

---

# **nidm Documentation**

***Release 1.0***

**NIDASH Working Group**

February 22, 2016



<b>1</b>	<b>Why do I want to use this?</b>	<b>3</b>
<b>2</b>	<b>Under Development</b>	<b>5</b>
2.1	Installation . . . . .	5
2.2	Getting Started . . . . .	5
2.3	nidm . . . . .	10
2.4	Development . . . . .	10
<b>3</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



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.



---

## **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.





---

## 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

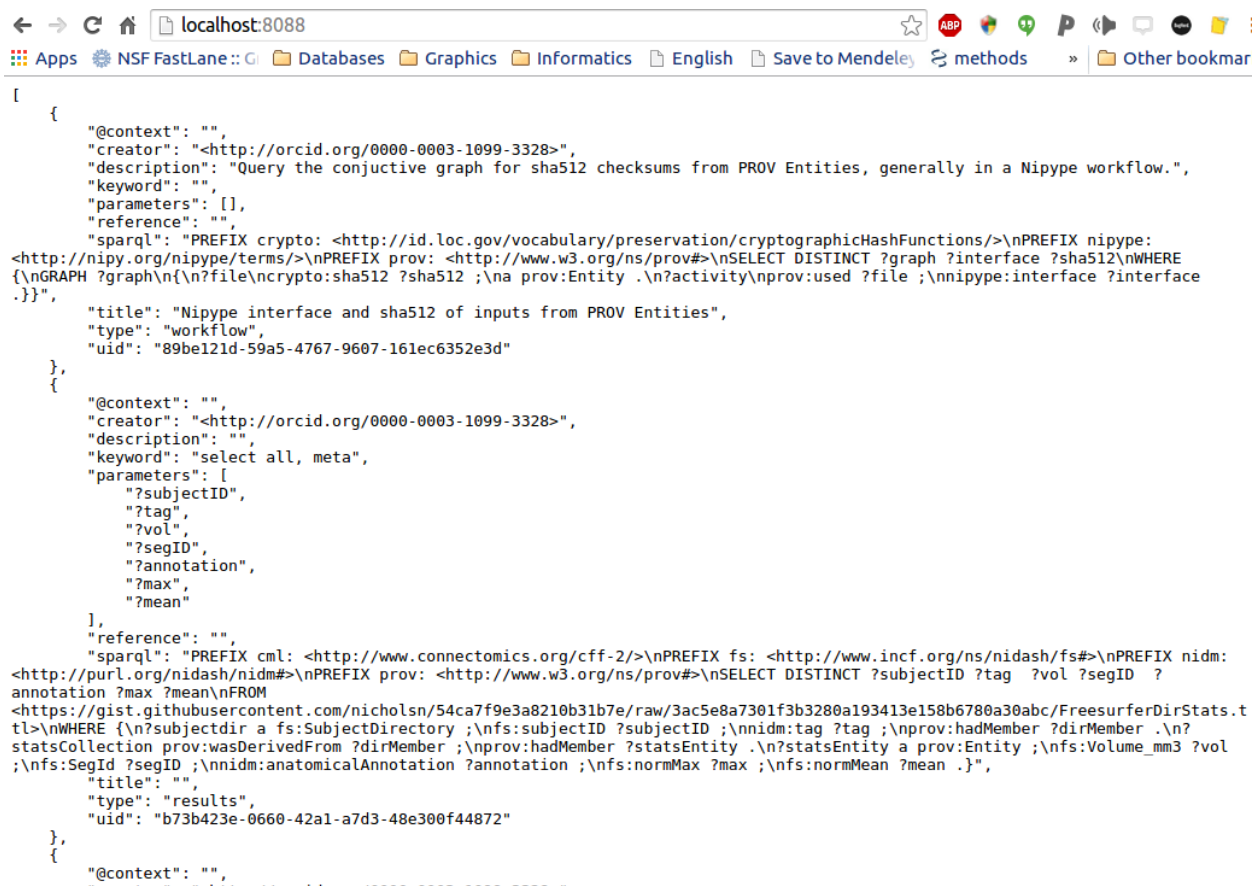
You have three options for using the nidm-api: as a local REST API, a REST API (served to the world on a public server), or as modules (to integrate into your python applications).

### 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 conjunctive 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\n}}",
    "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\n<https://gist.githubusercontent.com/nicholsn/54ca7f9e3a8210b31b7e/raw/3ac5e8a7301f3b3280a193413e158b6780a30abc/FreesurferDirStats.tl>\nWHERE {\n?subjectdir a fs:SubjectDirectory ;\n\nfs:subjectID ?subjectID ;\n\nnidm:tag ?tag ;\n\nprov:hadMember ?dirMember .\n\n?statsCollection prov:wasDerivedFrom ?dirMember ;\n\nprov:hadMember ?statsEntity .\n\n?statsEntity a prov:Entity ;\n\nfs:Volume_mm3 ?vol ;\n\nfs:SegId ?segID ;\n\nnidm:anatomicalAnnotation ?annotation ;\n\nfs:normMax ?max ;\n\nfs:normMean ?mean .\n}",
    "title": "",
    "type": "results",
    "uid": "b73b423e-0660-42a1-a7d3-48e300f44872"
  },
  {
    "@context": "",
    "reference": ""
  }
]
```

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:

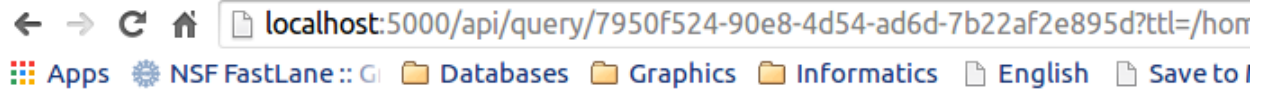
```
http://localhost:8088/api/7950f524-90e8-4d54-ad6d-7b22af2e895d
```

