pyLDAPI Documentation

Release 2.0.13

CSIRO Land and Water

Aug 09, 2021

Getting Started

CHAPTER 1

Welcome to pyLDAPI

The Python Linked Data API (pyLDAPI) is:

A very small module to add Linked Data API functionality to a Python FastAPI installation.

1.1 What is it?

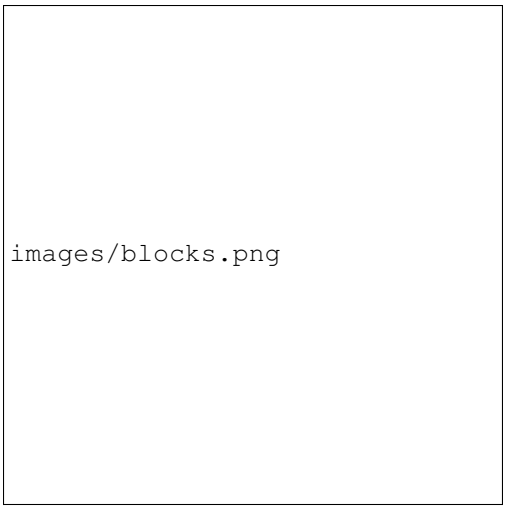
This module contains a small Python module which is intended to be added (imported) into a FastAPI (v4.x +) or Python Flask (v3.x) installation to add a small library of `Renderer` classes which can be used to handle requests and return responses in a manner consistent with Linked Data principles of operation.

The intention is to make it easy to "Linked Data-enable" web APIs.

An API using this module will get:

- **an *alt profile* for each endpoint that uses a `Renderer` class to return responses that the API delivers**
  - this is a *profile*, or *view* of the resource that lists all other available profiles
- **a *Register of Registers***
  - a start-up function that auto-generates a Register of Registers is run when the API is launched.
- a basic, over-writeable template for Registers' HTML & RDF
- **all of the functionality defined by the W3C's Content Negotiation by Profile specification**
  - to allow for requests of content that conform to data specifications and profiles

The main parts of pyLDAPI are as follows:



Web requests arrive at a Web Server, such as *Apache* or *nginx*, which then forwards (some of) them on to *FastAPI*, a Python web framework. FastAPI calls Python functions for web requests defined in a request/function mapping and may call pyLDAPI elements. FastAPI need not call pyLDAPI for all requests, just as Apache/nginx need not forward all web request to FastAPI. pyLDAPI may then draw on any Python data source, such as database APIs, and uses the *rdflib* Python module to formulate RDF responses.

1.2 Definitions

1.2.1 Alt Profile

The *model view* that lists all other views. This API uses the definition of *Alternate Profiles Data Model as an OWL ontology* presented at https://www.w3.org/TR/dx-prof-conneg/#altr-owl.

1.2.2 Linked Data Principles

The principles of making things available over the internet in both human and machine-readable forms. Codified by the World Wide Web Consortium. See https://www.w3.org/standards/semanticweb/data.

1.2.3 Model View

A set of properties of a Linked Data object codified according to a standard or profile of a standard.

1.2.4 Object

Any individual thing delivered according to *Linked Data* principles.

1.2.5 Register

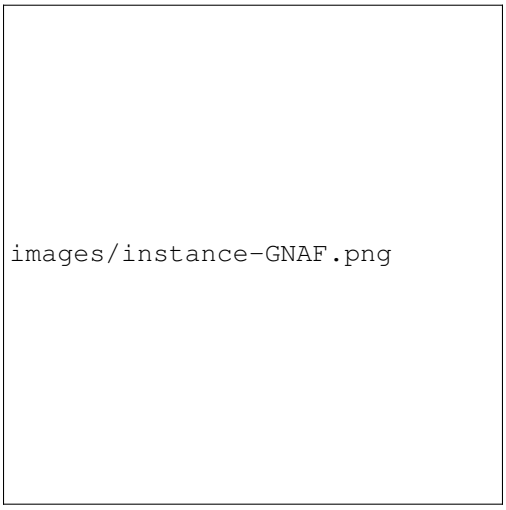
A simple listing of URIs of objects, delivered according to *Linked Data principles*.

1.2.6 Register of Registers

A *register* that lists all other registers which this API provides.

