nidm Documentation

Release 1.0

NIDASH Working Group

November 05, 2015

Contents

This is a tool to deploy a REST API to run queries on and visualize NIDM turtle objects. Queries can be found in the nidm-query, repo, along with a nidm viewer. This API is under development, and please submit issues and requests to the nidm-api repo.

CHAPTER 1

Why do I want to use this?

You might want to use this tool if you have a NIDM data structure, meaning NIDM-results, NIDM-experiment, or NIDM-workflow, and you want to get information out of it but you don't know a single thing about RDF files or the query language for them, which is called sparql. This tool will allow you to run pre-generated queries easily, and return results in a format that is easily parsable by modern web technologies (eg, json for javascript or python), and (coming soon) csv and tsv files.

CHAPTER 2

Under Development

The tool currently implements returning basic json from a query against a ttl (turtle) file. The following will be developed:

- interactive interfaces for creating new queries
- returning interactive graphs (d3, neo4j)
- functions to search / filter queries
- returning more data types
- validation of query data structures

Contents:

2.1 Installation

To install

```
pip install git+git://github.com/incf-nidash/nidm-api.git
```

2.2 Getting Started

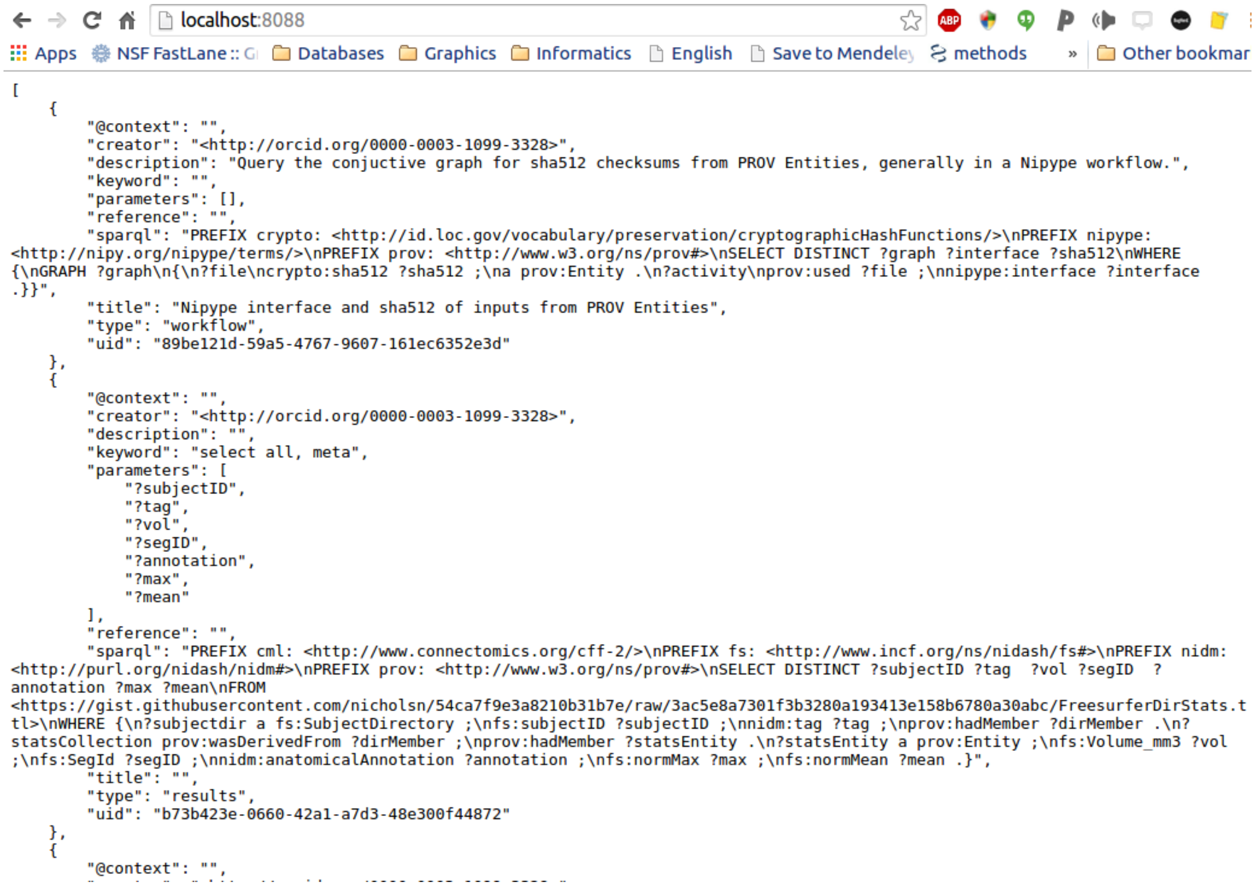
You have three options for using the nidm-api: as a local REST API, a REST API (served to the world on a public server), or as modules (to integrate into your python applications).

