PyOData Documentation

Release 1.2.1

SAP

Sep 16, 2023

Contents

Python agnostic implementation of OData client library.

CHAPTER 1

Supported features

- OData V2

CHAPTER 2

Basic usage

The only thing you need to do is to import the **pyodata** Python module and provide an object implementing interface compatible with Session from the library Requests.

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'
HTTP_LIB = requests.Session()

northwind = pyodata.Client(SERVICE_URL, HTTP_LIB)

for customer in northwind.entity_sets.Customers.get_entities().execute():
    print(customer.CustomerID, customer.CompanyName)
```

CHAPTER 3

The User Guide

3.1 Initialization

PyOData requires an external HTTP library which has API compatible with Session from Requests.

3.1.1 Get the service

Basic initialization which is going to work for everybody:

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

northwind = pyodata.Client(SERVICE_URL, requests.Session())
```

3.1.2 Get the service proxy client for an OData service requiring sap-client parameter

This is a sample when it is necessary to specify sap-client:

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

session = requests.Session()
param = {'sap-client': '510'}
session.params = param
northwind = pyodata.Client(SERVICE_URL, session)
```

3.1.3 Get the service proxy client for an OData service requiring authentication

Let's assume you need to work with a service at the URL *https://odata.example.com/Secret.svc* and User ID is 'MyUser' with the password 'MyPassword'.

```
import pyodata
import requests

SERVICE_URL = 'https://odata.example.com/Secret.svc'

session = requests.Session()
session.auth = ('MyUser', 'MyPassword')

theservice = pyodata.Client(SERVICE_URL, session)
```

3.1.4 Get the service proxy client for an OData service requiring Certificate authentication

Let's assume your service requires certificate based authentication and you are able to export the certificate into the file *mycert.p12*. You need to split the certificate into public key, private key and certificate authority key.

The following steps has been verified on Fedora Linux and Mac OS.

```
openssl pkcs12 -in mycert.p12 -out ca.pem -cacerts -nokeys
openssl pkcs12 -in mycert.p12 -out client.pem -clcerts -nokeys
openssl pkcs12 -in mycert.p12 -out key.pem -nocerts
openssl rsa -in key.pem -out key_decrypt.pem
```

You can verify your steps by curl:

```
curl --key key_decrypt.pem --cacert ca.pem --cert client.pem -k 'SERVICE_URL/$metadata
↪'
```

Python client initialization:

```
import pyodata
import requests

SERVICE_URL = 'https://odata.example.com/Secret.svc'

session = requests.Session()
session.verify = 'ca.pem'
session.cert = ('client.pem', 'key_decrypt.pem')

client = pyodata.Client(SERVICE_URL, session)
```

For more information on client side SSL cerificationcas, please refer to this [gist](https://gist.github.com/mtigas/952344).

3.1.5 Get the service with local metadata

It may happen that you will have metadata document for your service downloaded in a local file and you will want to initialize the service proxy from this file. In such a case you can provide content of the file as the named argument *metadata*. Please, make sure you provide *bytes* and not *str* (required by the xml parser lxml).

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

with open('/the/file/path.xml', 'rb') as mtd_file:
    local_metadata = mtd_file.read()

northwind = pyodata.Client(SERVICE_URL, requests.Session(), metadata=local_metadata)
```

3.1.6 Dealing with errors during parsing metadata

In the case where you need to consume a service which has not fully valid metadata document and is not under your control, you can configure the metadata parser to try to recover from detected problems.

Parser recovery measures include actions such as using a stub entity type if the parser cannot find a referenced entity type. The stub entity type allows the parser to continue processing the given metadata but causes fatal errors when accessed from the client.

Class config provides easy to use wrapper for all parser configuration. These are:

- XML namespaces
- **Parser policies (how parser act in case of invalid XML tag). We now support three types of policies:**
  - Policy fatal - the policy raises exception and terminates the parser
  - Policy warning - the policy reports the detected problem, executes a fallback code and then continues normally
  - Policy ignore - the policy executes a fallback code without reporting the problem and then continues normally

Parser policies can be specified individually for each XML tag (See enum ParserError for more details). If no policy is specified for the tag, the default policy is used.

