
DataONE Python Products

unknown

2019-07-03

CONTENTS

1	Utilities (for end users)	3
1.1	DataONE Command Line Utilities and Examples	3
1.2	DataONE ONEDrive	3
1.3	DataONE Command Line Interface	3
2	Member Node (for Member Node partners)	5
2.1	Generic Member Node (GMN)	5
3	Python Libraries (for software developers)	7
3.1	DataONE Common Library for Python	7
3.2	DataONE Client Library for Python	7
3.3	DataONE Science Metadata Validator for Python	7
3.4	DataONE Test Utilities	7
3.5	DataONE Dev Tools	8
3.6	DataONE CSW Harvester	8
4	Contents	9
4.1	DataONE Utilities / Examples	9
4.2	DataONE ONEDrive	25
4.3	DataONE Command Line Interface (CLI)	57
4.4	Generic Member Node (GMN)	85
4.5	Indices and tables	161
4.6	DataONE Common Library for Python	162
4.7	Indices and tables	246
4.8	DataONE Client Library for Python	246
4.9	Indices and tables	284
4.10	DataONE Science Metadata library for Python	284
4.11	Indices and tables	292
4.12	DataONE Test Utilities	292
	Python Module Index	317
	Index	321

DataONE provides a number of products implemented in Python and Java, as part of the *Investigator Toolkit (ITK)*. Potential users of these products include software developers, Member Node partners and end users. Only the Python products are outlined in this document.

For software developers, DataONE provides development libraries implemented in Python. These provide functionality commonly needed by projects that interact with the DataONE infrastructure. It is recommended that applications implemented in Python use the libraries instead of interacting directly with the infrastructure as this is likely to reduce the development effort.

For Member Node partners, DataONE provides a Member Node (MN) implemented in Python, called Generic Member Node (GMN).

Lastly, DataONE provides various tools intended for end users, also implemented in Python. These include ONEDrive and the DataONE Command Line Client.

UTILITIES (FOR END USERS)

1.1 DataONE Command Line Utilities and Examples

The *DataONE Utilities / Examples* package contains command line utilities for interacting with the DataONE infrastructure.

The utilities are implemented using the DataONE *Common* and *Client* libraries for Python. Effort has been put into keeping the implementations clear and easy to read, allowing the utilities to also serve as examples on how to use the DataONE Python libraries.

After setup, all the utilities will be in the search path for the shell, so can be started from any directory. If using a virtual environment, the virtual environment must be active for the commands to work.

1.2 DataONE ONEDrive

DataONE ONEDrive enables DataONE objects stored in *Zotero citation manager* libraries to be accessed like regular files on Windows, Mac OS X and Linux systems. This allows users to open remote DataONE objects locally and work with them as if they reside on the user's computer. For instance, a spread sheet that is stored on a Member Node can be opened directly in Excel.

DataONE objects can be added to a Zotero library via the ONEMercury search tool. Objects can also be added in all the other ways that Zotero supports. ONEDrive connects to a Zotero library and makes all DataONE objects within the library accessible as regular files. Zotero collections are represented as folders in the ONEDrive filesystem.

1.3 DataONE Command Line Interface

The *DataONE Command Line Interface (CLI)* enables operations to be performed against the DataONE infrastructure from the command line. Supported operations include creating and retrieving DataONE objects, searching, updating access control rules and retrieving statistics.

MEMBER NODE (FOR MEMBER NODE PARTNERS)

2.1 Generic Member Node (GMN)

The *Generic Member Node (GMN)* is a DataONE Member Node *MN*. It provides an implementation of MN APIs and can be used by organizations to expose their science data to DataONE if they do not wish to create their own, native MN.

GMN can be used as a standalone MN or it can be used for exposing data that is already available on the web, to DataONE. When used in this way, GMN provides a DataONE compatible interface to existing data and does not store the data.

GMN can also be used as a workbone or reference for a 3rd party MN implementation. If an organization wishes to donate storage space to DataONE, GMN can be set up as a replication target.