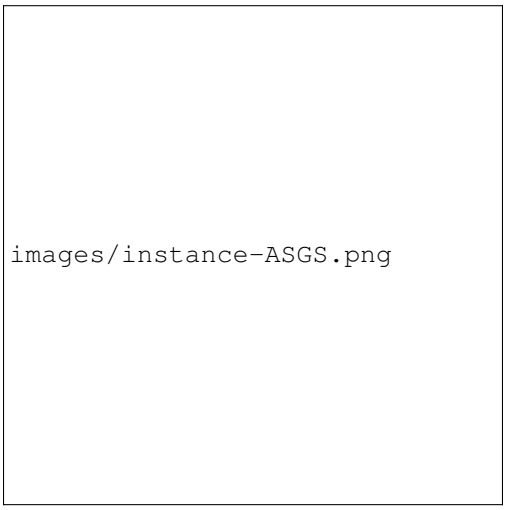
1.3 pyLDAPI in action

- **Register of Media Types**
  - https://w3id.org/mediatype/
- **Linked Data version of the Geocoded National Address File**
  - http://linked.data.gov.au/dataset/gnaf



Parts of the GNAF implementation

- **Geoscience Australia's Sites, Samples Surveys Linked Data API**
  - http://pid.geoscience.gov.au/sample/
- **Linked Data version of the Australian Statistical Geography Standard product**
  - http://linked.data.gov.au/dataset/asgs



Parts of the ASGS implementation

1.4 Documentation

Detailed documentation can be found at https://pyldapi.readthedocs.io/

1.5 Licence

This is licensed under GNU General Public License (GPL) v3.0. See the LICENSE deed for more details.

1.6 Contact

1.6.1 Dr Nicholas Car (lead)

Data Systems Architect
SURROUND Australia Pty Ltd
nicholas.car@surroundaustralia.com
https://orcid.org/0000-0002-8742-7730

1.6.2 Ashley Sommer (senior developer)

Informatics Software Engineer
CSIRO Land and Water
ashley.sommer@csiro.au

1.7 Related work

pyLDAPI Client

- *A Simple helper library for consuming registers, indexes, and instances of classes exposed via a pyLDAPI endpoint.*

1.8 Changelog

4.x

- Version 4+ uses FastAPI, not Flask. For Flask, use <=3.11

3.11

- tokens applied to Representations in Alternate View profile, not Profiles

3.0

- Content Negotiation specification by Profile supported
- replaced all references to “format” with “Media Type” and “view” with “profile”

- renamed class View to Profile
- added unit tests for all profile functions
- added unit tests for main ConnegP functions

1.8.1 Requirements

Note: To use the pyLDAPI module, a set of requirements must be met for the tool to work correctly.

Jinja2 Templates

Register

A `members.html` template is required to deliver a register of items.