For parser to use your custom configuration, it needs to be passed as an argument to the client.

```
import pyodata
from pyodata.v2.model import PolicyFatal, PolicyWarning, PolicyIgnore, ParserError,␣
↪Config
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

namespaces = {
    'edmx': 'customEdmxUrl.com',
    'edm': 'customEdmUrl.com'
}

custom_config = Config(
    xml_namespaces=namespaces,
    default_error_policy=PolicyFatal(),
    custom_error_policies={
         ParserError.ANNOTATION: PolicyWarning(),
         ParserError.ASSOCIATION: PolicyIgnore()
    })
```

(continues on next page)

(continued from previous page)

```
northwind = pyodata.Client(SERVICE_URL, requests.Session(), config=custom_config)
```

Additionally, Schema class has Boolean atribute 'is_valid' that returns if the parser encountered errors. It's value does not depends on used Parser policy.

```
northwind.schema.is_valid
```

3.1.7 Prevent substitution by default values

Per default, missing properties get filled in by type specific default values. While convenient, this throws away the knowledge of whether a value was missing in the first place. To prevent this, the class config mentioned in the section above takes an additional parameter, *retain_null*.

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

northwind = pyodata.Client(SERVICE_URL, requests.Session(), config=pyodata.v2.model.
↪Config(retain_null=True))

unknown_shipped_date = northwind.entity_sets.Orders_Qries.get_entity(OrderID=11058,
                                                                     CompanyName=
↪'Blauer See Delikatessen').execute()

print(
    f'Shipped date: {"unknown" if unknown_shipped_date.ShippedDate is None else␣
↪unknown_shipped_date.ShippedDate}')
```

Changing *retain_null* to *False* will print *Shipped date: 1753-01-01 00:00:00+00:00*.

3.1.8 Set custom namespaces (Deprecated - use config instead)

Let's assume you need to work with a service which uses namespaces not directly supported by this library e. g. ones hosted on private urls such as *customEdmxUrl.com* and *customEdmUrl.com*:

```
import pyodata
import requests

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

namespaces = {
    'edmx': 'customEdmxUrl.com'
    'edm': 'customEdmUrl.com'
}

northwind = pyodata.Client(SERVICE_URL, requests.Session(), namespaces=namespaces)
```

3.2 Querying

3.2.1 Get one entity identified by its key value

Get employee identified by 1 and print employee first name:

```
employee1 = northwind.entity_sets.Employees.get_entity(1).execute()
print(employee1.FirstName)
```

3.2.2 Get one entity identified by its key value which is not scalar

Get number of orderd units in the order identified by ProductID 42 and OrderID 10248:

```
order = northwind.entity_sets.Order_Details.get_entity(OrderID=10248, ProductID=42).
↪execute()
print(order.Quantity)
```

3.2.3 Get all entities of an entity set

Print unique identification (Id) and last name of all employees:

```
employees = northwind.entity_sets.Employees.get_entities().select('EmployeeID,LastName
↪').execute()
for employee in employees:
    print(employee.EmployeeID, employee.LastName)
```

3.2.4 Get entities matching a filter

Print unique identification (Id) of all employees with name John Smith:

```
smith_employees_request = northwind.entity_sets.Employees.get_entities()
smith_employees_request = smith_employees_request.filter("FirstName eq 'John' and␣
↪LastName eq 'Smith'")

for smith in smith_employees_request.execute():
    print(smith.EmployeeID)
```

3.2.5 Get entities matching a filter in more Pythonic way

Print unique identification (Id) of all employees with name John Smith:

```
from pyodata.v2.service import GetEntitySetFilter as esf

smith_employees_request = northwind.entity_sets.Employees.get_entities()
smith_employees_request = smith_employees_request.filter(esf.and_(
                                                    smith_employees_request.
↪FirstName == 'Jonh',
                                                    smith_employees_request.
↪LastName == 'Smith'))
for smith in smith_employees_request.execute():
    print(smith.EmployeeID)
```

3.2.6 Get entities matching a filter in ORM style

Print unique identification (Id) of all employees with name John Smith:

```
from pyodata.v2.service import GetEntitySetFilter as esf

smith_employees_request = northwind.entity_sets.Employees.get_entities()
smith_employees_request = smith_employees_request.filter(FirstName="John", LastName=
↪"Smith")
for smith in smith_employees_request.execute():
    print(smith.EmployeeID)
```

3.2.7 Get entities matching a complex filter in ORM style

Print unique identification (Id) of all employees with name John Smith:

```
from pyodata.v2.service import GetEntitySetFilter as esf

smith_employees_request = northwind.entity_sets.Employees.get_entities()
smith_employees_request = smith_employees_request.filter(FirstName__contains="oh",␣
↪LastName__startswith="Smi")
for smith in smith_employees_request.execute():
    print(smith.EmployeeID)
```

3.2.8 Get a count of entities

Print a count of all employees:

```
count = northwind.entity_sets.Employees.get_entities().count().execute()
print(count)
```

Print all employees and their count:

```
employees = northwind.entity_sets.Employees.get_entities().count(inline=True).
↪execute()
print(employees.total_count)

for employee in employees:
    print(employee.EmployeeID, employee.LastName)
```

3.2.9 Get a count of entities via navigation property

Print a count of all orders associated with Employee 1:

```
count = northwind.entity_sets.Employees.get_entity(1).nav('Orders').get_entities().
↪count().execute()
print(count)
```

Print all orders associated with Employee 1 and their count:

```
orders = northwind.entity_sets.Employees.get_entity(1).nav('Orders').get_entities().
↪count(inline=True).execute()
print(orders.total_count)

for order in orders:
    print(order.OrderID, order.ProductID)
```

3.2.10 Use non-standard OData URL Query parameters

Sometimes services implement extension to OData model and require addition URL query parameters. In such a case, you can enrich HTTP request made by pyodata with these parameters by the method *custom(name: str, value: str)*.

```
employee = northwind.entity_sets.Employees.get_entity(1).custom('sap-client', '100').
↪execute()
```

```
employees = northwind.entity_sets.Employees.get_entities().custom('sap-client', '100
↪').custom('$skiptoken', 'ABCD').top(10).execute()
```

3.2.11 Encode OData URL Path

By default the resource paths of requests are percent encoded. However if this is not what your API expects, you can disable the encoding with the variable encode_path by setting it to False.

```
employee = northwind.entity_sets.Employees.get_entity(1, encode_path=False).execute()
```

3.2.12 Query server-side paginations using the __next field

Response may contains ony partial listings of the Collection. In this case, "__next" name/value pair is included, where the value is a URI which identifies the next partial set of entities.

```
employees = northwind.entity_sets.Employees.get_entities().select('EmployeeID,LastName
↪').execute()
while True:
    for employee in employees:
     print(employee.EmployeeID, employee.LastName)

    # Stop if server has no more entities left
    if employees.next_url is None:
      break

    # We got a partial answer - continue with next page
    employees = northwind.entity_sets.Employees.get_entities().next_url(employees.
↪next_url).execute()
```

3.3 Creating

The create action executes the HTTP method POST which is usually protected by CSRF and therefore you must make some effort to initialize your HTTP Session to send POST requests acceptable by the remote server.

Let's assume you use the python library Requests

```
import pyodata
import requests

SERVICE_URL = 'http://example.io/TheServiceRoot/'

session = requests.Session()
response = session.head(SERVICE_URL, headers={'x-csrf-token': 'fetch'})
token = response.headers.get('x-csrf-token', '')
session.headers.update({'x-csrf-token': token})

theservice = pyodata.Client(SERVICE_URL, session)
```

3.3.1 Create an entity with a complex type property

You need to use the method set which accepts key value parameters:

```
employee_data = {
    'FirstName': 'Mark',
    'LastName': 'Goody',
    'Address': {
        'HouseNumber': 42,
        'Street': 'Paradise',
        'City': 'Heaven'
    }
}

create_request = northwind.entity_sets.Employees.create_entity()
create_request.set(**employee_data)

new_employee_entity = create_request.execute()
```

or you can do it explicitly:

```
create_request = northwind.entity_sets.Employees.create_entity()
create_request.set(
    FirstName='Mark',
    LastName='Goody',
    Address={
        'HouseNumber': 42,
        'Street': 'Paradise',
        'City': 'Heaven'
    }
)

new_employee_entity = request.execute()
```

3.4 Updating

To update an entity, you must create an updated request, set properties to the values you want and execute the request. The library will send an HTTP PATCH request to the remote service.

```
import pyodata
import requests
```

(continues on next page)

(continued from previous page)

```
SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

northwind = pyodata.Client(SERVICE_URL, requests.Session())

update_request = northwind.entity_sets.Customers.update_entity(CustomerID='ALFKI')
update_request.set(CompanyName='Alfons Kitten')
update_request.execute()
```

In the case the service you are dealing with requires PUT method, you have two options.

The first option allows you to change the used HTTP method for a single call via the key word parameter *method* of the method *update_entity*.

```
update_request = northwind.entity_sets.Customers.update_entity(CustomerID='ALFKI',␣
↪method='PUT')
```

If you need to run more update requests for different entity sets and all of them must be *PUT*, then you can consider setting the default service's update method to *PUT*.

```
northwind.config['http']['update_method'] = 'PUT'
```

3.4.1 Encode OData URL Path

By default the resource paths of requests are percent encoded. However if this is not what your API expects, you can disable the encoding with the variable encode_path by setting it to False.

```
update_request = northwind.entity_sets.Customers.update_entity(CustomerID='ALFKI',␣
↪encode_path=False)
```

3.5 Deleting

The delete action executes the HTTP method DELETE which is usually protected by CSRF and therefore you must make some effort to initialize your HTTP Session to send DELETE requests acceptable by the remote server.

Let's assume you use the python library Requests

```
import pyodata
import requests

SERVICE_URL = 'http://example.io/TheServiceRoot/'

session = requests.Session()
response = session.head(SERVICE_URL, headers={'x-csrf-token': 'fetch'})
token = response.headers.get('x-csrf-token', '')
session.headers.update({'x-csrf-token': token})

theservice = pyodata.Client(SERVICE_URL, session)
```

3.5.1 Deleting an entity

You can either delete entity by passing its PropertyRef value to the delete function

```
request = service.entity_sets.Employees.delete_entity(23)
request.execute()
```

or by passing the EntityKey object

```
key = EntityKey(service.schema.entity_type('Employee'), ID=23)
request = service.entity_sets.Employees.delete_entity(key=key)
request.execute()
```

3.5.2 Encode OData URL Path

By default the resource paths of requests are percent encoded. However if this is not what your API expects, you can disable the encoding with the variable encode_path by setting it to False.

```
request = service.entity_sets.Employees.delete_entity(key=key, encode_path=False)
```

3.6 Function imports

3.6.1 Calling a function import

```
products = northwind.functions.GetProductsByRating.parameter('rating', 16).execute()

for prod in products:
    print(prod)
```

3.7 Metadata evaluation

By default, the client makes sure that references to properties, entities and entity sets are pointing to existing elements.

The most often problem that we had to deal with was an invalid *ValueList* annotation pointing to a non-existing property.

To enable verification of service definition, the client instance of the class *Service* publishes the property *schema* which returns an instance of the class *Schema* from the module *pyodata.v2.model* and it contains parsed *$metadata*.

3.7.1 List of the defined EntitySets

If you need to iterate over all EntitySets:

```
for es in service.schema.entity_sets:
     print(es.name)
```

or if you just need the list of EntitySet names:

```
entity_set_names = [es.name for es in service.schema.entity_sets]
```

3.7.2 Property has this label

