nanopub

Release 1.2.11

Robin Richardson, Sven van der Burg

Oct 24, 2022

GETTING STARTED

Nanopublications are a formalized and machine-readable way of communicating the smallest possible units of publishable information. This could be, for example, the outcome of a scientific study or a claim made by a particular scientist.

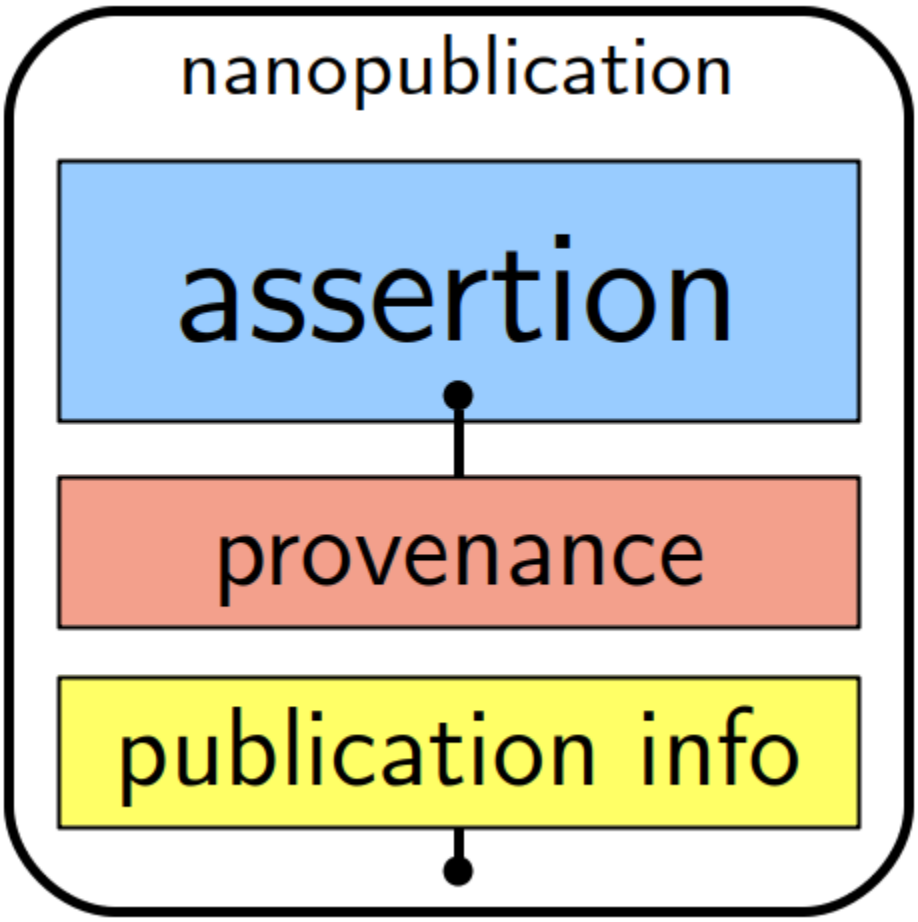
Nanopublications are searchable, citable, and contain authorship and attribution information. The aim is to encourage individual scientific results to be released in a traceable and interoperable format. As such, nanopublications are an effective FAIR means of communicating scientific claims and results. Read more about them at http://nanopub.org/.

CHAPTER

ONE

DIFFERENT ELEMENTS OF A NANOPUBLICATION

From nanopub.org documentation (2020/12/02)



Schematic representation of a nanopub

As can be seen in this image, a nanopublication has three basic elements:

1. Assertion: The assertion is the main content of a nanopublication in the form of an small atomic unit of information
2. Provenance: This part describes how the assertion above came to be. This can include the scientific methods that were used to generate the assertion, for example a reference to the kind of study that was performed and its

parameters.

3. Publication Info: This part contains metadata about the nanopublication as a whole, such as when and by whom it was created and the license terms for its reuse.

CHAPTER

TWO

SETUP INSTRUCTIONS

2.1 Install nanopub library

Install using pip:

```
pip install nanopub
```

2.2 Nanopub-java dependency

The nanopub library currently uses the nanopub-java tool for signing and publishing new nanopublications. This is automatically installed by the library.

2.2.1 Java

If you want to publish nanopublications you need to have the java runtime environment installed, this might already be installed on your system. You can check this for unix:

```
java --version
```

Or follow these instructions for windows

2.2.2 Installing java

If java is not installed follow these instructions

2.3 Setup for users new to python

We recommend using anaconda to install python and manage python dependencies

2.4 Setup your profile

To publish to the nanopub server you need to setup your profile (note that you can use fetch and search functionality without a profile). This allows the nanopub server to identify you.

Run the following interactive command:

```
setup_nanopub_profile
```

This will setup the following:

2.4.1 Stored profile

A local version of the profile will be stored in the nanopub user config dir (by default `HOMEDIR/.nanopub/profile.yml`)

2.4.2 RSA keys

It will add and store RSA keys to sign your nanopublications. By default they are stored under `HOMEDIR/.nanopub/id_rsa` and `HOMEDIR/.nanopub/id_rsa.pub`.