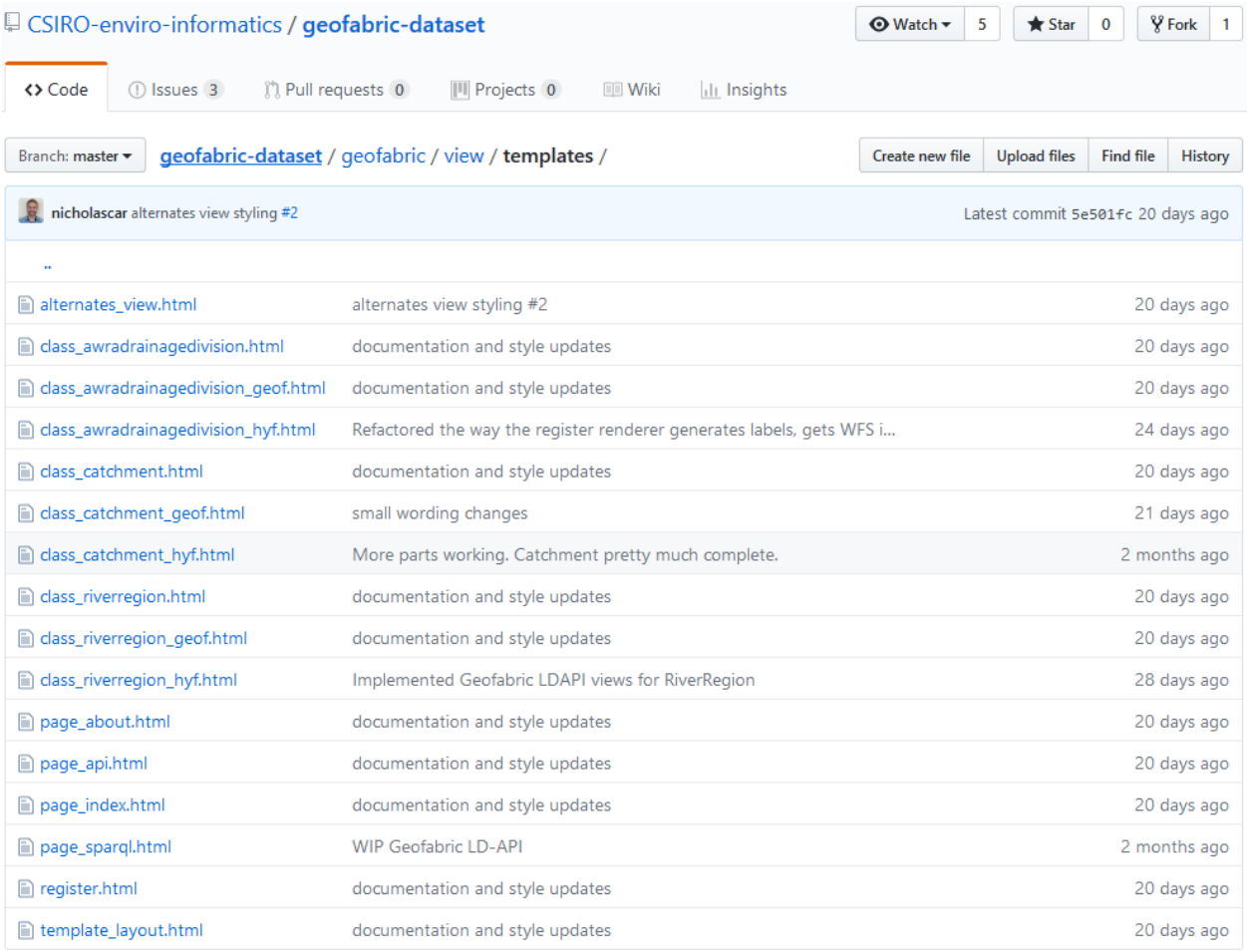
Alternates

An `alternates.html` template is required to deliver an *alternates view* of a register or instance of a class. Alternatively, you can specify a different template for the alternates view by passing an optional argument to the *pyldapi.Renderer.__init__()* as `alternates_template=`.

Class

A template for each class item in the dataset is required to render a class item.

Example: The online LD API for the Geofabric at geofabricld.net is exposing three class types, *Catchment*, *River Region* and *Drainage Division*. You can see in the image below showcasing the templates used for this API.



CSIRO-enviro-informatics / geofabric-dataset

Watch 5 | Star 0 | Fork 1

Code | Issues 3 | Pull requests 0 | Projects 0 | Wiki | Insights

Branch: master ▾ | geofabric-dataset / geofabric / view / templates / | Create new file | Upload files | Find file | History

nicholascar alternates view styling #2 — Latest commit 5e501fc 20 days ago

..		
alternates_view.html	alternates view styling #2	20 days ago
class_awradrainagedivision.html	documentation and style updates	20 days ago
class_awradrainagedivision_geof.html	documentation and style updates	20 days ago
class_awradrainagedivision_hyf.html	Refactored the way the register renderer generates labels, gets WFS i...	24 days ago
class_catchment.html	documentation and style updates	20 days ago
class_catchment_geof.html	small wording changes	21 days ago
class_catchment_hyf.html	More parts working. Catchment pretty much complete.	2 months ago
class_riverregion.html	documentation and style updates	20 days ago
class_riverregion_geof.html	documentation and style updates	20 days ago
class_riverregion_hyf.html	Implemented Geofabric LDAPI views for RiverRegion	28 days ago
page_about.html	documentation and style updates	20 days ago
page_api.html	documentation and style updates	20 days ago
page_index.html	documentation and style updates	20 days ago
page_sparql.html	WIP Geofabric LD-API	2 months ago
register.html	documentation and style updates	20 days ago
template_layout.html	documentation and style updates	20 days ago

Note: These are of course not the only Jinja2 templates that you will have. Other ones may include something like the API's home page, about page, etc. You can also see that there are more than one template for a specific class type in the image above. These different templates with *geof* and *hyf* are the different views for the specific class item. See `View` for more information.

See also:

See also the template information under the **Jinja2 Templates** section of the documentation for more information in regards to what variables are required to pass in to the required templates.