2.2.1 Local Machine REST API

When you install the module, an executable, “nidm” is placed into your system or local bin (it will tell you the location upon installation). If this bin is added to your path, you can start the server:

```
nidm
```

Then open your browser to localhost:8088. The index view that you see is a complete list of validated queries, for example:

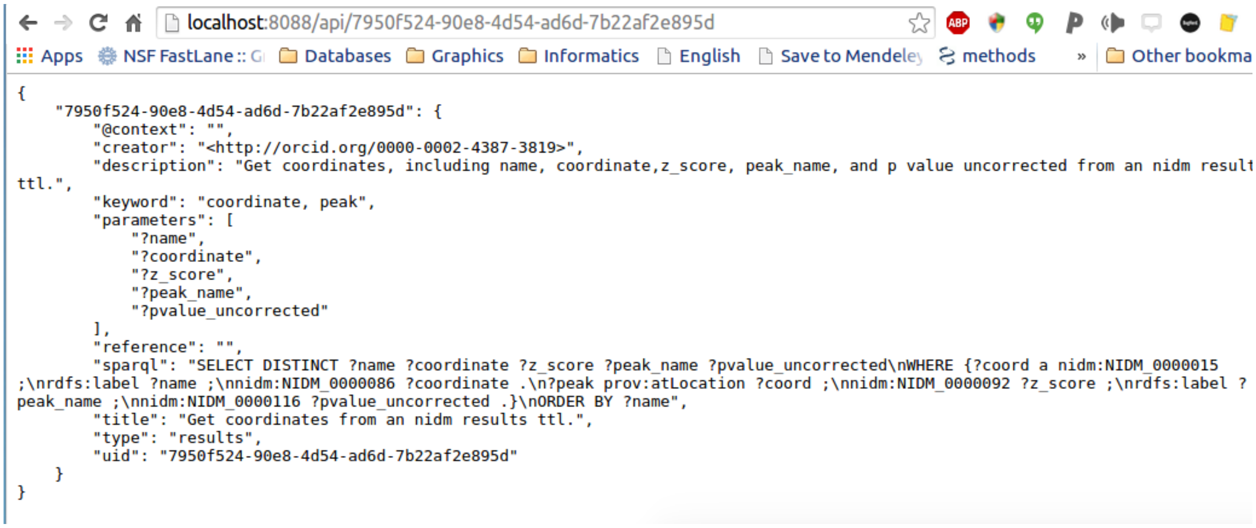


```
[
    {
        "@context": "",
        "creator": "<http://orcid.org/0000-0003-1099-3328>",
        "description": "Query the conjuctive graph for sha512 checksums from PROV Entities, generally in a Nipype workflow.",
        "keyword": "",
        "parameters": [],
        "reference": "",
        "sparql": "PREFIX crypto: <http://id.loc.gov/vocabulary/preservation/cryptographicHashFunctions/>\nPREFIX nipype:
<http://nipy.org/nipype/terms/>\nPREFIX prov: <http://www.w3.org/ns/prov#>\nSELECT DISTINCT ?graph ?interface ?sha512\nWHERE
{\nGRAPH ?graph\n{\n?file\ncrypto:sha512 ?sha512 ;\na prov:Entity .\n?activity\nprov:used ?file ;\nnipype:interface ?interface
.}}",
        "title": "Nipype interface and sha512 of inputs from PROV Entities",
        "type": "workflow",
        "uid": "89be121d-59a5-4767-9607-161ec6352e3d"
    },
    {
        "@context": "",
        "creator": "<http://orcid.org/0000-0003-1099-3328>",
        "description": "",
        "keyword": "select all, meta",
        "parameters": [
            "?subjectID",
            "?tag",
            "?vol",
            "?segID",
            "?annotation",
            "?max",
            "?mean"
        ],
        "reference": "",
        "sparql": "PREFIX cml: <http://www.connectomics.org/cff-2/>\nPREFIX fs: <http://www.incf.org/ns/nidash/fs#>\nPREFIX nidm:
<http://purl.org/nidash/nidm#>\nPREFIX prov: <http://www.w3.org/ns/prov#>\nSELECT DISTINCT ?subjectID ?tag  ?vol ?segID  ?
annotation ?max ?mean\nFROM
<https://gist.githubusercontent.com/nicholsn/54ca7f9e3a8210b31b7e/raw/3ac5e8a7301f3b3280a193413e158b6780a30abc/FreesurferDirStats.t
tl>\nWHERE {\n?subjectdir a fs:SubjectDirectory ;\nfs:subjectID ?subjectID ;\nnidm:tag ?tag ;\nprov:hadMember ?dirMember .\n?
statsCollection prov:wasDerivedFrom ?dirMember ;\nprov:hadMember ?statsEntity .\n?statsEntity a prov:Entity ;\nfs:Volume_mm3 ?vol
;\nfs:SegId ?segID ;\nnidm:anatomicalAnnotation ?annotation ;\nfs:normMax ?max ;\nfs:normMean ?mean .}",
        "title": "",
        "type": "results",
        "uid": "b73b423e-0660-42a1-a7d3-48e300f44872"
    },
    {
        "@context": "",
```

The available queries must pass through validation to be available (not yet implemented). The queries are organized by their uid, which is just the name of the json file that is found in the nidm-query repo. You can contribute to this repo if you want to make a new query, and tools will be developed for you to generate these data structures in a graphical interface. The "type" variable in the returned json is generated dynamically, and corresponds to the folder name in nidm-query repo. We currently support "results," "experiment," and "workflow," as these are the different kinds of NIDM data structures that are being developed.