2.4.3 ORCID iD

This includes your ORCID iD (i.e. https://orcid.org/0000-0000-0000-0000). If you don't have an ORCID iD yet, you need to register. We use the ORCID iD to automatically add as author to the provenance of any nanopublication you will publish using this library.

2.4.4 Introductory nanopublication

We encourage you to make use of `setup_nanopub_profile`'s option to publish your profile to the nanopub servers. This links your ORCID iD to your RSA key, thereby making all your publications linkable to you. Here is an example introductory nanopublicaiton.

The link to this nanopublication is also stored in your profile.

CHAPTER

THREE

THE NANOPUB TEST SERVER

Throughout this documentation we make use of the nanopub test server by setting `use_test_server=True` when instantiating `NanopubClient`:

```
>>> from nanopub import NanopubClient
>>> client = NanopubClient(use_test_server=True)
```

This will search and fetch from, and publish to the nanopub test server.

When learning about nanopub using the testserver is a good idea, because:

- You are free to experiment with publishing without polluting the production server.
- You can draft a publication and know exactly what it will look like on the nanopub server without polluting the production server.
- When searching (and to a lesser extent fetching) you are not putting an unnecessary load on the production server.

3.1 Test purl URIs do not point to the test server

There is one caveat when using the test server that can be confusing: The purl URI (for example: http://purl.org/np/RA71u9tYPd7ZQifE_6hXjqVim6pkweuvjoi-8ehvLvzg8) points to the nanopub production server resulting in a 404 page not found error.

A manual workaround is:

1. Open http://purl.org/np/RA71u9tYPd7ZQifE_6hXjqVim6pkweuvjoi-8ehvLvzg8 in your browser
2. Notice that the URL changed to http://server.nanopubs.lod.labs.vu.nl/RA71u9tYPd7ZQifE_6hXjqVim6pkweuvjoi-8ehvLvzg8.
3. Replace 'server' with 'test-server': http://test-server.nanopubs.lod.labs.vu.nl/RA71u9tYPd7ZQifE_6hXjqVim6pkweuvjoi-8ehvLvzg8.

   NB: `NanopubClient.fetch()` does this for you if `use_test_server=True`.

CHAPTER

FOUR

PUBLISHING NANOPUBLICATIONS

The nanopub library provides an intuitive API that makes publishing nanopublications much easier. The rationale is that you often do not want to worry about the details of composing the RDF that is often the same in each nanopublication. Instead you should focus on the content of your nanopublication: the assertion.

4.1 Prerequisits for publishing

Before you can publish you should *setup your profile*

4.2 Quickly publishing nanopublications using `claim`

You can quickly publish a nanopublicaiton with a single simple statement using the `claim` method:

```
>>> from nanopub import NanopubClient

>>> # Create the client (we use use_test_server=True to point to the test server)
>>> client = NanopubClient(use_test_server=True)

>>> # Publish a simple statement to the server
>>> client.claim('All cats are gray')
Published to http://purl.org/np/RA47eJP2UBJCWuJ324c6Qw0OwtCb8wCrprwSk39am7xck
```

View the resulting nanopublication here.

The generated RDF makes use of the Hypotheses and Claims Ontology (HYCL)

This is the assertion part of the nanopublication, denoting the statement:

```
@prefix hycl: <http://purl.org/petapico/o/hycl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sub: <http://purl.org/np/RA47eJP2UBJCWuJ324c6Qw0OwtCb8wCrprwSk39am7xck#> .

sub:assertion {
    sub:mystatement a hycl:Statement ;
        rdfs:label "All cats are gray" .
}
```

The provenance part of the nanopublication denotes that the ORCID iD from the profile claimed the statement:

```
@prefix hycl: <http://purl.org/petapico/o/hycl#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix sub: <http://purl.org/np/RA47eJP2UBJCWuJ324c6Qw0OwtCb8wCrprwSk39am7xck#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

sub:provenance {
    sub:assertion prov:generatedAtTime "2020-12-01T10:39:09.427920"^^xsd:dateTime ;
        prov:wasAttributedTo <https://orcid.org/0000-0000-0000-0000> .

    <https://orcid.org/0000-0000-0000-0000> hycl:claims sub:mystatement .
}
```

4.3 A simple recipe for publishing RDF triples

You can use `Publication` objects to easily publish nanopublications with your assertion (think of the assertion as the content of your nanopublication).

This is a 3-step recipe that works for most cases:

1. Construct a desired assertion using `rdflib`.
2. Make a `Publication` object using the assertion, making use of `Publication.from_assertion()`.
3. Publish the `Publication` object using `NanopubClient.publish()`.

Here is a minimal example:

```
>>> import rdflib
>>> from nanopub import NanopubClient, Publication
>>>
>>> # Create the client (we use use_test_server=True to point to the test server)
>>> client = NanopubClient(use_test_server=True)
>>>
>>> # 1. construct a desired assertion (a graph of RDF triples) using rdflib
>>> my_assertion = rdflib.Graph()
>>> my_assertion.add((rdflib.URIRef('www.example.org/timbernerslee'),
>>>                   rdflib.RDF.type,
>>>                   rdflib.FOAF.Person))
>>>
>>> # 2. Make a Publication object with this assertion
>>> publication = Publication.from_assertion(assertion_rdf=my_assertion)
>>>
>>> # 3. Publish the Publication object.
>>> publication_info = client.publish(publication)
Published to http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ
```

View the resulting nanopublication here.