1.8.2 Installation

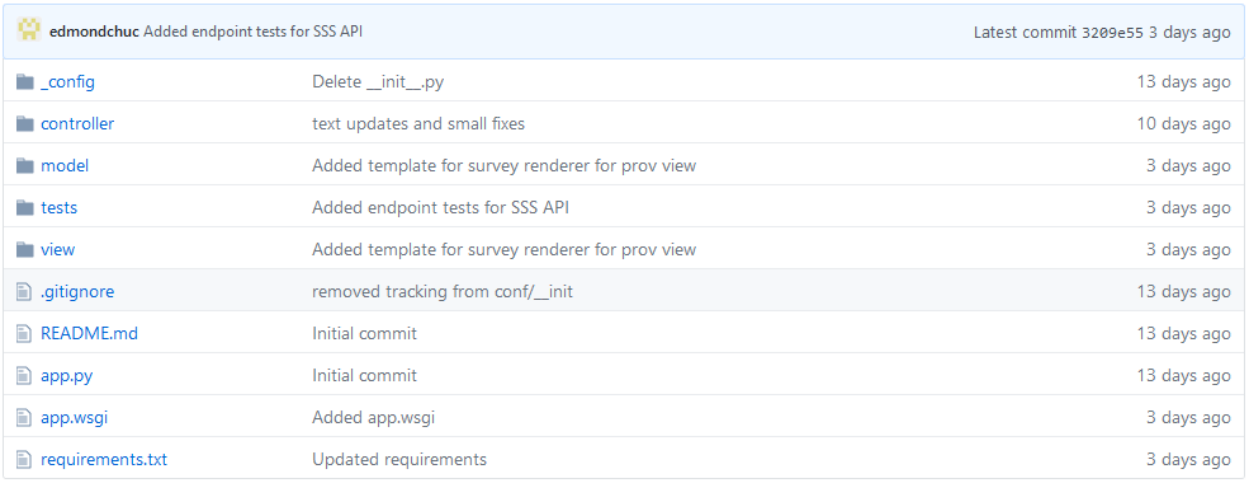
Attention: See *Requirements* before getting started and make sure the requirements are met.

To install, use Python's PyPI by invoking `pip install pyldapi` on the command line interface.

Now download the set of *Download Jinja Templates* and put them into a directory called `view/templates` in your Flask project.

Recommended project structure

We recommend a project structure as follows:



As shown in the image above, we recommend *this* model-view-controller architectural pattern for the project structure to maximise separation of concerns. The image above was taken from this repository.

The controller directory

The **controller** directory is used to declare the Flask routes to your python functions.

The model directory

The **model** directory is used to declare the Python files that manage the data of the API.

The view directory

The **view** directory contains the static content as well as the *required* Jinja2 templates for this API.

1.8.3 Indices and tables

- genindex
- modindex
- search

1.8.4 Members template

Example of a generic register template:

```
{% extends "layout.html" %}
{% block content %}
    <h1>{{ label }}</h1>
    <h2>Register View</h2>
```

(continues on next page)

(continued from previous page)

```
    {% for class in contained_item_classes %}
        <span><h3>Of <a href="{{ class }}">{{ class }}</a> class items</h3></span>
    {% endfor %}
    <table>
        <tr>
            <td style="vertical-align:top; width:500px;">
                <h3>Items in this Register</h3>
                <ul>
                {%- for item in register_items -%}
                    {%- if item is not string %}
                    <li class="no-line-height"><a href="{{ item[0] }}">{{ item[1] }}</
↪a></li>
                    {%- else %}
                    <li class="no-line-height"><a href="{{ item }}">{{ item.split('#
↪')[-1].split('/')[-1] }}</a></li>
                    {%- endif %}
                {%- endfor -%}
                </ul>
                {%  if pagination.links %}
                <h5>Paging</h5>
                {%  endif %}
                {{ pagination.links }}
            </td>
            <td style="vertical-align:top;">
                <h3>Alternate profiles</h3>
                <p>Different profiles of this register are listed at its <a href="{{␣
↪request.base_url }}?_profile=alternates">Alternate profiles</a> page.</p>
                <h3>Automated Pagination</h3>
                <p>To page through these items, use the query string arguments 'page'␣
↪for the page number and 'per_page' for the number of items per page. HTTP <code>Link
↪</code> headers of <code>first</code>, <code>prev</code>, <code>next</code> &amp;
↪<code>last</code> indicate URIs to the first, a previous, a next and the last page.
↪</p>
                <p>Example:</p>
                <pre>
                    {{ request.base_url }}?page=7&amp;per_page=50
                </pre>
                <p>Assuming 500 items, this request would result in a response with␣
↪the following Link header:</p>
                <pre>
                    Link:   &lt;{{ request.base_url }}?per_page=50&gt; rel="first",
                        &lt;{{ request.base_url }}?per_page=50&page=6&gt; rel="prev",
                        &lt;{{ request.base_url }}?per_page=50&page=8&gt; rel="next",
                        &lt;{{ request.base_url }}?per_page=50&page=10&gt; rel="last"
                </pre>
                <p>If you want to page the whole collection, you should start at
↪<code>first</code> and follow the link headers until you reach <code>last</code> or␣
↪until there is no <code>last</code> link given. You shouldn't try to calculate each
↪<code>page</code> query string argument yourself.</p>
            </td>
        </tr>
    </table>
{% endblock %}
```

