schema will be discussed in details in later sections.
Notes:

1. Each document has these default fields:
    a. `innoplexus_id` : Default primary key field basis each resource class. Every document will have this field by default. If a custom field is not specified as primary key field, then this ID is the primary key field.
    b. `created_at` : Epoch timestamp indicating when the actual resource, i.e. the publication or congress was published/created.
    c. `last_updated` : Epoch timestamp indicating when the resource data was last updated in our database

Resource Agnostic APIs

KOL Library

- *API for scoring Key Opinion leaders (KOL) on specific criteria.*
- KOL can be of following types :
    - Pharma_top : Scoring for these KOL is done on the basis of its presence across different resources and includes all the historical data
    - Pharma_emerging : Scoring for these KOL is done on the basis of its presence in Publication, Clinical trials, and Congresses and includes the data of the past three years

        - Financial_top : Scoring for these KOL is done on the basis of its presence in Financial resources.

API Structure:

`qa.api.innoplexus.de/api/v1/kols/score?`

Response:

- Positive**:**
    - Status Code: 200
    - Body:
        - **`author_id`** `: String id for a particular kol`
        - **`score`** `: Score of the KOL`
        - **`score_profiles`**
            - **`asset_class`** `: Resource for which scoring is done`
            - **`asset_class_score`** `: Score for that Resource`
            - **`doc_count`** `: Number of document on which score is computed by the algorithm`

- Negative:
    - Status Code:
        - 400: For a malformed request
        - 401: Unauthorized request (wrong or expired API Key)
        - 5xx: Server side error, please try after some-time; inform Devs if error persists
    - Body:
        - Standard error response, with possible indications of syntax error in filter grammar

Possible functionalities **:**

- Filters
    - This param defines the filters that are to be applied while scoring
    - Pattern for a single filter is <filter_name> <comparison op> <filter_values> Eg. `indication EQ "Multiple Sclerosis"`
    - Pattern for multiple filters is `filter_string_1` and `filter_string_2`..... Eg. `indication EQ "Multiple Sclerosis" and ta EQ "Neurology"`
    - Supported comparison operators:
        - Equals (**`EQ`**): Match a single attribute against single value
        - Range (**`GT/GTE/LT/LTE`**): If the value lies within a given range
    - Clauses can be connected with logical operators -
        - `AND`
        - `OR`
    - Clause can be prioritised using round braces i.e "**`()`**"

Request URL :

"https://qa.api.innoplexus.de/api/v1/kols/score?api_key={{api_key}}&type=pharma_top&asset_classes=publication&weightages=publication%20EQ%20%223.40%22&is_normalize=true&is_sort=true&is_enable_cache=true&size=10"

S no	Query Parameter	Description	Syntax Description	Simple Example
1.	type	This is type of opinion leader to be considered for scoring	Following are list of different type of KOL :- ● pharma_top ● pharma_emerging ● financial_top	`1.type =`pharma_top `2. type=financial_top`

2	filters	List of clauses connected with operators	This is following list of operators :
			● EQ
			● GTE
			● LTE
			● IN
			● GT
			● LT
				`1. ‘x’ in [‘a’,’b’]`
				`2. ‘x’ eq ‘y’`
				`3. ‘x’ gte ‘y’`
				`4. ‘x’ lte ‘y’`
				`5. ‘x’ gt ‘y’`
				`6. ‘x’ lt ‘y’`

3.	weightage	This param is used to put more weight on a resource while computing KOL score. (** max limit for any resource is 100)	Publication eq “50” and Clinical_trials eq “25” and Congress eq “25”	
4.	is_normalize	Is a boolean string specified weather to include normalizing factor or not.	● is_normalize=True	`is_normalize=True || false`
5.	is_sort	To sort the score in ascending or descending order	● is_sort=true i.e. order is descending ● ● is_sort=false i.e. order is ascending	
6.	Size	This is an integer value which mentions the total number of documents per page. The number defaults to 10		

For more details or feedback please write to us at daas@innoplexus.com