This is the resulting assertion part of the nanopublication:

```
@prefix sub: <http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ#> .

sub:assertion {
```

(continues on next page)

(continued from previous page)

```
    <https://www.example.org/timbernerslee> a <http://xmlns.com/foaf/0.1/Person> .
}
```

The library automatically adds relevant RDF triples for the provenance part of the nanopublication:

```
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix sub: <http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

sub:provenance {
    sub:assertion prov:generatedAtTime "2020-12-01T10:44:32.367084"^^xsd:dateTime .
}
```

as well as for the publication info part of the nanopublication:

```
@prefix npx: <http://purl.org/nanopub/x/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix sub: <http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ#> .
@prefix this: <http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

sub:pubInfo {
    sub:sig npx:hasAlgorithm "RSA" ;
        npx:hasPublicKey "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCmso7vmRO/
↪Cp4Pt0RkJJkV5qfc1WFYU/jMtkdxxb5+lfIVXNV97XQnM1Tj4fkb/
↪W6jkP6fHl8mj8Q7hl7VgUnQ6I+B7cMGpxW9Z8Br+JNx8DPMMt08VCH5+JMENPRKl91r7rF/
↪YPWCAgL9eqXSixCNMNAj5RBmMTQoPuRkpgmt1wIDAQAB" ;
        npx:hasSignature "aPZMJ3Md6X1PHYvXJiNoRUni9+1oS9faCfiPRRCrj4K/uZPN0J/
↪znjxGuCUxoZRJ4b4RfSxmHFGRKfCFusJX+7Y3xuxYx4GYHzYhBciK7T5pO02V4w6sdwHLKd5E+Wcl0PTr2t3lEjq6yzY98wEXlZLAl
↪" ;
        npx:hasSignatureTarget this: .

    this: prov:generatedAtTime "2020-12-01T10:44:32.367084"^^xsd:dateTime ;
        prov:wasAttributedTo <https://orcid.org/0000-0000-0000-0000> .
}
```

CHAPTER

FIVE

USING THE NANOPUBLICATION'S NAMESPACE

In a nanopublication you often want to refer to a concept that is not defined somewhere on the WWW. In that case it makes sense to make use of the namespace of the nanopublication itself, see for example this assertion that uses `nanopub-uri#timbernerslee` to refer to the concept Tim Berner's Lee.

```
@prefix sub: <http://purl.org/np/RA_j6TPcnoQJ_XkISjugTgaRsFGLhpbZCC3mE7fXs0REI#> .

sub:assertion {
    sub:timbernerslee a <http://xmlns.com/foaf/0.1/Person> .
}
```

5.1 Using blank nodes

But how do you make use of the nanopublication's namespace if you do not have access to the published nanopublication URI yet? We solve that by making use of blank nodes.

Upon publication, any blank nodes in the rdf graph are replaced with the nanopub's URI, with the blank node name as a fragment. For example, if the blank node is called 'timbernerslee', that would result in a URI composed of the nanopub's (base) URI, followed by #timbernslee. We can thus use blank nodes to refer to new concepts, making use of the namespace of the to-be-published URI.

An example:

```
>>> import rdflib
>>> from nanopub import Publication, NanopubClient
>>>
>>> my_assertion = rdflib.Graph()
>>>
>>> # We want to introduce a new concept in our publication: Tim Berners Lee
>>> tim = rdflib.BNode('timbernerslee')
>>>
>>> # We assert that he is a person
>>> my_assertion.add((tim, rdflib.RDF.type, rdflib.FOAF.Person) )
>>>
>>> # And create a publication object for this assertion
>>> publication = Publication.from_assertion(assertion_rdf=my_assertion)
>>>
>>> # Let's publish this to the test server
>>> client = NanopubClient(use_test_server=True)
>>> client.publish(publication)
Published to http://purl.org/np/RAdaZsPRcY5usXFKwSBfz9g-HOu-Bo1XmmhQc4g7uESgU
```

View the full nanopublication here.

As you can see in the assertion, the 'timbernerslee' blank node is replaced with a uri in the nanopublication's namespace:

```
@prefix sub: <http://purl.org/np/RAdaZsPRcY5usXFKwSBfz9g-HOu-Bo1XmmhQc4g7uESgU#> .

sub:assertion {
    sub:timbernerslee a <http://xmlns.com/foaf/0.1/Person> .
}
```

5.2 Introducing a concept

You can optionally specify that the Publication introduces a particular concept using blank nodes. The pubinfo graph will note that this nanopub npx:introduces the concept. The concept should be a blank node (rdflib.term.BNode), and is converted to a URI derived from the nanopub's URI with a fragment (#) made from the blank node's name.