PYTHON LIBRARIES (FOR SOFTWARE DEVELOPERS)

3.1 DataONE Common Library for Python

The *DataONE Common Library for Python* is a component of the DataONE Investigator Toolkit (ITK). It forms the foundation on which higher level components in the DataONE Python stack are built. It provides functionality commonly needed by clients, servers and other applications that interact with the *DataONE* infrastructure, including:

- Serializing, deserializing, validating and type conversions for the DataONE XML types
- Parsing and generating X.509 v3 certificates with DataONE extension
- Parsing and generating OAI-ORE Resource Maps as used by DataONE
- Utilities for working with XML documents, URLs, date-times, etc, in the context of DataONE

3.2 DataONE Client Library for Python

The *DataONE Client Library for Python* works together with the *DataONE Common Library for Python* to provide functionality commonly needed by client software that connects to DataONE nodes.

The main functionality provided by this library is a complete set of wrappers for all DataONE API methods. There are many details related to interacting with the DataONE API, such as creating MIME multipart messages, encoding parameters into URLs and handling Unicode. The wrappers hide these details, allowing the developer to communicate with nodes by calling native Python methods which take and return native Python objects.

The wrappers also convert any errors received from the nodes into native exceptions, enabling clients to use Python's concise exception handling system to handle errors.

3.3 DataONE Science Metadata Validator for Python

The *DataONE Science Metadata library for Python* is a component of the DataONE Investigator Toolkit (ITK). It currently provides schema validation of DataONE Science Metadata XML documents.

3.4 DataONE Test Utilities

The *DataONE Test Utilities* package contains various utilities for testing DataONE infrastructure components and clients. These include the *Instance Generator*, used for creating randomized System Metadata documents, and the *Stress Tester*, used for stress testing of Member Node implementations. The `stress_tester` can create many concurrent

connections to a Member Node and simultaneously create any number of randomly generated objects while running queries and object retrievals. There are also various *Utilities*.

3.5 DataONE Dev Tools

3.6 DataONE CSW Harvester

CONTENTS

4.1 DataONE Utilities / Examples

The *DataONE Utilities / Examples* package contains command line utilities for interacting with the DataONE infrastructure.

The utilities are implemented using the DataONE *Common* and *Client* libraries for Python. Effort has been put into keeping the implementations clear and easy to read, allowing the utilities to also serve as examples on how to use the DataONE Python libraries.

After setup, all the utilities will be in the search path for the shell, so can be started from any directory. If using a virtual environment, the virtual environment must be active for the commands to work.

Contents:

4.1.1 API

d1_util package

DataONE Utilities and Examples.

A collection of scripts intended to be useful both as command line utilities and as examples on how to interact with the DataONE infrastructure via the DataONE Python stack.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_util.tests package

Submodules

d1_util.tests.test_cert module

Submodules

d1_util.cert_check_cn module

Submit a PEM (Base64) encoded X.509 v3 certificate to a CN for validation.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Submit a PEM (Base64) encoded X.509 v3 certificate, optionally containing a DataONE SubjectInfo extension, to a CN to check if it passes validation and to determine which DataONE subjects are authenticated by it.

Example

```
$ cert-check-cn --cert-pub /tmp/x509up_u1000
```

Notes:

- Both the public and private key of the certificate are required. They may be in the same file, in which case only the `--cert-pub` option is required.
- See *check_x509_certificate_local.py* for how to process the lists of equivalent identities and group memberships in a DataONE SubjectInfo extension into a list of authenticated DataONE subjects.

```
d1_util.cert_check_cn.main()
```

d1_util.cert_check_local module

Parse a PEM (Base64) encoded X.509 v3 certificate.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Parse a PEM (Base64) encoded X.509 v3 certificate, optionally containing a DataONE SubjectInfo extension, to determine which DataONE subjects are authenticated by it.
- Process the lists of equivalent identities and group memberships in a DataONE SubjectInfo extension into a list of authenticated DataONE subjects.