The first thing you might want to do is retrieve all the meta data for a single query. This means that we will look at the list in the photo above, and find the uid of one that we like. We can then ask to see a single query:

```
http://localhost:8088/api/7950f524-90e8-4d54-ad6d-7b22af2e895d
```

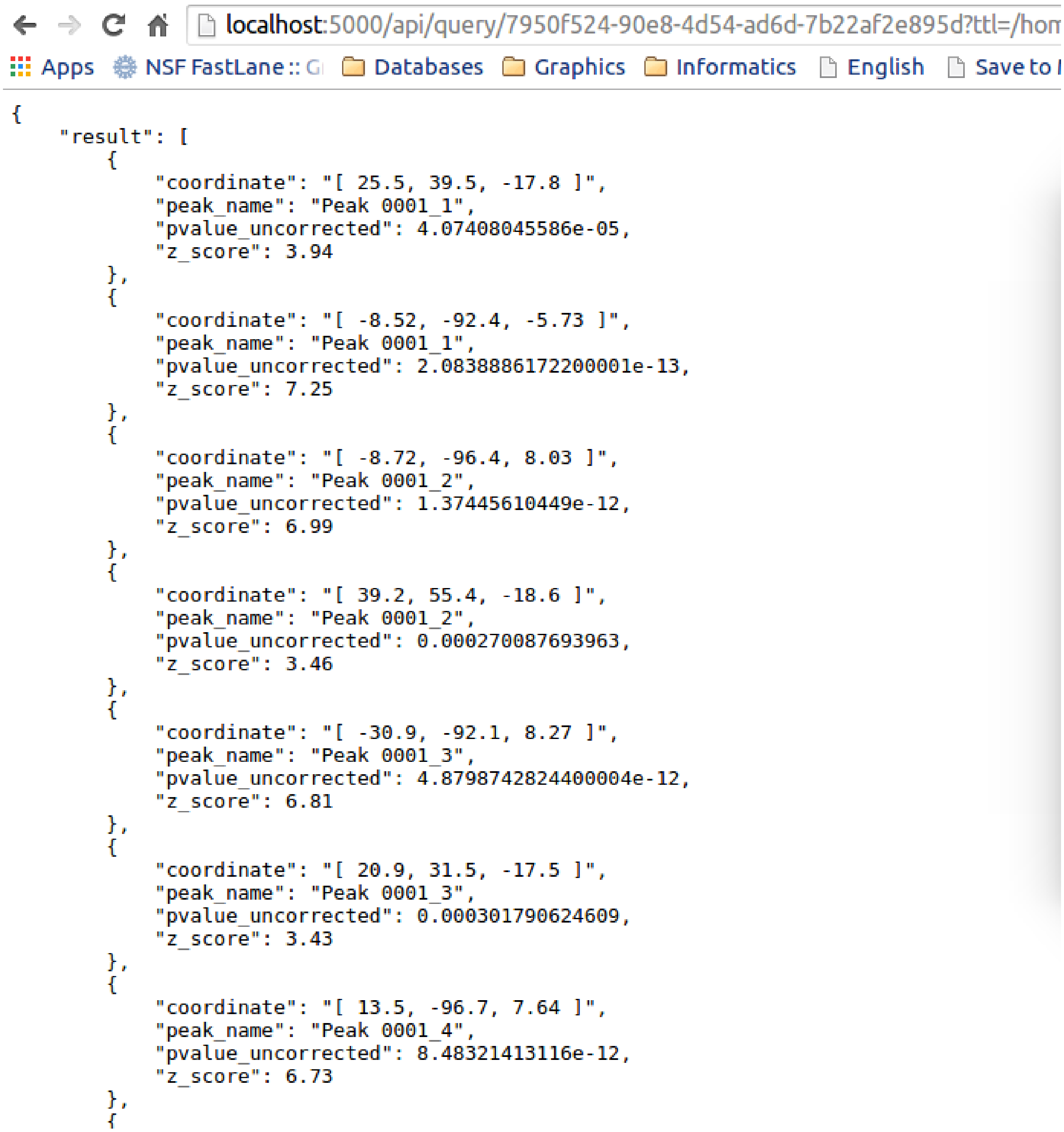


```
{
    "7950f524-90e8-4d54-ad6d-7b22af2e895d": {
        "@context": "",
        "creator": "<http://orcid.org/0000-0002-4387-3819>",
        "description": "Get coordinates, including name, coordinate,z_score, peak_name, and p value uncorrected from an nidm result
ttl.",
        "keyword": "coordinate, peak",
        "parameters": [
            "?name",
            "?coordinate",
            "?z_score",
            "?peak_name",
            "?pvalue_uncorrected"
        ],
        "reference": "",
        "sparql": "SELECT DISTINCT ?name ?coordinate ?z_score ?peak_name ?pvalue_uncorrected\nWHERE {?coord a nidm:NIDM_0000015
;\nrdfs:label ?name ;\nnidm:NIDM_0000086 ?coordinate .\n?peak prov:atLocation ?coord ;\nnidm:NIDM_0000092 ?z_score ;\nrdfs:label ?
peak_name ;\nnidm:NIDM_0000116 ?pvalue_uncorrected .}\nORDER BY ?name",
        "title": "Get coordinates from an nidm results ttl.",
        "type": "results",
        "uid": "7950f524-90e8-4d54-ad6d-7b22af2e895d"
    }
}
```

You might then have a turtle file that you want to actually run a query on. For example, if you look in the nidm-api examples directory, we have provided a nidm.ttl file that corresponds to a NIDM-Results folder. We can run the query we just saw above to get coordinates:

```
http://localhost:8088/api/query/7950f524-90e8-4d54-ad6d-7b22af2e895d?ttl=/home/vanessa/Desktop/nidm.t
```

The only difference is that we've added the "query" to the URL, and specified the ttl file as a variable, indicated by the "?" at the end of the URL. We will eventually give you more options to customize your query, for now that's it. When you do the query, your result will be returned again in json:


localhost:5000/api/query/7950f524-90e8-4d54-ad6d-7b22af2e895d?ttl=/hom
Apps
NSF FastLane :: G
Databases
Graphics
Informatics
English
Save to I