```
{
  "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.ttl
```

The only difference is that we've added the "query" to the URL, and specified the `ttl` file as a variable, indicated by the "ttl=" 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:



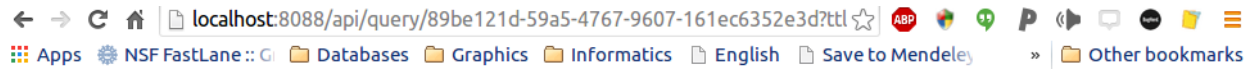
```
{
  "result": [
    {
      "coordinate": "[ 25.5, 39.5, -17.8 ]",
      "peak_name": "Peak 0001_1",
      "pvalue_uncorrected": 4.07408045586e-05,
      "z_score": 3.94
    },
    {
      "coordinate": "[ -8.52, -92.4, -5.73 ]",
      "peak_name": "Peak 0001_1",
      "pvalue_uncorrected": 2.0838886172200001e-13,
      "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://raw.githubusercontent.com/incf-r
```

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 conjunctive 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 .}\n}",
    "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:

```
#!/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

`nidm.experiment` module

`nidm.query` module

`nidm.results` module

`nidm.scripts` module

`nidm.utils` module

`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

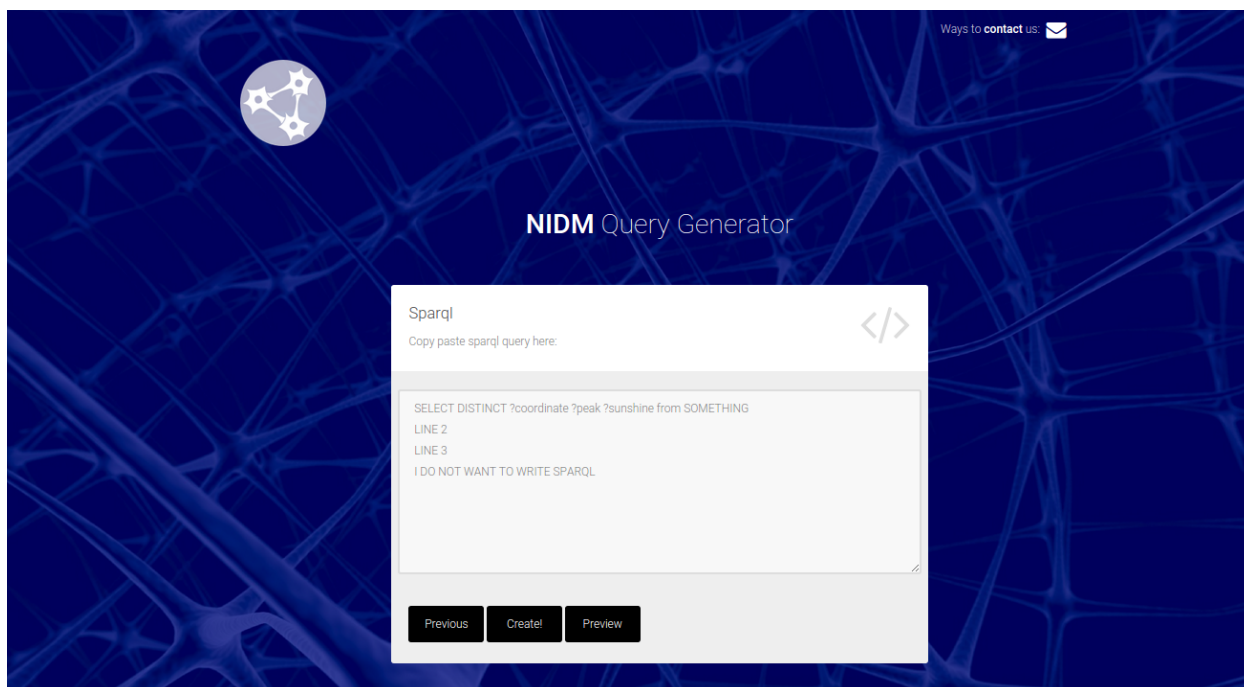
```
nidm
```

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:



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.

```
localhost:8088/query/preview
Apps NSF FastLane:: G Databases Graphics Informatics English Save to Mendeley methods Nutrition Sharir
{
  "@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`).

```

1 {
2   "@context": "",
3   "creator": "0000-0000-0000-0000",
4   "description": "This query is intended to be run on an nidm-results turtle file. It will output coordinates, peaks, and
5   other interesting things.",
6   "keyword": "coordinates, peak, lovelythings",
7   "model": "meta",
8   "parameters": [
9     "?coordinate",
10    "?peak",
11    "?sunshine"
12  ],
13  "reference": "",
14  "sparql": [
15    "SELECT DISTINCT ?coordinate ?peak ?sunshine from SOMETHING",
16    "LINE 2",
17    "LINE 3",
18    "I DO NOT WANT TO WRITE SPARQL"
19  ],
20  "title": "Title of my query",
21  "uid": "8632690a-7c3b-470a-bc7d-0bf07212df68"
22 }
```

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 executable 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 `localhost: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.

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## n

nidm, [10](#)  
nidm.script, [10](#)  
nidm.script.post\_install, [10](#)  
nidm.templates, [10](#)



## N

nidm (module), [10](#)

nidm.script (module), [10](#)

nidm.script.post\_install (module), [10](#)

nidm.templates (module), [10](#)