Variables used by the register template:

```
render_template(
    self.register_template or 'register.html',          # the register template to use
    uri=self.uri,                                       # the URI requested
    label=self.label,                                   # The label of the Register
    comment=self.comment,                               # A description of the␣
↪Register
    contained_item_classes=self.contained_item_classes, # The list of URI strings of␣
↪each distinct class of item contained in this Register
    register_items=self.register_items,                 # The class items in this␣
↪Register
    page=self.page,                                     # The page number of this␣
↪current Register's instance
    per_page=self.per_page,                             # The number of class items␣
↪per page. Default is 20
    first_page=self.first_page,                         # deprecated, use pagination␣
↪instead
    prev_page=self.prev_page,                           # deprecated, use pagination␣
↪instead
    next_page=self.next_page,                           # deprecated, use pagination␣
↪instead
    last_page=self.last_page,                           # deprecated, use pagination␣
↪instead
    super_register=self.super_register,                 # A super-Register URI for␣
↪this register. Can be within this API or external
    pagination=pagination                               # pagination object from␣
↪module flask_paginate
)
```

See `RegisterRenderer` for an example on how to render the register profile.

1.8.5 Alternates template

Example of a generic alternates template:

```
{% extends "layout.html" %}
{% block content %}
    <h1>{{ register_name }} Linked Data API</h1>
    {% if class_uri %}
        <h3>Alternates view of a <a href="{{ class_uri }}">{{ class_uri }}</a></h3>
    {% else %}
        <h3>Alternates view</h3>
    {% endif %}
    {% if instance_uri %}
        <h3>Instance <a href="{{ instance_uri }}">{{ instance_uri }}</a></h3>
    {% endif %}
    <p>Default profile: <a href="{{ request.base_url }}?_profile={{ default_profile_
↪token }}">{{ default_profile_token }}</a></p>
    <table class="pretty">
    <tr><th>View</th><th>Formats</th><th>View Desc.</th><th>View Namespace</th></tr>
    {% for v, vals in profiles.items() %}
            <tr>
                <td><a href="{{ request.base_url }}?_profile={{ v }}">{{ v }}</a></td>
                <td>
                {% for f in vals['formats'] %}
                    <a href="{{ request.base_url }}?_profile={{ v }}&_format={{ f }}">
↪{{ f }}</a>
```

(continues on next page)

(continued from previous page)

```
                    {% if loop.index != vals['formats']|length %}<br />{% endif %}
                {% endfor %}
                </td>
                <td>{{ vals['namespace'] }}</td>
                <td>{{ vals['comment'] }}</td>
            </tr>
    {% endfor %}
    </table>
{% endblock %}
```

The alternates profile template is shared for both a Register's alternates profile as well as a class instance item's alternates profile. In any case, since a `RegisterRenderer` class and a *Custom Renderer* class both inherit from the base class *Renderer*, then they can both easily render the alternates profile by calling the base class' `pyldapi.Renderer.render_alternates_profile()` method. One distinct difference is that pyLDAPI will handle the alternates profile automatically for a `RegisterRenderer` whereas a *Custom Renderer* will have to explicitly call the `pyldapi.Renderer.render_alternates_profile()`.