An example:

```
>>> import rdflib
>>> from nanopub import Publication, NanopubClient
>>>
>>> my_assertion = rdflib.Graph()
>>>
>>> # We want to introduce a new concept in our publication: Tim Berners Lee
>>> tim = rdflib.BNode('timbernerslee')
>>>
>>> # We assert that he is a person
>>> my_assertion.add((tim, rdflib.RDF.type, rdflib.FOAF.Person) )
>>>
>>> # We can create a publication introducing this new concept
>>> publication = Publication.from_assertion(assertion_rdf=my_assertion,
>>>                                          introduces_concept=tim)
>>>
>>> # Let's publish this to the test server
>>> client = NanopubClient(use_test_server=True)
>>> client.publish(publication)
Published to http://purl.org/np/RAq9gFEgxlOyG9SSDZ5DmBbyGet2z6pkrdWXIVYa6U6qI
Published concept to http://purl.org/np/RAq9gFEgxlOyG9SSDZ5DmBbyGet2z6pkrdWXIVYa6U6qI
↪#timbernerslee
```

Note that `NanopubClient.publish()` now also prints the published concept URI.

View the full nanopublication here.

The publication info of the nanopublication denotes that this nanopublication introduces the 'timbernerslee' concept:

```
@prefix npx: <http://purl.org/nanopub/x/> .
@prefix sub: <http://purl.org/np/RAq9gFEgxlOyG9SSDZ5DmBbyGet2z6pkrdWXIVYa6U6qI#> .
@prefix this: <http://purl.org/np/RAq9gFEgxlOyG9SSDZ5DmBbyGet2z6pkrdWXIVYa6U6qI> .

sub:pubInfo {
   this: npx:introduces sub:timbernerslee .
}
```

CHAPTER

SIX

SETTING PUBLICATION INFO AND PROVENANCE

Here we show how you can control the publication info and provenance parts of the nanopublication.

6.1 Specifying where the nanopublication is derived from

You can specify that the nanopub's assertion is derived from another URI (such as an existing nanopublication):

```
import rdflib
from nanopub import Publication

my_assertion = rdflib.Graph()
my_assertion.add((rdflib.term.BNode('timbernserslee'), rdflib.RDF.type, rdflib.FOAF.
↪Person))


publication = Publication.from_assertion(
    assertion_rdf=my_assertion,
    derived_from=rdflib.URIRef('http://www.example.org/another-nanopublication'))
```

Note that `derived_from` may also be passed a list of URIs.

The provenance part of the publication will denote:

```
@prefix sub: <http://purl.org/nanopub/temp/mynanopub#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:provenance {
    sub:assertion prov:wasDerivedFrom <http://www.example.org/another-nanopublication> .
}
```

6.2 Attributing the assertion to someone

You can attribute the assertion to someone by specifying the `assertion_attributed_to` argument:

```
import rdflib
from nanopub import Publication

my_assertion = rdflib.Graph()
my_assertion.add((rdflib.term.BNode('timbernserslee'), rdflib.RDF.type, rdflib.FOAF.
↪Person))

publication = Publication.from_assertion(
    assertion_rdf=my_assertion,
    assertion_attributed_to=rdflib.URIRef('https://orcid.org/0000-0000-0000-0000'))
```

The provenance part of the publication will denote:

```
@prefix : <http://purl.org/nanopub/temp/mynanopub#> .
@prefix prov: <http://www.w3.org/ns/prov#> .

:provenance {
    :assertion prov:wasAttributedTo <https://orcid.org/0000-0000-0000-0000> .
}
```

Note: Often the assertion should be attributed to yourself. Instead of passing your ORCID iD to `assertion_attributed_to`, you can easily tell nanopub to attribute the assertion to the ORCID iD in your profile by setting `attribute_assertion_to_profile=True`.

6.3 Specifying custom provenance triples

You can add your own triples to the provenance graph of the nanopublication by passing them in an `rdflib.Graph` object to the `provenance_rdf` argument:

```
import rdflib
from nanopub import namespaces, Publication

my_assertion = rdflib.Graph()
my_assertion.add((rdflib.term.BNode('timbernserslee'), rdflib.RDF.type, rdflib.FOAF.
↪Person))

provenance_rdf = rdflib.Graph()
provenance_rdf = provenance_rdf.add((rdflib.term.BNode('timbernserslee'),
                                     namespaces.PROV.actedOnBehalfOf,
                                     rdflib.term.BNode('markzuckerberg')))
publication = Publication.from_assertion(assertion_rdf=my_assertion,
                                         provenance_rdf=provenance_rdf)
```

6.4 Specifying custom publication info triples

You can add your own triples to the publication info graph of the nanopublication by passing them in an `rdflib.Graph` object to the `pubinfo_rdf` argument:

```
import rdflib
from nanopub import namespaces, Publication

my_assertion = rdflib.Graph()
my_assertion.add((rdflib.term.BNode('timbernserslee'), rdflib.RDF.type, rdflib.FOAF.
↪Person))

pubinfo_rdf = rdflib.Graph()
pubinfo_rdf = pubinfo_rdf.add((rdflib.term.BNode('activity'),
                               rdflib.RDF.type,
                               namespaces.PROV.Activity))
publication = Publication.from_assertion(assertion_rdf=my_assertion,
                                         pubinfo_rdf=pubinfo_rdf)
```