```
assert northwind.schema.entity_type('Customer').proprty('CustomerID').label ==
↪'Identifier'
```

3.7.3 Property has a value helper

```
assert northwind.schema.entity_type('Customer').proprty('City').value_helper is not␣
↪None
```

3.8 Advanced usage

3.8.1 Batch requests

Example of batch request that contains 3 simple entity queries:

```
batch = northwind.create_batch()

batch.add_request(northwind.entity_sets.Employees.get_entity(108))
batch.add_request(northwind.entity_sets.Employees.get_entity(234))
batch.add_request(northwind.entity_sets.Employees.get_entity(23))

response = batch.execute()

print(response[0].EmployeeID, response[0].LastName)
print(response[1].EmployeeID, response[1].LastName)
print(response[2].EmployeeID, response[2].LastName)
```

Example of batch request that contains simple entity query as well as changest consisting of two requests for entity modification:

```
batch = northwind.create_batch()

batch.add_request(northwind.entity_sets.Employees.get_entity(108))

changeset = northwind.create_changeset()

changeset.add_request(northwind.entity_sets.Employees.update_entity(45).set(LastName=
↪'Douglas'))

batch.add_request(changeset)

response = batch.execute()

print(response[0].EmployeeID, response[0].LastName)
```

3.8.2 Error handling

PyOData returns *HttpError* when the response code does not match the expected code. Basically the exception is raised for all status codes >= 400 and its instances have the property response which holds return value of the library making HTTP requests.

For example, the following code show how to print out error details if Python Requests is used as the HTTP communication library.

```
try:
    new_data = create_request.execute()
except pyodata.exceptions.HttpError as ex:
    print(ex.response.text)
```

In the case you know the implementation of back-end part and you want to show accurate errors reported by your service, you can replace *HttpError* by your sub-class *HttpError* by assigning your type into the class member *VendorType* of the class *HttpError*.

```
from pyodata.exceptions import HttpError

class MyHttpError(HttpError):

    def __init__(self, message, response):
        super(MyHttpError, self).__init__('Better message', response)


HttpError.VendorType = MyHttpError
```

The class *pyodata.vendor.SAP.BusinessGatewayError* is an example of such an HTTP error handling.

3.8.3 Enable Logging

The library uses Python logging without own handler, so to enable logging it is enough to set log level to the desired value.

```
import logging

logging.basicConfig()
root_logger = logging.getLogger()
root_logger.setLevel(logging.DEBUG)
```

3.8.4 Dynamically referenced EntitySet

If you need to work with many Entity Sets the same way or if you just need to pick up the used Entity Set name in run-time, you may find out the ability to get an instance of Entity Set proxy dynamically. Here is an example of how you can print a count of all employees:

```
count = getattr(northwind.entity_sets, 'Employees').get_entities().count().execute()
print(count)
```

3.9 URLs generation

Sometimes you may want to not use **PyOData** to actually make the HTTP requests, but just grab the url and body for some other library. For that, you can use following methods from ODataHttpRequest class - which is base of every query, update or delete covered in pyodata documentation.

```
.get_method()
.get_path()
.get_query_params()
.get_body()
```

3.9.1 Locust integration example

Warning - execute load testing scripts only against service you own!

Following is example of integration of pyodata as url provider for Locust load testing tool.

```
import requests
import pyodata
import os
from locust import HttpUser, task, between

SERVICE_URL = 'http://services.odata.org/V2/Northwind/Northwind.svc/'

odataClient = pyodata.Client(SERVICE_URL, requests.Session())
smith_employees_query = odataClient.entity_sets.Employees.get_entities().filter(
↪"FirstName eq 'John' and LastName eq 'Smith'")

class MyUser(HttpUser):
    wait_time = between(5, 15)
    host = SERVICE_URL

    @task(1)
    def filter_query(self):
        urlpath = smith_employees_query.get_path()
        url = os.path.join(SERVICE_URL,urlpath)
        params = smith_employees_query.get_query_params()
        self.client.get(url,params=params)
```