Example usage for a *Custom Renderer*:

```
# context: inside a 'custom' Renderer class which inherits from pyldapi.Renderer

# this is an implementation of the abstract render() of the base class Renderer
def render(self):
    # ...
    if self.profile == 'alternates':
        return self.render_alternates_profile() # render the alternates profile for this class instance
    # ...
```

1.8.6 Class template

Example of a class item template customised for the mediatypes dataset:

```
{% extends "layout.html" %}

    {% block content %}
    <h1>{{ mediatype }}</h1>
    <h3><a href="{{ request.values.get('uri') }}">{{ request.values.get('uri') }}</a></h3>
    <h4>Source:</h4>
    <ul>
        <li><a href="https://www.iana.org/assignments/media-types/{{ mediatype }}">https://www.iana.org/assignments/media-types/{{ mediatype }}</a></li>
    </ul>
    {% if deets['contributors'] is not none %}
    <h4>Contributors:</h4>
    <ul>
    {% for contributor in deets['contributors'] %}
        <li><a href="{{ contributor }}">{{ contributor }}</a></li>
    {% endfor %}
    </ul>
    {% endif %}
    <h3>Other profiles, formats and languages:</h3>
    <ul><li><a href="{{ request.base_url }}?uri={{ request.values.get('uri') }}&_view=alternates">Alternate Views</a></li></ul>
```

(continues on next page)

(continued from previous page)

```
{% endblock %}
```

Variables used by the class instance template:

This will be called within a custom Renderer class' `render()`. See *Custom Renderer*.

```
return render_template(
    'mediatype-en.html',     # the class item template
    deets=deets,             # a python dict containing keys *label* and␣
↪*contributors* to its respective values.
    mediatype=mediatype      # the mediatype class instance item name
)
```

1.8.7 Download Jinja Templates

This page contains a few general templates that are likely to be used in a pyLDAPI instance. They are provided to ease the initial development efforts with pyLDAPI.

All Jinja2 templates should use the Jinja2 `extends` keyword to extend the generic `page.html` to reduce duplicated HTML code.

Page

This template contains the persistent HTML code like the product's logo, the navigation bar and the footer. All other persistent things should go in this template.

`page.html`

Index

The home page of the pyLDAPI instance. Add whatever you like to this page.

`index.html`

Register

The register template lists all the items in a register.

`register.html`

Instance

The instance template presents the basic metadata of an instance item.

`instance.html`

Alternates

The alternates template renders a list of alternate views and formats for a register or instance item.

`alternates.html`

1.8.8 Renderer

class pyldapi.**Renderer**(*request*, *instance_uri*, *profiles*, *default_profile_token*)

Abstract class as a parent for classes that validate the profiles & mediatypes for an API-delivered resource (typically either registers or objects) and also creates an 'alternates profile' for them, based on all available profiles & mediatypes.

__init__(*request*, *instance_uri*, *profiles*, *default_profile_token*)

Constructor

Parameters

- **request** (flask.request) – Flask request object that triggered this class object's creation.
- **instance_uri** (*str*) – The URI that triggered this API endpoint (can be via redirects but the main URI is needed).
- **profiles** (dict (of View class objects)) – A dictionary of profiles available for this resource.
- **default_profile_token** – The ID of the default profile (key of a profile in the dictionary of :class:

.*Profile* objects) :type default_profile_token: str (a key in profiles) :param alternates_template: The Jinja2 template to use for rendering the HTML *alternates view*. If None, then it will default to try and use a template called alternates.html. :type alternates_template: str

See also:

See the View class on how to create a dictionary of profiles.

render(*alt_template: str = 'alt.html'*, *additional_alt_template_context=None*, *alt_template_context_replace=False*)

Use the received profile and mediatype to create a response back to the client.

TODO: Ashley, are you able to update this description with your new changes please? What is the method for rendering other profiles now? - Edmond

This is an abstract method.

Note: The *pyldapi.Renderer.render* requires you to implement your own business logic to render

---

custom responses back to the client using flask.render_template() or flask.Response object.

Example Implementation of pyldapi.Renderer.render()

```
# context: a custom Renderer class which inherits from pyldapi.Renderer

def render(self):
    if self.site_no is None:
        return Response('Site {} not found.'.format(self.site_no), status=404,␣
↪mimetype='text/plain')
    if self.view == 'alternates':
        # call the base class' render alternates view method
        return self._render_alternates_view()
```

(continues on next page)

(continued from previous page)

```
    elif self.view == 'pdm':
        # render the view with the token 'pdm' as text/html
        if self.format == 'text/html':
            # you need to define your own self.export_html()
            return self.export_html(model_view=self.view)
        else:
            # you need to define your own self.export_rdf()
            return Response(self.export_rdf(self.view, self.format), mimetype=self.
↪format)
    elif self.view == 'nemsr':
        # you need to define your own self.export_nemsr_geojson()
        return self.export_nemsr_geojson()
```

The example code determines the response based on the set *view* and *format* of the object.

See also:

See *Custom Renderer* for implementation details for `Renderer`.