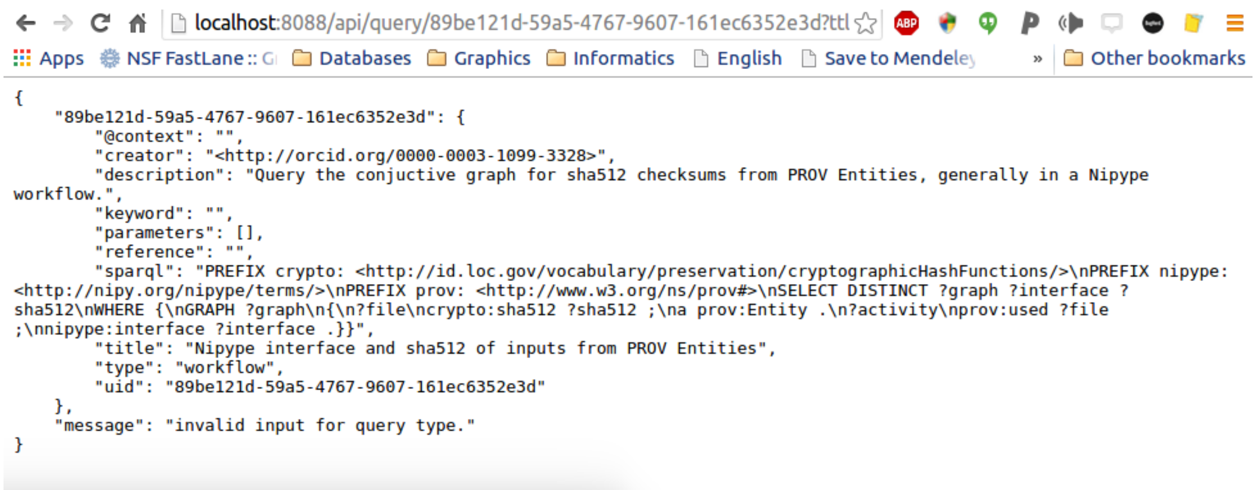
```
{
    "result": [
        {
            "coordinate": "[ 25.5, 39.5, -17.8 ]",
            "peak_name": "Peak 0001_1",
            "pvalue_uncorrected": 4.07408045586e-05,
            "z_score": 3.94
        },
        {
            "coordinate": "[ -8.52, -92.4, -5.73 ]",
            "peak_name": "Peak 0001_1",
            "pvalue_uncorrected": 2.0838886172200001e-13,
            "z_score": 7.25
        },
        {
            "coordinate": "[ -8.72, -96.4, 8.03 ]",
            "peak_name": "Peak 0001_2",
            "pvalue_uncorrected": 1.37445610449e-12,
            "z_score": 6.99
        },
        {
            "coordinate": "[ 39.2, 55.4, -18.6 ]",
            "peak_name": "Peak 0001_2",
            "pvalue_uncorrected": 0.000270087693963,
            "z_score": 3.46
        },
        {
            "coordinate": "[ -30.9, -92.1, 8.27 ]",
            "peak_name": "Peak 0001_3",
            "pvalue_uncorrected": 4.8798742824400004e-12,
            "z_score": 6.81
        },
        {
            "coordinate": "[ 20.9, 31.5, -17.5 ]",
            "peak_name": "Peak 0001_3",
            "pvalue_uncorrected": 0.000301790624609,
            "z_score": 3.43
        },
        {
            "coordinate": "[ 13.5, -96.7, 7.64 ]",
            "peak_name": "Peak 0001_4",
            "pvalue_uncorrected": 8.48321413116e-12,
            "z_score": 6.73
        },
        {
```

Note that I gave the REST API a local path on my computer. We can also give it a URL and it will work.

```
http://localhost:8088/api/query/7950f524-90e8-4d54-ad6d-7b22af2e895d?ttl=https://rawgithub.com/incf-
```

Boum.

If you screw something up, meaning that the query did not work for any reason (e.g., you gave it a wrong file, improperly formatted file, or the query logic has an error), it will tell you:



```
{
    "89be121d-59a5-4767-9607-161ec6352e3d": {
        "@context": "",
        "creator": "<http://orcid.org/0000-0003-1099-3328>",
        "description": "Query the conjuctive graph for sha512 checksums from PROV Entities, generally in a Nipype
workflow.",
        "keyword": "",
        "parameters": [],
        "reference": "",
        "sparql": "PREFIX crypto: <http://id.loc.gov/vocabulary/preservation/cryptographicHashFunctions/>\nPREFIX nipype:
<http://nipy.org/nipype/terms/>\nPREFIX prov: <http://www.w3.org/ns/prov#>\nSELECT DISTINCT ?graph ?interface ?
sha512\nWHERE {\nGRAPH ?graph\n{\n?file\ncrypto:sha512 ?sha512 ;\na prov:Entity .\n?activity\nprov:used ?file
;\nnipype:interface ?interface .}}",
        "title": "Nipype interface and sha512 of inputs from PROV Entities",
        "type": "workflow",
        "uid": "89be121d-59a5-4767-9607-161ec6352e3d"
    },
    "message": "invalid input for query type."
}
```

Note that the current (tiny) set of queries is currently not being validated, and they need work and contribution.

2.2.2 REST API on Server

You can use something like gunicorn to run the flask application on a server, for the world to use. More documentation on how to do this will come, as we currently do not have a server to host this. Please be aware that the debug mode in the Flask application is set to True, and you should **'read about <http://flask.pocoo.org/docs/0.10/deploying/ >'_** the proper way to deploy a flask application before doing something that might compromise the security of your server.