CHAPTER

SEVEN

RETRACTING A NANOPUBLICATION

A nanopublication is persistent, you can never edit nor delete it. You can however retract a nanopublication. This is done by publishing a new nanopublication that states that you retract the original publication. You can use `NanopubClient.retract()`:

```
>>> from nanopub import NanopubClient
>>> client = NanopubClient(use_test_server=True)
>>> client.retract('http://purl.org/np/RAfk_zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ')
Published to http://purl.org/np/RAv75Xhhz5jv--Nnu9RDqIGy2xHr74REGC4vtOSxrwX4c
```

View the full retraction nanopublication here.

The assertion states that the researcher (denoted by the ORCID iD from your profile) retracts the provided nanopublication:

```
@prefix npx: <http://purl.org/nanopub/x/> .
@prefix sub: <http://purl.org/np/RAv75Xhhz5jv--Nnu9RDqIGy2xHr74REGC4vtOSxrwX4c#> .

sub:assertion {
    <https://orcid.org/0000-0000-0000-0000> npx:retracts <http://purl.org/np/RAfk_
↪zBYDerxd6ipfv8fAcQHEzgZcVylMTEkiLlMzsgwQ> .
}
```

By default nanopublications that have a valid retraction do not show up in search results. A valid retraction is a retraction that is signed with the same public key as the nanopublication that it retracts.

7.1 Retracting a nanopublication that is not yours

By default we do not retract nanopublications that are not yours (i.e. signed with another public key). If you try to do this it will trigger an AssertionError:

```
>>> from nanopub import NanopubClient
>>> client = NanopubClient(use_test_server=True)
>>> not_my_nanopub_uri = 'http://purl.org/np/RAr6rs7o8Sr5OGCs0127ah37DYUvgiWzjOuCvV-
↪OSusAk'
>>> client.retract(not_my_nanopub_uri)
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-30-7141d9e82fbc> in <module>
      1 not_my_nanopub_uri = 'http://purl.org/np/RAr6rs7o8Sr5OGCs0127ah37DYUvgiWzjOuCvV-
↪OSusAk'
```

(continues on next page)

(continued from previous page)

```
----> 2 client.retract(not_my_nanopub_uri)

~/projects/fair-workflows/nanopub/nanopub/client.py in retract(self, uri, force)
    265         """
    266         if not force:
--> 267             self._check_public_keys_match(uri)
    268         assertion_rdf = rdflib.Graph()
    269         orcid_id = profile.get_orcid_id()

~/projects/fair-workflows/nanopub/nanopub/client.py in _check_public_keys_match(self,␣
↪uri)
    245                                 f'this one: {their_public_key}')
    246             if their_public_key != profile.get_public_key():
--> 247                 raise AssertionError('The public key in your profile does not␣
↪match the public key'
    248                                      'that the publication that you want to␣
↪retract is signed '
    249                                      'with. Use force=True to force retraction␣
↪anyway.')

AssertionError: The public key in your profile does not match the public keythat the␣
↪publication that you want to retract is signed with. Use force=True to force␣
↪retraction anyway.
```

We can use force=True to override this behavior:

```
client.retract(not_my_nanopub_uri, force=True)
```

7.2 Find retractions of a given nanopublication

You can find out whether a given publication is retracted and what the nanopublications are that retract it using `NanopubClient.find_retractions_of`:

```
>>> from nanopub import NanopubClient
>>> client = NanopubClient(use_test_server=True)
>>> # This URI has 1 retraction:
>>> client.find_retractions_of('http://purl.org/np/RAirauh-vy5f7UJEMTm08C5bh5pnWD-abb-
↪qk3fPYWCzc')
['http://purl.org/np/RADjlGIB8Vqt7NbG1kqzw-4aIV_k7nyIRirMhPKEYVSlc']
>>> # This URI has no retractions
>>> client.find_retractions_of('http://purl.org/np/
↪RAeMfoa6I05zoUmK6sRypCIy3wIpTgS8gkum7vdfOamn8')
[]
```

CHAPTER

EIGHT

SEARCHING THE NANOPUB SERVER

The `NanopubClient` provides methods for searching the nanopub server. It provides an (uncomplete) mapping to the nanopub server grlc endpoint.

8.1 Text search

Search for all nanopublications containing some text using `NanopubClient.find_nanopubs_with_text()`

```
from nanopub import NanopubClient
client = NanopubClient()
results = client.find_nanopubs_with_text('fair')
```

8.2 Triple pattern search

Search for nanopublications whose assertions contain triples that match a specific pattern.

```
from nanopub import NanopubClient
client = NanopubClient()
# Search for nanopublications whose assertions contain triples that are␣
↪```rdf:Statement```s.
results = client.find_nanopubs_with_pattern(
                pred='http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
                obj='http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement')
```

8.3 Search on introduced concept

Search for any nanopublications that introduce a concept of the given type, that contain text with the given search term.

```
from nanopub import NanopubClient
client = NanopubClient()
# Search for nanopublications that introduce a concept that is a ```p-plan:Step```.
results = client.find_things('http://purl.org/net/p-plan#Step')
```

8.4 Interpreting search results

Each search method returns a generator of dicts depicting matching nanopublications.