Notes:

- This does not require the private key of the certificate and does not validate the certificate.

```
d1_util.cert_check_local.main()
```

d1_util.cert_create_ca module

Generate a self signed root Certificate Authority (CA) certificate.

The certificate can be used for issuing certificates and sign CSRs that are locally trusted.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Use the `d1_common.cert.x509` module to create a local self-signed CA certificate.

```
d1_util.cert_create_ca.main()
```

```
d1_util.cert_create_ca.create_ca(args)
```

```
exception d1_util.cert_create_ca.CACreateError
    Bases: Exception
```

d1_util.cert_create_csr module

Generate a Certificate Signing Request (CSR) for a Member Node Client Side Certificate, suitable for submitting to DataONE.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Use the `d1_common.cert.x509` module to create a local Certificate Signing Request

(CSR).

```
d1_util.cert_create_csr.main()
```

```
d1_util.cert_create_csr.create_csr(args)
```

exception `d1_util.cert_create_csr.CSRCreateError`

Bases: `Exception`

d1_util.cert_sign_csr module

Sign a Certificate Signing Request (CSR).

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Use the `d1_common.cert.x509` module to sign a Certificate Signing Request (CSR) using a

local CA.

```
d1_util.cert_sign_csr.main()
```

```
d1_util.cert_sign_csr.sign_csr(args)
```

```
d1_util.cert_sign_csr.assert_valid_path(p)
```

exception `d1_util.cert_sign_csr.CSRSignError`

Bases: `Exception`

d1_util.check_object_checksums module

Compare Science Object checksums for replicas on CNs and MNs.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Download Science Object checksums from CNs and MNs

```
d1_util.check_object_checksums.main()
```

```
d1_util.check_object_checksums.log_dict(d)
```

d1_util.check_scimeta_indexing module

d1_util.check_x509_certificate_cn module

Submit a PEM (Base64) encoded X.509 v3 certificate to a CN for validation.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Submit a PEM (Base64) encoded X.509 v3 certificate, optionally containing a DataONE SubjectInfo extension, to a CN to check if it passes validation and to determine which DataONE subjects are authenticated by it.

Notes:

- This requires the private key of the certificate, and the CN validates the certificate.
- See `check_x509_certificate_local.py` for how to process the lists of equivalent identities and group memberships in a DataONE SubjectInfo extension into a list of authenticated DataONE subjects.

```
d1_util.check_x509_certificate_cn.main()
```

d1_util.check_x509_certificate_local module

Parse a PEM (Base64) encoded X.509 v3 certificate.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Parse a PEM (Base64) encoded X.509 v3 certificate, optionally containing a DataONE SubjectInfo extension, to determine which DataONE subjects are authenticated by it.
- Process the lists of equivalent identities and group memberships in a DataONE SubjectInfo extension into a list of authenticated DataONE subjects.

Notes:

- This does not require the private key of the certificate and does not validate the certificate.

```
d1_util.check_x509_certificate_local.main()
```

d1_util.compare_object_lists module

d1_util.create_data_packages module

Create Data Package (Resource Map) on Member Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Create Data Packages (Resource Maps) from local files
- Upload local files and Data Packages to a Member Node

Operation:

Data packages are created from files in a folder provided by the user. Files with the same basename are combined into a package, with the basename being the name of the package.

Example:

The files

```
myfile.1.txt myfile.2.txt myfile.jpg
```

would be grouped into a package because they share the same basename. First, each of the files would be uploaded to the Member Node separately. The full filename is used as the PID.

For each file, a system metadata file is generated, based on information from the file and from a set of fixed settings.