2.2.3 Integration into Python

An example turtle file is provided in the "example" directory of the repo, and running a query on this file from within python is shown below:

```python
#!/usr/bin/python

from nidm.query import get_query_directory, validate_queries, make_lookup, do_query

# Get updated queries, validate, and generate a lookup dict:
query_dir = get_query_directory()
query_json = validate_queries(query_dir)
query_dict = make_lookup(query_json,key_field="uid")

# Let's use the query to get coordinates
qid = "7950f524-90e8-4d54-ad6d-7b22af2e895d"

# Here is a ttl file that I want to query, nidm-results
ttl_file = "nidm.ttl"

result = do_query(ttl_file=ttl_file,query=query_dict[qid]["sparql"])

# The result is a pandas data frame. I can turn it into other things too
result = result.to_dict(orient="records")
```

2.3 nidm

2.3.1 nidm package

Subpackages

nidm.script package

Submodules

nidm.script.post_install module

Module contents

nidm.templates package

Module contents

Submodules

nidm.app module

class `nidm.app.`**`NIDMServer`**`(`**args*, ***kwargs*`)`

Bases: `flask.app.Flask`

class `nidm.app.`**`apiDoQuery`**

Bases: `flask_restful.Resource`

return result of query on ttl file Paramters ========= qid: str

the uid associated with the query

ttl: str the url of the turtle file

`endpoint` **= 'apidoquery'**

`get``(`*qid*, *output_format*`)`

`mediatypes``(`*resource_cls*`)`

`methods` **= ['GET']**

class `nidm.app.`**`apiIndex`**

Bases: `flask_restful.Resource`

Main view for REST API to display all available queries

`endpoint` **= 'apiindex'**

`get``()`

`mediatypes``(`*resource_cls*`)`

`methods` **= ['GET']**

class `nidm.app.`**`apiQueryMeta`**

Bases: `flask_restful.Resource`

return complete meta data for specific query

`endpoint` **= 'apiquerymeta'**

`get` (*qid*)

`mediatypes` (*resource_cls*)

`methods` **= ['GET']**

`nidm.app.`**`generateQuery`**()

`nidm.app.`**`newQuery`**()

`nidm.app.`**`previewQuery`**()

`nidm.app.`**`start`**(*port=8088*)

nidm.experiment module

nidm.query module

query: part of the nidm-api general functions to work with query data structures for nidm-queries

class `nidm.query.`**`Queries`**(*components=['experiment', 'results', 'workflow']*)

`nidm.query.`**`do_query`**(*ttl_file*, *query*, *rdf_format='turtle'*, *serialize_format='csv'*, *output_df=True*)

`nidm.query.`**`download_queries`**(*destination*)

Download queries repo to a destination Parameters ========== destination:

the full path to download the repo to

`nidm.query.`**`find_queries`**(*query_folders*, *search_pattern='*.json'*)

searches one or more folders for valid queries, meaning json files. In the case of multiple directories, will append the folder name as a variable to indicate the type Parameters ========== query_folders: list or str

one or more full paths to directories with json objects

search_pattern: str pattern for glob to use to find query objects default is "*.json"

queries: list a list of full paths to query object files

`nidm.query.`**`format_sparql`**(*sparql_text*)

split sparql text into a list, and extract parameter options from select.

`nidm.query.`**`generate_query_template`**(*output_dir=None*, *template_path=None*, *fields=None*)

output_dir: str full path to output directory for json data structure. if none specified, will not save the data structure

template_path: str path to json file to use as a template. Only should be specified if the user has reason to use a custom template default is the standard provided by nidm-api.

fields: dict (optional) a dictionary with fields that correspond to template keys. if provided, template will be filled with keys. Possible values include

template: json (dict) A python dictionary (json) that can be filled with new query information nidm.query.**get_query_directory**(*tmpdir=None*)

get_query_directory: Download queries repo to tmp directory Parameters ========== tmpdir: str

path to directory to download queries to

nidm.query.**make_lookup**(*query_list*, *key_field*)

returns dict object to quickly look up query based on uid Parameters ========== query_list: list

a list of query (dict objects)

key_field: str the key in the dictionary to base the lookup key

query_dict: dict dict (json) with key as "key_field" from query_list

nidm.query.**read_queries**(*query_paths*)

Read in a list of query (json) objects. Parameters ========== query_paths: list a list of full paths to query objects to read Returns ======= **queries_**: list

dict to be served as json describing queries available a "type" variable is added to indicate folder query was found in

nidm.query.**save_query_template**(*template*, *output_dir*)

generate_query_template Parameters ========== output_dir: string path

full path to output directory for json data structure. the template filename is generated from the uid variable

success: boolean True if save was successful, false otherwise

nidm.query.**validate_queries**(*query_dir, queries=None, components=['sparql']*)

returns json object with query data structures, and a field 'valid' to describe if query was valid Parameters ========== queries: list

a list of full paths to json files, each a query

query_dir: str full path to a nidm-query repo

components: folders to include corresponding to nidm query language (currently only option is sparql)

queries: json dict (json) with all read in queries available

from nidm-query, provided by API

nidm.results module

nidm.scripts module

script.py: part of nidmapi package Runtime executable

nidm.scripts.**main**()

nidm.utils module

utils: part of the nidm-api general functions for the api

nidm.utils.**clean_fields**(*mydict*)

Ensures that keys and values of dictionary are in utf-8 so rendering in javascript is clean. Paramters ========= mydict: dict dictionary to clean

newdict: dict dictionary with all fields encoded in utf-8

`nidm.utils.`**`copy_directory`**(*src*, *dest*)
Copy an entire directory recursively