1.8.9 View

Example Usage

A dictionary of views:

```
views = {
    'csirov3': View(
        'CSIRO IGSN View',
        'An XML-only metadata schema for descriptive elements of IGSNs',
        ['text/xml'],
        'text/xml',
        namespace='https://confluence.csiro.au/display/AusIGSN/
↪CSIRO+IGSN+IMPLEMENTATION'
    ),

    'prov': View(
        'PROV View',
        "The W3C's provenance data model, PROV",
        ["text/html", "text/turtle", "application/rdf+xml", "application/rdf+json"],
        "text/turtle",
        namespace="http://www.w3.org/ns/prov/"
    ),
}
```

A dictionary of views are generally intialised in the constructor of a specialised *ClassRenderer*. This ClassRenderer inherits from `Renderer`

See also:

See *Custom Renderer* for more information.

1.8.10 Register Renderer

Example Usage

This example contains a Flask route `/sample/` which maps to the *Register of Samples*. We use the `RegisterRenderer` to create the Register and return a response back to the client. The code of interest is highlighted on lines 20-30.

```
1  @classes.route('/sample/')
2  def samples():
3      """
4      The Register of Samples
5      :return: HTTP Response
6      """
7
8      # get the total register count from the XML API
9      try:
10         r = requests.get(conf.XML_API_URL_TOTAL_COUNT)
11         no_of_items = int(r.content.decode('utf-8').split('<RECORD_COUNT>')[1].split('
   ↪</RECORD_COUNT>')[0])
12
13         page = request.values.get('page') if request.values.get('page') is not None␣
   ↪else 1
14         per_page = request.values.get('per_page') if request.values.get('per_page')␣
   ↪is not None else 20
15         items = _get_items(page, per_page, "IGSN")
16     except Exception as e:
17         print(e)
18         return Response('The Samples Register is offline', mimetype='text/plain',␣
   ↪status=500)
19
20     r = pyldapi.RegisterRenderer(
21         request,
22         request.url,
23         'Sample Register',
24         'A register of Samples',
25         items,
26         [conf.URI_SAMPLE_CLASS],
27         no_of_items
28     )
29
30 return r.render()
```

1.8.11 Register of Registers Renderer (RoR)

Note: To use this, ensure *`pyldapi.setup()`* is called before calling Flask's `app.run()`. See *RoR Setup* for more information.

Example Usage

A Flask route at the root of the application serving the *Register of Registers* page to the client.

```
@app.route('/')
def index():
```

(continues on next page)

(continued from previous page)

```
    cofc = RegisterOfRegistersRenderer(
        request,
        API_BASE,
        "Register of Registers",
        "A register of all of my registers.",
        "./cofc.ttl"
    )
    return cofc.render()
```

1.8.12 RoR Setup

pyldapi.**setup**(*app*, *api_home_dir*, *api_uri*)

This is used to set up the RegisteC of CegistersRenderer for this pyLDAPI instance.

Note: This must run before Flask's app.run() like this: pyldapi.setup(app, '.', conf.URI_BASE). See the example below.

Parameters

- **app** (flask.Flask) – The Flask app containing this pyLDAPI instance.
- **api_home_dir** (*str*) – The path of the API's hom directory.
- **api_uri** (*str*) – The URI base of the API.

Returns None

Return type None

Example Usage

```
from flask import Flask
from pyldapi import setup as pyldapi_setup

API_BASE = 'http://127.0.0.1:8081'
app = Flask(__name__)

if __name__ == "__main__":
    pyldapi_setup(app, '.', API_BASE)
    app.run("127.0.0.1", 8081, debug=True, threaded=True, use_reloader=False)
```

1.8.13 Exceptions

exception pyldapi.exceptions.**CofCTtlError**

TODO: Ashley add docstring for documentation

exception pyldapi.exceptions.**PagingError**

TODO: Ashley add docstring for documentation

exception pyldapi.exceptions.**ProfilesMediatypesException**

TODO: Ashley add docstring for documentation

1.8.14 Custom Renderer

In this example, we are creating a custom *Renderer* class by inheritance to cater for a media type instance. More information about this code can be found at this repository.

- The interest for `View` declarations are on lines 10-19.
- On line 20-25, we pass we call the `__init__()` of the super class, passing in the list of `View` objects and some other arguments.
- Lines 27-57 demonstrate how to implement the abstract *pyldapi.Renderer.render()* and how it works in tandem with the list of `View` objects.