Each dict has the following key-value pairs:

- `date`: The date and time the nanopublication was created.
- `description`: A description of the nanopublication that was parsed from the nanopublication RDF.
- `np`: The URI of the matching nanopublication.

Example results (from `NanopubClient.find_nanopubs_with_text('fair')`):

```
>>> print(list(results))
[{'date': '2020-05-01T08:05:25.575Z',
  'description': 'The primary objective of the VODAN Implementation Network is '
                 'to showcase the creation and deployment of FAIR data related '
                 'to COVID-19',
  'np': 'http://purl.org/np/RAdDKjIGPt_2mE9oJtB3YQX6wGGdCC8ZWpkxEIoHsxOjE'},
 {'date': '2020-05-14T09:34:53.554Z',
  'description': 'FAIR IN community',
  'np': 'http://purl.org/np/RAPE0A-NrIZDeX3pvFJr0uHshocfXuUj8n_J3BkY0sMuU'}]
```

8.5 Returning retracted publications in search

By default nanopublications that have a valid retraction do not show up in search results. A valid retraction is a retraction that is signed with the same public key as the nanopublication that it retracts. You can toggle this behavior with the `filter_retracted` parameter, here is an example with `NanopubClient.find_nanopubs_with_text`:

```
from nanopub import NanopubClient
client = NanopubClient()
# Search for nanopublications containing the text fair, also returning retracted␣
↪publications.
results = client.find_nanopubs_with_text('fair', filter_retracted=False)
```

8.6 Filtering search results for a particular publication key

You can filter search results to publications that are signed with a specific publication key (effectively filtering on publications from a single author). You use the `pubkey` argument for that. Here is an example with `NanopubClient.find_nanopubs_with_text`:

```
from nanopub import NanopubClient, profile
# Search for nanopublications containing the text 'test',
# filtering on publications signed with my publication key.
client = NanopubClient(use_test_server=True)
my_public_key = profile.get_public_key()
results = client.find_nanopubs_with_text('test', pubkey=my_public_key)
```

CHAPTER

NINE

FETCHING NANOPUBLICATIONS

You can fetch nanopublications from the nanopub server using `NanopubClient.fetch()`. The resulting object is a `Publication` object that you can use to inspect the nanopublication.

```
from nanopub import NanopubClient

# Fetch the nanopublication at the specified URI
client = NanopubClient()
publication = client.fetch('http://purl.org/np/
↪RApJG4fwj0szOMBMiYGmYvd5MCtRle6VbwkMJUb1SxxDM')

# Print the RDF contents of the nanopublication
print(publication)

# Iterate through all triples in the assertion graph
for s, p, o in publication.assertion:
    print(s, p, o)

# Iterate through the publication info
for s, p, o in publication.pubinfo:
    print(s, p, o)

# Iterate through the provenance graph
for s, p, o in publication.provenance:
    print(s,p,o)

# See the concept that is introduced by this nanopublication (if any)
print(publication.introduces_concept)
```

CHAPTER

TEN

NANOPUB.CLIENT

This module includes a client for the nanopub server.

class nanopub.client.**NanopubClient**(*use_test_server=False*)

Provides utility functions for searching, creating and publishing RDF graphs as assertions in a nanopublication.

Parameters
use_test_server (*bool*) – Toggle using the test nanopub server.

claim(*statement_text: str*)

Quickly claim a statement.

Constructs statement triples around the provided text following the Hypotheses and Claims Ontology (http://purl.org/petapico/o/hycl).

Parameters
statement_text (*str*) – the text of the statement, example: 'All cats are grey'

Returns
Publication info with: 'nanopub_uri': the URI of the published nanopublication, 'concept_uri': the URI of the introduced concept (if applicable)

Return type
dict of str

fetch(*uri: str*)

Fetch nanopublication

Download the nanopublication at the specified URI.

Parameters
uri (*str*) – The URI of the nanopublication to fetch.

Returns
a Publication object representing the nanopublication.

Return type
Publication

find_nanopubs_with_pattern(*subj: Optional[str] = None*, *pred: Optional[str] = None*, *obj: Optional[str] = None*, *filter_retracted: bool = True*, *pubkey: Optional[str] = None*)

Pattern search.

Search the nanopub servers for any nanopubs matching the given RDF pattern. You can leave parts of the triple to match anything by not specifying subj, pred, or obj arguments.

Parameters

- **subj** (*str*) – URI of the subject that you want to match triples on.

- **pred** (*str*) – URI of the predicate that you want to match triples on.
- **obj** (*str*) – URI of the object that you want to match triples on.
- **pubkey** (*str*) – Public key that the matching nanopubs should be signed with
- **filter_retracted** (*bool*) – Toggle filtering for publications that are retracted. Default is True, returning only publications that are not retracted.