`nidm.utils.`**`find_directories`**(*root*, *fullpath=True*)
Return directories at one level specified by user (not recursive)

`nidm.utils.`**`find_subdirectories`**(*basepath*)
Return directories (and sub) starting from a base

`nidm.utils.`**`get_installdir`**()
returns installation directory of nidm-api

`nidm.utils.`**`get_query_template`**()
get_standard_template returns the full path to the standard template for queries

`nidm.utils.`**`get_template`**(*template_file*)
get_template: read in and return a template file

`nidm.utils.`**`has_internet_connectivity`**()
Checks for internet connectivity by way of trying to retrieve google IP address. Returns True/False

`nidm.utils.`**`is_type`**(*var*, *types=[<type 'int'>, <type 'float'>, <type 'list'>]*)
Check type

`nidm.utils.`**`load_json`**(*json_path*)
returns a loaded json file Parameters ========== json_path: str

full path to json file to load

thejson: json (dict) loaded json (dict)

`nidm.utils.`**`remove_unicode_dict`**(*input_dict*, *encoding='utf-8'*)
remove unicode keys and values from dict, encoding in utf8

`nidm.utils.`**`save_template`**(*output_file*, *html_snippet*)

`nidm.utils.`**`set_permissions`**(*path*, *permission=128*)

`nidm.utils.`**`sub_template`**(*template*, *template_tag*, *substitution*)
make a substitution for a template_tag in a template

nidm.workflow module

Module contents

2.4 Development

2.4.1 How do I contribute a new query?

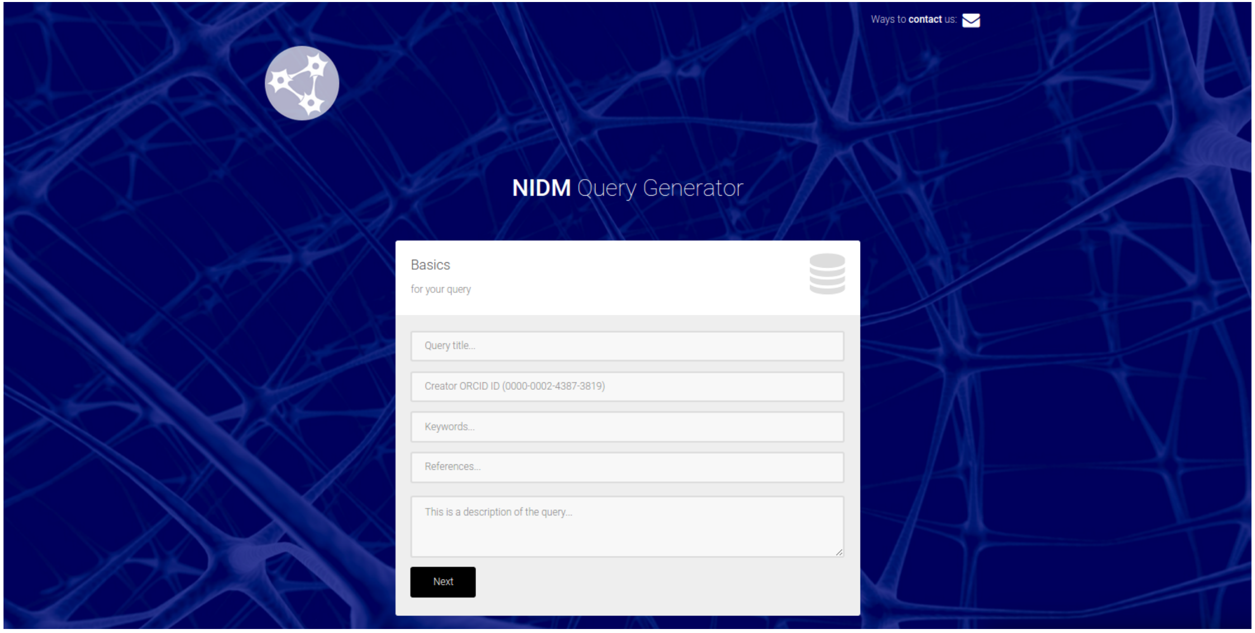
We have a dynamic web interface that will allow you to generate, preview (and eventually test) a new data structure for adding to nidm-query. This can also be done, using the same functions, programatically. You can then add it to the repo by submitting a pull request to add it. A pull request affords group discussion, and we will eventually have continuous integration that will run tests on your new query. We recommend that you use the generation functions to ensure accuracy in the format and fields of your data structure.

Web Query Generator

To generate a query with the interactive web interface, first start up the nidm application

```
nidm
```