Then, a package for all the files is generated. System metadata is generated for the package, and the package is uploaded to the Member Node.

```
d1_util.create_data_packages.main()
```

```
d1_util.create_data_packages.create_science_object_on_member_node(client,  
                                                                    file_path)
```



```

d1_util.create_data_packages.create_package_on_member_node(client, files_in_group)
d1_util.create_data_packages.create_resource_map_for_pids(package_pid, pids)
d1_util.create_data_packages.generate_system_metadata_for_science_object(pid,
                                                                    for-
                                                                    mat_id,
                                                                    sci-
                                                                    ence_object)

d1_util.create_data_packages.generate_sys_meta(pid, format_id, size, md5, now)
d1_util.create_data_packages.generate_public_access_policy()
d1_util.create_data_packages.find_file_groups(directory_path)
d1_util.create_data_packages.find_files_in_group(directory_path, group)
d1_util.create_data_packages.group_name(file_path)
d1_util.create_data_packages.base_name_without_extension(file_path)

```

d1_util.create_object_on_member_node module

Create Science Object on Member Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Upload a local file to a Member Node as a Science Object

Operation:

- The first time the script is run, a message indicating that the object was successfully created should be displayed, and the object should become available on the Member Node.
- If the script is then launched again without changing the identifier (PID), an IdentifierNotUnique exception should be returned. This indicates that the identifier is now in use by the previously created object.
- Any other errors will also be returned as DataONE exceptions.

```
d1_util.create_object_on_member_node.main()
```

d1_util.delete_all_objects_of_type module

Delete all Science Objects of specific type from Member Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Retrieve a list of all objects with specific FormatID on a Member Node
- Delete all objects with a specific FormatID from a Member Node

Notes:

- Do NOT use this script to delete undesired objects from a production Member Node!
- The objects are deleted with the MNStorage.delete() API method. The API method is intended to be called only by CNs under specific circumstances. In a stand-alone or non-production environment, the API can be used for removing objects from a Member Node.
- MNStorage.delete() is only available to subjects which have delete permission on the node.
- To delete all the objects on the node, remove the formatId and replicaStatus parameters in the listObjects() call below.

- The Member Node object list is retrieved in small sections, called pages. Because removing objects may, depending on the implementation of `listObjects()`, cause the contents in each page to shift, the entire list of objects to delete is created first and then the deletions are performed in a separate step. This could require a lot of memory if running on a server with a large number of objects. In that case, an alternative implementation is to delete the objects as they are discovered and repeat the process until no more objects to delete are found.
- The `listObjects()` Member Node API method may not be efficiently implemented by all Member Nodes as it is intended primarily for use by Coordinating Nodes.

Operation:

- Configure the script in the Config section below

```
d1_util.delete_all_objects_of_type.main()

class d1_util.delete_all_objects_of_type.MemberNodeObjectDeleter(base_url)
    Bases: object

    delete_objects_from_member_node()
```

d1_util.display_node_status module

Display node status.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Retrieve a list of all DataONE Nodes
- Get and display key metrics for each of the Nodes.

Notes:

- See the description for the CERTIFICATE setting below for limitations in the information displayed by this script.

Operation:

- Configure the script in the Config section below

```
d1_util.display_node_status.main()

d1_util.display_node_status.get_node_list_from_coordinating_node()

d1_util.display_node_status.get_cn_metrics(cn)

d1_util.display_node_status.get_mn_metrics(mn)

d1_util.display_node_status.print_capabilities(client)

d1_util.display_node_status.get_gen_metrics(client, node)

d1_util.display_node_status.get_ping(client)

d1_util.display_node_status.get_number_of_objects(client)

d1_util.display_node_status.get_number_of_log_records(client)

d1_util.display_node_status.is_member_node(node)

d1_util.display_node_status.is_coordinating_node(node)
```

d1_util.download_all_objects module

Download all Science Objects in a DataONE environment.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Retrieve a list of all DataONE Member Nodes
- Retrieve a list of all objects of specific FormatID on each of those Member Nodes
- Retrieve and examine the System Metadata for each of the listed objects
- Based on information in the System Metadata, determine if the corresponding object should be downloaded
- Download the corresponding object

Notes:

- This approach retrieves object lists directly from each Member Node and is mainly suitable in special situations where a 3rd party wishes to examine the overall state of objects in DataONE, for instance, for creating statistics or data quality reports.
- This approach uses the `listObjects()` Member Node API method, which has limited filtering facilities. The example shows how to use this filtering to list objects that are of a specific type (FormatID) and that are native to the Member Node (i.e., not replicas). If a more specific set of objects is desired, it is better to use DataONE's query interface, which offers much richer filtering facilities.
- It is not possible to filter out non-public objects with `listObjects()`. Instead, this script attempts to download the object's System Metadata and checks for `NotAuthorized` exceptions.
- If a completely unfiltered object list is required, simply remove the `formatId` and `replicaStatus` parameters in the `listObjects()` call below.
- The Member Node object list is retrieved in small sections, called pages. The objects on each page are processed before retrieving the next page.
- The `listObjects()` Member Node API method may not be efficiently implemented by all Member Nodes as it is intended primarily for use by Coordinating Nodes.
- The `listObjects()` method may miss objects that are created while the method is in use.

```
d1_util.download_all_objects.main()
d1_util.download_all_objects.get_node_list_from_coordinating_node()
d1_util.download_all_objects.is_member_node(node)
class d1_util.download_all_objects.MemberNodeObjectDownloader(node)
    Bases: object
        download_objects_from_member_node()
        download_d1_object(pid)
```

d1_util.download_and_display_data_package module

Download and display Data Package (Resource Map) from Member Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Download a Data Package (Resource Map)
- Parse and display the Resource Map.

Operation:

- Configure the script in the Config section below

```
d1_util.download_and_display_data_package.main()
```

d1_util.download_mn_objects module

Download Science Objects from a Member Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Retrieve a list of all objects on a MN
- Retrieve the bytes and System Metadata for each object

Operation:

- Configure the script in the Config section below

```
d1_util.download_mn_objects.main()
```

```
class d1_util.download_mn_objects.MemberNodeObjectDownloader(base_url, download_folder, object_id_filter_list=None, max_object_size=None)
```

Bases: object

download_all()

```
exception d1_util.download_mn_objects.DownloadError
```

Bases: Exception

d1_util.download_sciobj module

Download Science Objects from a Member Node or Coordinating Node.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Download a Science Object from a MN or CN.

```
d1_util.download_sciobj.main()
```

d1_util.download_server_certs module

Download the issuer CA X.509 certificates for all DataONE nodes.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Downloads server side certificate from a DataONE nodes
- Parse certificates to find the issuer CA certificate URLs
- Downloads the CA certs

Operation:

This process downloads the server side certificates from the DataONE nodes and parses them to find the issuer CA certificate URLs. It then downloads the CA certs.

The CA certs can then be installed as trusted CAs in the local environment in order to ensure that the DataONE client library trusts all server side certs currently in use in DataONE.

To install the CA bundles on Ubuntu and derived distributions, move the files to:

/usr/local/share/ca-certificates

Then run:

update-ca-certificates

update-ca-certificates only processes “.crt” files, so this script saves the certificates with that extension.

```
d1_util.download_server_certs.main()
d1_util.download_server_certs.download_server_cert(base_url, node_id, download_dir_path)
```

d1_util.download_sysmeta module

Download and display Science Metadata.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Download Science Metadata from a MN or CN.
- Format the Science Metadata XML document for display or save to disk.

```
d1_util.download_sysmeta.main()
```

d1_util.download_sysmeta_multiproc module

Bulk download System Metadata object from MN.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Use the multiprocessed System Metadata iterator to efficiently perform bulk downloads of System Metadata from a Member Node

```
d1_util.download_sysmeta_multiproc.main()
d1_util.download_sysmeta_multiproc.parse_cmd_line()
exception d1_util.download_sysmeta_multiproc.SysMetaRetrieveError
    Bases: Exception
```

d1_util.download_test_docs module

Download randomly selected Science Metadata objects from CN.

