Data Object Service Documentation

Release 0.4.2

David Steinberg

Feb 06, 2019

Contents

1 Schemas for the Data Object Service (DOS) API **3**
- 1.1 Cloud Workstream . . . 3
- 1.2 What is DOS? . . . 3
- 1.3 Key features . . . 4
- 1.4 Implementations . . . 4
- 1.5 More information . . . 4

2 Quickstart **5**
- 2.1 Installing . . . 5
- 2.2 Running the client and server . . . 5
- 2.3 Further reading . . . 6

3 Data Object Service Demonstration Server **7**

4 DOS Python HTTP Client **9**

5 Tools for DOS Implementations **11**
- 5.1 Dynamic `/swagger.json` with Chalice . . . 11
- 5.2 Compliance testing . . . 11

6 Contributor’s Guide **15**
- 6.1 Installing . . . 15
- 6.2 Documentation . . . 15
- 6.3 Tests . . . 15
- 6.4 Schema architecture . . . 16
- 6.5 Releases . . . 16
- 6.6 Code contributions . . . 16

7 Indices and tables **17**

Welcome to the documentation for the Data Object Service Schemas! These schemas present an easy-to-implement interface for publishing and accessing data in heterogeneous storage environments. It also includes a demonstration client and server to make creating your own DOS implementation easy!

CHAPTER 1

Schemas for the Data Object Service (DOS) API

The Global Alliance for Genomics and Health is an international coalition formed to enable the sharing of genomic and clinical data. This collaborative consortium takes place primarily via GitHub and public meetings.

1.1 Cloud Workstream

The Data Working Group concentrates on data representation, storage, and analysis, including working with platform development partners and industry leaders to develop standards that will facilitate interoperability. The Cloud Workstream is an informal, multi-vendor working group focused on standards for exchanging Docker-based tools and CWL/WDL workflows, execution of Docker-based tools and workflows on clouds, and abstract access to cloud object stores.

1.2 What is DOS?

This proposal for a DOS release is based on the schema work of Brian W. and others from OHSU along with work by UCSC. It also is informed by existing object storage systems such as:

- GNOS (as used by PCAWG)
- ICGC Storage (as used to store data on S3, see overture-stack/score)
- Human Cell Atlas Storage (see HumanCellAtlas/data-store)
- NCI GDC Storage
- Keep by Curoverse (see curoverse/arvados)

The goal of DOS is to create a generic API on top of these and other projects, so workflow systems can access data in the same way regardless of project.

1.3 Key features

1.3.1 Data object management

This section of the API focuses on how to read and write data objects to cloud environments and how to join them together as data bundles. Data bundles are simply a flat collection of one or more files. This section of the API enables:

- create/update/delete a file
- create/update/delete a data bundle
- register UUIDs with these entities (an optionally track versions of each)
- generate signed URLs and/or cloud specific object storage paths and temporary credentials

1.3.2 Data object queries

A key feature of this API beyond creating/modifying/deletion files is the ability to find data objects across cloud environments and implementations of DOS. This section of the API allows users to query by data bundle or file UUIDs which returns information about where these data objects are available. This response will typically be used to find the same file or data bundle located across multiple cloud environments.

1.4 Implementations

There are currently a few experimental implementations that use some version of these schemas.

- DOS Connect observes cloud and local storage systems and broadcasts their changes to a service that presents DOS endpoints.
- DOS Downloader is a mechanism for downloading Data Objects from DOS URLs.
- dos-gdc-lambda presents data from the GDC public REST API using the Data Object Service.
- dos-signpost-lambda presents data from a signpost instance using the Data Object Service.

1.5 More information

- Global Alliance for Genomics and Health
- GA4GH Cloud Workstream

CHAPTER 2

Quickstart

2.1 Installing

Installing is quick and easy. First, it's always good practice to work in a virtualenv:

```
$ virtualenv venv
$ source venv/bin/activate
```

Then, install from PyPI:

```
$ pip install ga4gh-dos-schemas
```

Or, to install from source:

```
$ git clone https://github.com/ga4gh/data-object-service-schemas.git
$ cd data-object-service-schemas
$ python setup.py install
```

2.2 Running the client and server

There's a handy command line hook for the server:

```
$ ga4gh_dos_server
```

and for the client:

```
$ ga4gh_dos_demo
```

(The client doesn't do anything yet but will soon.)

2.3 Further reading

- gdc_notebook.ipynb outlines examples of how to access data with this tool.
- demo.py demonstrates basic CRUD functionality implemented by this package.

CHAPTER 3

Data Object Service Demonstration Server

CHAPTER 4

DOS Python HTTP Client

CHAPTER 5

Tools for DOS Implementations

The `ga4gh.dos` package contains some utilities that can help you develop a compliant DOS resolver.

5.1 Dynamic `/swagger.json` with Chalice

If you're using Chalice, you can expose a subset of the Data Object Service schema using `ga4gh.dos.schema.from_chalice_routes()`:

```
from chalice import Chalice
app = Chalice(...)

@app.route('/swagger.json')
def swagger():
    return ga4gh.dos.schema.from_chalice_routes(app.routes)
```

With the above code, a GET request to `/swagger.json` will return a schema in the Swagger / OpenAPI 2 format that correctly lists only the endpoints that are exposed by your app.

If you have a different `basePath`, you can also specify that:

```
@app.route('/swagger.json')
def swagger():
    return ga4gh.dos.schema.from_chalice_routes(app.routes, base_path='/api')
```

5.2 Compliance testing

This package contains a testing suite (`AbstractComplianceTest`) that streamlines testing implementations of the Data Object Service for compliance with the DOS schema.