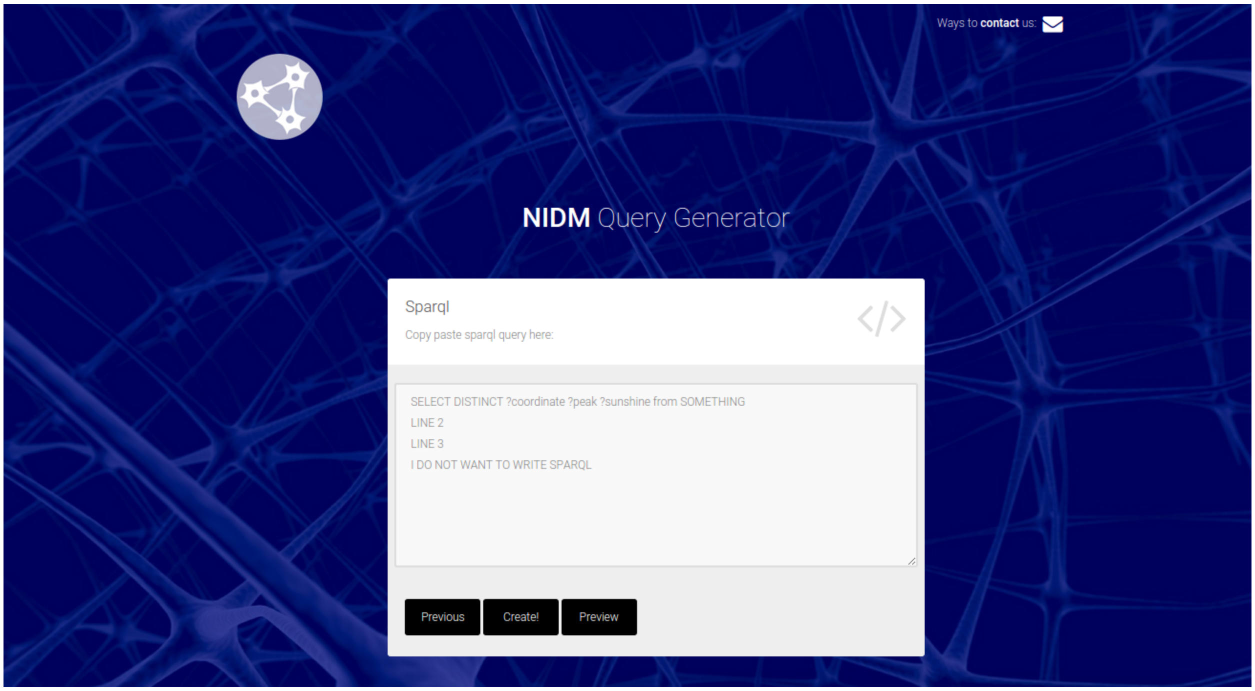
Then open your browser to localhost:8088/query/new. You will be presented with fields to fill in for your query:



Fields

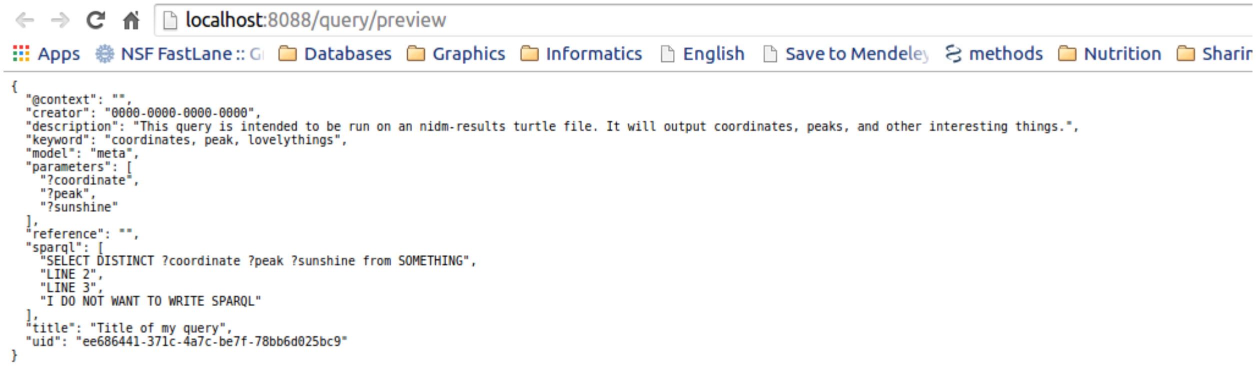
- title: This should be a single sentence that describes what your query does, what object model it is associated with, etc.
- creator: We ask for your ORCID ID to be filled into the creator spot. If you don't have this, you can put a name or email. We are currently not validating this, so you can really do whatever you like. We will (hopefully) decide on a standard.
- keywords: should be keywords to describe your query.
- component: name of the nidm component the query belongs to - one of [PROV, Results, Experiment, Workflow, Dataset Description, General]
- format: the type of sparql query - one of [SELECT, CONSTRUCT, ASK, UPDATE]
- model: name of the object model being queried - one of [fMRI Results, Freesurfer, General]
- description: This should be a text description of your query, please give details. The title and description will eventually be provided in a static web interface served with the repo for people to search and find queries they want to use.

When you click next, the next page is where you should copy paste your sparql:



```
SELECT DISTINCT ?coordinate ?peak ?sunshine from SOMETHING
LINE 2
LINE 3
I DO NOT WANT TO WRITE SPARQL
```