Note: The focus here is to demonstrate how to create a custom *Renderer* class, defining a custom `render()` method and defining a list of `View` objects.

```
1  from flask import Response, render_template
2  from SPARQLWrapper import SPARQLWrapper, JSON
3  from rdflib import Graph, URIRef, Namespace, RDF, RDFS, XSD, OWL, Literal
4  from pyldapi import Renderer, Profile
5  import _conf as conf
6
7
8  class MediaTypeRenderer(Renderer):
9      def __init__(self, request, instance_uri):
10         profiles = {
11             'mt': Profile(
12                 'Mediatype View',
13                 'Basic properties of a Media Type, as recorded by IANA',
14                 ['text/html'] + Renderer.RDF_MEDIA_TYPES,
15                 'text/turtle',
16                 languages=['en', 'pl'],
17                 uri='http://test.linked.data.gov.au/def/mt#'
18             )
19         }
20         super(MediaTypeRenderer, self).__init__(
21             request,
22             instance_uri,
23             profiles,
24             'mt'
25         )
26
27     def render(self):
28         if hasattr(self, 'vf_error'):
29             return Response(self.vf_error, status=406, mimetype='text/plain')
30         else:
31             if self.profile == 'alternates':
32                 return self._render_alternates_profile()
33             elif self.profile == 'mt':
34                 if self.format in Renderer.RDF_MEDIA_TYPES:
35                     rdf = self._get_instance_rdf()
36                     if rdf is None:
37                         return Response('No triples contain that URI as subject',␣
   ↪status=404, mimetype='text/plain')
38                     else:
39                         return Response(rdf, mimetype=self.format)
40                 else:  # only the HTML format left
```

(continues on next page)

(continued from previous page)

```
                    deets = self._get_instance_details()
                    if deets is None:
                        return Response('That URI yielded no data', status=404,␣
↪mimetype='text/plain')
                    else:
                        mediatype = self.instance_uri.replace('%2B', '+').replace('%2F
↪', '/').split('/mediatype/')[1]
                        if self.language == 'pl':
                            return render_template(
                                'mediatype-pl.html',
                                deets=deets,
                                mediatype=mediatype
                            )
                        else:
                            return render_template(
                                'mediatype-en.html',
                                deets=deets,
                                mediatype=mediatype
                            )

    def _get_instance_details(self):
        sparql = SPARQLWrapper(conf.SPARQL_QUERY_URI, returnFormat=JSON)
        q = '''
            PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
            PREFIX dct:  <http://purl.org/dc/terms/>
            SELECT *
            WHERE {{
                <{0[uri]}>  rdfs:label ?label .
                OPTIONAL {{ <{0[uri]}> dct:contributor ?contributor . }}
            }}
        '''.format({'uri': self.instance_uri})
        sparql.setQuery(q)
        d = sparql.query().convert()
        d = d.get('results').get('bindings')
        if d is None or len(d) < 1:  # handle no result
            return None

        label = ''
        contributors = []
        for r in d:
            label = str(r.get('label').get('value'))
            contributors.append(str(r.get('contributor').get('value')))

        return {
            'label': label,
            'contributors': contributors
        }

    def _get_instance_rdf(self):
        deets = self._get_instance_details()

        g = Graph()
        DCT = Namespace('http://purl.org/dc/terms/')
        g.bind('dct', DCT)
        me = URIRef(self.instance_uri)
        g.add((me, RDF.type, DCT.FileFormat))
        g.add((
```

(continues on next page)

(continued from previous page)

```
            me,
            OWL.sameAs,
            URIRef(self.instance_uri.replace('https://w3id.org/mediatype/', 'https://
↪www.iana.org/assignments/media-types/'))
        ))
        g.add((me, RDFS.label, Literal(deets.get('label'), datatype=XSD.string)))
        source = 'https://www.iana.org/assignments/media-types/' + self.instance_uri.
↪replace('%2B', '+').replace('%2F', '/').split('/mediatype/')[1]
        g.add((me, DCT.source, URIRef(source)))
        if deets.get('contributors') is not None:
            for contributor in deets.get('contributors'):
                g.add((me, DCT.contributor, URIRef(contributor)))

        if self.format in ['application/rdf+json', 'application/json']:
            return g.serialize(format='json-ld')
        else:
            return g.serialize(format=self.format)
```

1.8.15 A Toy pyLDAPI Example Usage

Warning: TODO: Explain the import statements and the example code.

Here is a very simple example usage of pyLDAPI.

Python Module Index

p

Index

Symbols

C

P

R

S