Yields
dicts depicting matching nanopublications. Each dict holds: 'np': the nanopublication uri, 'date': date of creation of the nanopublication, 'description': A description of the nanopublication (if found in RDF).

find_nanopubs_with_text(*text: str*, *pubkey: Optional[str] = None*, *filter_retracted: bool = True*)

Text search.

Search the nanopub servers for any nanopubs matching the given search text.

Parameters

- **text** (*str*) – The text to search on
- **pubkey** (*str*) – Public key that the matching nanopubs should be signed with
- **filter_retracted** (*bool*) – Toggle filtering for publications that are retracted. Default is True, returning only publications that are not retracted.

Yields
dicts depicting matching nanopublications. Each dict holds: 'np': the nanopublication uri, 'date': date of creation of the nanopublication, 'description': A description of the nanopublication (if found in RDF).

find_retractions_of(*source: Union[str, Publication]*, *valid_only=True*) → List[str]

Find retractions of given URI

Find all nanopublications that retract a certain nanopublication.

Parameters

- **source** (*str or nanopub.Publication*) – URI or Publication object to find retractions for
- **valid_only** (*bool*) – Toggle returning only valid retractions, i.e. retractions that are signed with the same public key as the publication they retract. Default is True.

Returns
List of uris that retract the given URI

find_things(*type: str*, *searchterm: str = ''*, *pubkey: Optional[str] = None*, *filter_retracted: bool = True*)

Search things (experimental).

Search for any nanopublications that introduce a concept of the given type, that contain text with the given search term.

Parameters

- **type** (*str*) – A URI denoting the type of the introduced concept
- **searchterm** (*str*) – The term that you want to search on
- **pubkey** (*str*) – Public key that the matching nanopubs should be signed with
- **filter_retracted** (*bool*) – Toggle filtering for publications that are retracted. Default is True, returning only publications that are not retracted.

Yields
dicts depicting matching nanopublications. Each dict holds: 'np': the nanopublication uri, 'date': date of creation of the nanopublication, 'description': A description of the nanopublication (if found in RDF).

publish(*publication:* Publication)

Publish a Publication object.

Publish Publication object to the nanopub server. It uses nanopub_java commandline tool to sign the nanopublication RDF with the RSA key in the profile and then publish.

Parameters
publication (Publication) – Publication object to publish.

Returns
Publication info with: 'nanopub_uri': the URI of the published nanopublication, 'concept_uri': the URI of the introduced concept (if applicable)

Return type
dict of str

retract(*uri: str*, *force=False*)

Retract a nanopublication.

Publish a retraction nanpublication that declares retraction of the nanopublication that corresponds to the 'uri' argument.

Parameters

- **uri** (*str*) – The uri pointing to the to-be-retracted nanopublication
- **force** (*bool*) – Toggle using force to retract, this will even retract the nanopublication if it is signed with a different public key than the one in the user profile.

Returns
Publication info with: 'nanopub_uri': the URI of the published nanopublication, 'concept_uri': the URI of the introduced concept (if applicable)

Return type
dict of str

CHAPTER

ELEVEN

NANOPUB.PUBLICATION

This module holds code for representing the RDF of nanopublications, as well as helper functions to make handling RDF easier.

class nanopub.publication.**Publication**(*rdf: ConjunctiveGraph*, *source_uri: Optional[str] = None*)

Representation of the rdf that comprises a nanopublication

rdf

The full RDF graph of this nanopublication

Type
rdflib.ConjunctiveGraph

assertion

The part of the graph describing the assertion.

Type
rdflib.Graph

pubinfo

The part of the graph describing the publication information.

Type
rdflib.Graph

provenance

The part of the graph describing the provenance.

Type
rdflib.Graph

source_uri

The URI of the nanopublication that this Publication represents (if applicable)

Type
str

introduces_concept

The concept that is introduced by this Publication.

signed_with_public_key

The public key that this Publication is signed with.

is_test_publication

Whether this is a test publication

classmethod from_assertion(*assertion_rdf: Graph*, *introduces_concept: Optional[BNode] = None*, *derived_from=None*, *assertion_attributed_to=None*, *publication_attributed_to=None*, *attribute_assertion_to_profile: bool = False*, *attribute_publication_to_profile: bool = True*, *provenance_rdf: Optional[Graph] = None*, *pubinfo_rdf: Optional[Graph] = None*, *add_generated_at_time: bool = True*)

Construct Nanopub object based on given assertion.

Any blank nodes in the rdf graph are replaced with the nanopub's URI, with the blank node name as a fragment. For example, if the blank node is called 'step', that would result in a URI composed of the nanopub's (base) URI, followed by #step.