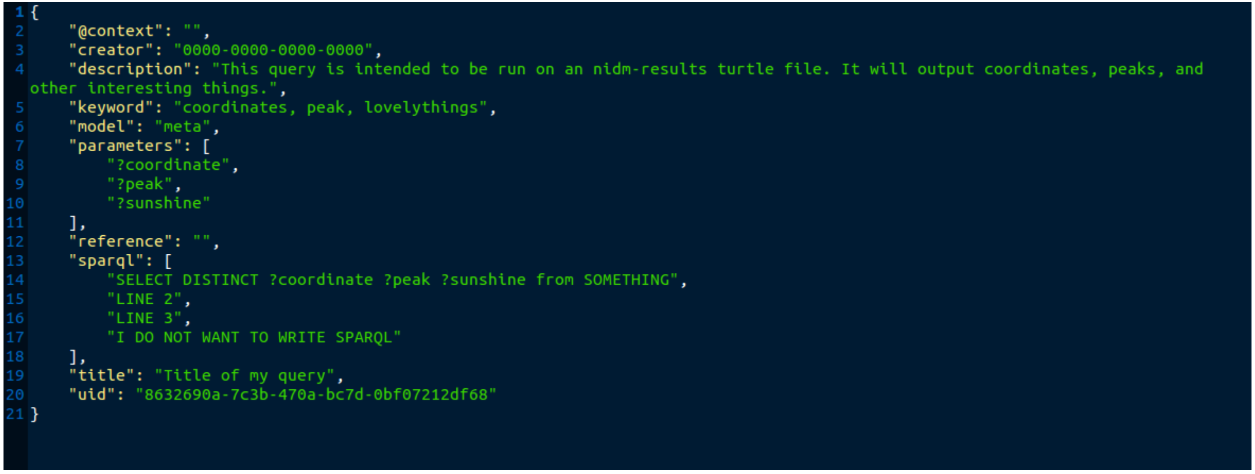
If you are not ready to generate your file, you can click "Preview" for a new tab to open with the query. The parameters that the user is allowed to ask for will be extracted from your "select" line, indicated by a word preceded with ? (e.g., ?hello). Do not worry about capitalization.



```
{
  "@context": "",
  "creator": "0000-0000-0000-0000",
  "description": "This query is intended to be run on an nidm-results turtle file. It will output coordinates, peaks, and other interesting things.",
  "keyword": "coordinates, peak, lovelythings",
  "model": "meta",
  "parameters": [
    "?coordinate",
    "?peak",
    "?sunshine"
  ],
  "reference": "",
  "sparql": [
    "SELECT DISTINCT ?coordinate ?peak ?sunshine from SOMETHING",
    "LINE 2",
    "LINE 3",
    "I DO NOT WANT TO WRITE SPARQL"
  ],
  "title": "Title of my query",
  "uid": "ee686441-371c-4a7c-be7f-78bb6d025bc9"
}
```

When you are ready to save your query, click "Create!"

A file will be downloaded to your computer. You should drop this file into the appropriate directory in your nidm-query. repo and submit a PR to add it to the nidm-api. We will eventually have tests for the queries, and an interactive web interface hosted with the nidm-api to explore the queries available (before downloading the nidm-api).



```
{
    "@context": "",
    "creator": "0000-0000-0000-0000",
    "description": "This query is intended to be run on an nidm-results turtle file. It will output coordinates, peaks, and other interesting things.",
    "keyword": "coordinates, peak, lovelythings",
    "model": "meta",
    "parameters": [
        "?coordinate",
        "?peak",
        "?sunshine"
    ],
    "reference": "",
    "sparql": [
        "SELECT DISTINCT ?coordinate ?peak ?sunshine from SOMETHING",
        "LINE 2",
        "LINE 3",
        "I DO NOT WANT TO WRITE SPARQL"
    ],
    "title": "Title of my query",
    "uid": "8632690a-7c3b-470a-bc7d-0bf07212df68"
}
```

You can then download to your local machine:

2.4.2 How do I develop the API?

You will want to fork the repo, clone the fork, and then run the flask application directly (so that it updates with changes to your code):

```
git clone https://github.com/[username]/nidm-api
cd nidm-api
python setup.py install --user
python nidm/app.py
```

2.4.3 How does this work?

Flask is a web framework (in python) that makes it ridiculously easy to do stuff with python in the browser. You can conceptually think of it like Django released wild and free from its mom's minivan. If you look at app.py, you will see a bunch of functions that return different views, and each view is associated with a particular url (with variables) that the user might want. You should first familiarize yourself with flask before trying to develop with it.

Queries vs. API

The queries are kept separate from the api itself, in the nidm-query repo. We did this because the world of writing sparql, and developing a web framework / API to serve the queries, are two separate things. A developer writing queries should be able to submit a PR to add a single file to the nidm-queries repo without needing to know about the nidm-api infrastructure. A developer working on the API shouldn't need to worry about the sparql side of things.

Application Logic

The basic application logic is as follows:

- The user installs the application with pip. This installs the python modules to the user's site-packages, but it also adds an executable, "nidm" to the users bin. This exectuable can be run to start the server instantly.
- Upon the creation of the server, the nidm-queries repo is downloaded to a temporary directory. This ensures that queries are up to date at the start of the server. If you are using the functions from within your application, you can download the repo to a location of your choice and specify the location in your application.
- The queries are json (ld) files. This just means they have a key called @context with some kind of stuff that semantic web folk understand. We are showing them as standard .json files because the .jsonld extension is not widely known, and could be confusing.
- The tool reads in all queries, and presents valid queries to the user at the base url of the server, localhost:8088. (Note that validation is not currently implemented). The user can select a query of choice based on the unique id, the "uid" variable in the json structure presented at localhost:8088.
- The user can then look at the details for a query by way of locahost:8088/api/[qid], or perform a query on a ttl file with localhost:8088/api/query/[qid]?ttl=[ttl_file]. The [ttl_file] can be a local path, or a URL. This is the extent of the tool thus far, it is still under development.

2.4.4 Serving an API and web interfaces

The url /api/[more-stuff-here] is linked up to serve a RESTful API, however the beauty of flask is that we can configure other URLs to do other interesting things. For example, /create might bring up an interactive web interface to write inputs to generate a new query object. /api/visual may be configured to return an interactive d3 or neo4j version of some part of the graph extracted from your ttl file. Having python and the infinite amount of web visualization technology at our fingertips makes the options really unlimited.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index