This test suite is meant to supplement, and not replace, an existing test suite. It does not:

- test authentication

- test health of the service(s) underpinning an implementation
- test any endpoints not defined in the Data Object Service schema

class `ga4gh.dos.test.compliance.`**`AbstractComplianceTest`**(*methodName='runTest'*)

This class implements a number of compliance tests for Data Object Service implementations. It is meant to provide a single, standardized test harness to verify that a given DOS implementation acts in a manner consistent with the schema.

Using the test harness is pretty straightforward, and only requires implementing a method that can make requests to the service under test (*`_make_request()`*). As this class subclasses `unittest.TestCase`, all the functions exposed to a subclass of `unittest.TestCase` (e.g. `setUpClass()`) are available for use.

This test suite does not perform any authentication testing. Requests made during testing are made with the assumption that they will be properly authenticated in *`_make_request()`* or similar.

For a service built using Chalice, you would likely be able to write something similar to this:

```
from ga4gh.dos.test.compliance import AbstractComplianceTest
from chalice import LocalGateway, Config
from my_chalice_app import chalice_app

class TestApp(AbstractComplianceTest):
    @classmethod
    def setUpClass(cls):
        cls.lg = LocalGateway(chalice_app, Config())

    @classmethod
    def _make_request(self, meth, path, headers=None, body=None)
        headers = headers or {}
        r = self.lg.handle_request(method=meth, path='/ga4gh/dos/v1' + path,
                                   headers=headers, body=body)
        return r['body'], r['statusCode']
```

You would then be able to run the compliance test suite however you normally run your tests (e.g. `nosetests` or `python -m unittest discover`).

Variables supports – a list of supported DOS operations. By default, this is the list of all DOS operations, named by the *operationId* key in the schema:

```
supports = ['GetServiceInfo', 'GetDataBundleVersions',
            'CreateDataBundle', 'ListDataBundles',
            'UpdateDataObject', 'GetDataObject', ...]
```

Adding / removing operations from this list will adjust which tests are run. So, doing something like:

```
class Test(AbstractComplianceTest):
    self.supports = ['ListDataObjects']
```

would skip all tests calling UpdateDataBundle, GetDataBundle, and any other endpoint that is not ListDataObjects.

classmethod **`_make_request`**(*meth*, *path*, *headers=None*, *body=None*)

Method that makes requests to a DOS implementation under test given a method, path, request headers, and a request body.

The provided path is the path provided in the Data Object Service schema - this means that in your implementation of this method, you might need to prepend the provided path with your `basePath`, e.g. `/ga4gh/dos/v1`.

This method should return a tuple of the raw request content as a string and the return code of the request as an int.

Parameters

- **`meth`** (*`str`*) – the HTTP method to use in the request (i.e. GET, PUT, etc.)
- **`path`** (*`str`*) – path to make a request to, sans hostname (e.g. */databundles*)
- **`headers`** (*`dict`*) – headers to include with the request
- **`body`** (*`dict`*) – data to be included in the request body (serialized as JSON)

Return type tuple

Returns a tuple of the response body as a JSON-formatted string and the response code as an int

CHAPTER 6

Contributor's Guide

6.1 Installing

To install for development, install from source (and be sure to install the development requirements as well):

```
$ git clone https://github.com/ga4gh/data-object-service-schemas.git
$ cd data-object-service-schemas
$ python setup.py develop
$ pip install -r requirements.txt
```

6.2 Documentation

We use Sphinx for our documentation. You can generate an HTML build like so:

```
$ cd docs/
$ make html
```

You'll find the built documentation in `docs/build/`.

6.3 Tests

To run tests:

```
$ nosetests python/
```

The Travis test suite also tests for PEP8 compliance (checking for all errors except line length):

```
$ flake8 --select=E121,E123,E126,E226,E24,E704,W503,W504 --ignore=E501 python/
```

6.4 Schema architecture

The canonical, authoritative schema is located at `openapi/data_object_service.swagger.yaml`. All schema changes must be made to the Swagger schema, and all other specifications (e.g. SmartAPI, OpenAPI 3) are derived from it.

6.4.1 Building documents

To generate the OpenAPI 3 and SmartAPI descriptions, install swagger2openapi then run:

```
$ make schemas
```

6.5 Releases

New versions are released when `ga4gh.dos.__version__` is incremented, a commit is tagged (either through a release or manually), and the tagged branch builds successfully on Travis. When both conditions are met, Travis will automatically upload the distribution to PyPI.

If `ga4gh.dos.__version__` is not incremented in a new release, the build may appear to complete successfully, but the package will not be uploaded to PyPI as the distribution will be interpreted as a duplicate release and thus refused.

The process above is currently managed by david4096. To transfer this responsibility, ownership of the PyPI package must be transferred to a new account, and their details added to `.travis.yml` as described above.

Note that this repository will not become compliant with Semantic Versioning until version 1.0 - until then, the API should be considered unstable.

Documentation is updated independently of this release cycle.

6.6 Code contributions

We welcome code contributions! Feel free to fork the repository and submit a pull request. Please refer to this contribution guide for guidance as to how you should submit changes.

Data Object Service Schemas is licensed under the Apache 2.0 license. See LICENSE for more info.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

Symbols

_make_request() (ga4gh.dos.test.compliance.AbstractComplianceTest class method), 12

A

AbstractComplianceTest (class in ga4gh.dos.test.compliance), 12