Parameters

- **assertion_rdf** (*rdflib.Graph*) – The assertion RDF graph.
- **introduces_concept** (*rdflib.term.BNode*) – the pubinfo graph will note that this nanopub npx:introduces the concept. The concept should be a blank node (rdflib.term.BNode), and is converted to a URI derived from the nanopub's URI with a fragment (#) made from the blank node's name.
- **derived_from** (*rdflib.URIRef, str, or list*) – Add a triple to the provenance graph stating that this nanopub's assertion prov:wasDerivedFrom the given URI. If a list of URIs is passed, a provenance triple will be generated for each.
- **assertion_attributed_to** (*rdflib.URIRef or str*) – the provenance graph will note that this nanopub's assertion prov:wasAttributedTo the given URI.
- **publication_attributed_to** (*rdflib.URIRef or str*) – the pubInfo graph will note that this nanopub itself prov:wasAttributedTo the given URI. If 'None' then this defaults to using the ORCID id provided in the user's profile.
- **attribute_assertion_to_profile** (*bool*) – Attribute the assertion to the ORCID iD in the profile
- **attribute_publication_to_profile** (*bool*) – Attribute the publication to the ORCID iD in the profile
- **provenance_rdf** (*rdflib.Graph*) – RDF triples to be added to provenance graph of the nanopublication. This is optional, for most cases the defaults will be sufficient.
- **pubinfo_rdf** (*rdflib.Graph*) – RDF triples to be added to the publication info graph of the nanopublication. This is optional, for most cases the defaults will be sufficient.
- **add_generated_at_time** (*bool*) – Add prov:generatedAtTime in the pubinfo and prov graphs

nanopub.publication.**replace_in_rdf**(*rdf: Graph*, *oldvalue*, *newvalue*)

Replace values in RDF.

Replace all subjects or objects matching *oldvalue* with *newvalue*. Replaces in place.

Parameters

- **rdf** (*rdflib.Graph*) – The RDF graph in which we want to replace nodes
- **oldvalue** – The value to be replaced
- **newvalue** – The value to replace with

CHAPTER

TWELVE

NANOPUB.NAMESPACES

This module holds handy namespaces that are often used in nanopublications.

nanopub.namespaces.**AUTHOR = Namespace('http://purl.org/person#')**
Namespace for *http://purl.org/person#*

nanopub.namespaces.**HYCL = Namespace('http://purl.org/petapico/o/hycl#')**
Namespace for *http://purl.org/petapico/o/hycl#*

nanopub.namespaces.**NP = Namespace('http://www.nanopub.org/nschema#')**
Namespace for *http://www.nanopub.org/nschema#*

nanopub.namespaces.**NPX = Namespace('http://purl.org/nanopub/x/')**
Namespace for *http://purl.org/nanopub/x/*

nanopub.namespaces.**ORCID = Namespace('https://orcid.org/')**
Namespace for *https://orcid.org/*

nanopub.namespaces.**PROV = Namespace('http://www.w3.org/ns/prov#')**
Namespace for *http://www.w3.org/ns/prov#*

CHAPTER

THIRTEEN

WELCOME TO NANOPUB'S DOCUMENTATION!

The nanopub library provides a high-level, user-friendly python interface for searching, publishing and retracting nanopublications.

Nanopublications are a formalized and machine-readable way of communicating the smallest possible units of publishable information. See *What are nanopublications?* for more information.

13.1 Setup

Install using pip:

```
pip install nanopub
```

To publish to the nanopub server you need to setup your profile. This allows the nanopub server to identify you. Run the following interactive command:

```
setup_nanopub_profile
```

This will add and store RSA keys to sign your nanopublications, publish a nanopublication with your name and ORCID iD to declare that you are using using these RSA keys, and store your ORCID iD to automatically add as author to the provenance of any nanopublication you will publish using this library.

13.2 Quick Start

13.2.1 Publishing nanopublications

```
import rdflib
from nanopub import Publication, NanopubClient

# Create the client (we use use_test_server=True to point to the test server)
client = NanopubClient(use_test_server=True)

# Either quickly publish a statement to the server
client.claim('All cats are gray')

# Or: 1. construct a desired assertion (a graph of RDF triples) using rdflib
my_assertion = rdflib.Graph()
my_assertion.add((rdflib.URIRef('www.example.org/timbernerslee'),
```

(continues on next page)

(continued from previous page)

```
                  rdflib.RDF.type,
                  rdflib.FOAF.Person))

# 2. Make a Publication object with this assertion
publication = Publication.from_assertion(assertion_rdf=my_assertion)

# 3. Publish the Publication object. The URI at which it is published is returned.
publication_info = client.publish(publication)
print(publication_info)
```

13.2.2 Searching for nanopublications

```
from nanopub import NanopubClient

# Create the client
client = NanopubClient()

# Search for all nanopublications containing the text 'fair'
results = client.find_nanopubs_with_text('fair')
for result in results:
    print(result)
```

13.2.3 Fetching nanopublications and inspecting them

```
from nanopub import NanopubClient

# Create the client
client = NanopubClient()

# Fetch the nanopublication at the specified URI
publication = client.fetch('http://purl.org/np/
↪RApJG4fwj0szOMBMiYGmYvd5MCtRle6VbwkMJUb1SxxDM')

# Print the RDF contents of the nanopublication
print(publication)

# Iterate through all triples in the assertion graph
for s, p, o in publication.assertion:
    print(s, p, o)
```

PYTHON MODULE INDEX

n

INDEX