This is an example on how to use the DataONE Science Metadata library for Python. It shows how to:

- Query the DataONE Solr index for a random selection of object identifiers for a given formatId.
- Download objects with high bandwidth throughput by using the async DataONEClient to perform concurrent downloads.

```
async d1_util.download_test_docs.main()
async d1_util.download_test_docs.validate_bulk(client, out_dir_path, format_id,
                                              pid_count, solr_client, progress_logger,
                                              task_name)
async d1_util.download_test_docs.download(client, out_dir_path, pid, format_id,
                                          progress_logger)
async d1_util.download_test_docs.save_xml(out_dir_path, pid, sciobj_f)
```

async `d1_util.download_test_docs.get_random_pid_list (solr_client, format_id,
pid_count)`
Query Solr for a list of randomly selected PIDs of objects with a given formatId.

d1_util.find_gmn_instances module

d1_util.generate_data_package_from_stream module

d1_util.generate_object_stats module

Generate statistics for Science Objects on a given set of Member Nodes.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Aggregate values from System Metadata on a set of Member Nodes

Operation:

- Configure the script in the Config section below

```
d1_util.generate_object_stats.main()  
d1_util.generate_object_stats.find_object_size_stats_node_all (gmn_node_list)  
d1_util.generate_object_stats.find_object_size_stats_node (gmn_dict)  
d1_util.generate_object_stats.max_size_sysmeta_list (sysmeta_pyxb_list, sys-  
meta_pyxb, max_size=10)  
d1_util.generate_object_stats.log_dict (d)
```

d1_util.jwt_token_tasks module

Perform various operations on Java Web Tokens (JWTs)

This is an example on how to use the DataONE Client and Common libraries for Python.

```
d1_util.jwt_token_tasks.main()  
d1_util.jwt_token_tasks.validate_and_decode (jwt_bu64, cert_obj)  
Example for validating the signature of a JWT using only the cryptography library.
```

Note that this does NOT validate the claims in the claim set.

```
d1_util.jwt_token_tasks.find_valid_combinations (cert_file_name_list,  
jwt_file_name_list)  
Given a list of cert and JWT file names, print a list showing each combination along with indicators for combinations where the JWT signature was successfully validated with the cert.  
d1_util.jwt_token_tasks.download_cn_certs ()  
d1_util.jwt_token_tasks.jwt_cleanup ()  
d1_util.jwt_token_tasks.cert_cleanup ()  
d1_util.jwt_token_tasks.filename_from_cert_obj (cert_obj)
```

d1_util.parse_format_id_list module

Parse ObjectFormatList XML doc with XPath.

This is an example on how to use the DataONE Client and Common libraries for Python. It shows how to:

- Extract formatIds from a DataONE ObjectFormatList using XPath

```
d1_util.parse_format_id_list.main()
```

```
d1_util.parse_format_id_list.get_scimeta_format_id_list(xsd_path)
```

d1_util.resolve_package_identifier module

Resolve an OAI-ORE Resource Map (data package) identifier to download URL for a BagIt ZIP archive of the package.

This is an example on how to use the DataONE Client and Common libraries for Python.

```
d1_util.resolve_package_identifier.main()
```

d1_util.solr_query module

Solr query.

This is an example on how to use the DataONE Client Library for Python. It shows how to:

- Query DataONE's Solr index
- Display the results

```
d1_util.solr_query.main()
```

d1_util.validate_system_metadata module

d1_util.xml_apply_xslt module

Apply XSLT transform to XML document.

This is an example on how to use the DataONE Science Metadata library for Python. It shows how to:

- Deserialize, process and serialize XML docs.
- Apply an XSLT transform.
- Display or save the resulting XML doc.

```
d1_util.xml_apply_xslt.main()
```

```
exception d1_util.xml_apply_xslt.ResolveError
    Bases: Exception
```

d1_util.xml_remove_empty_elements module

Remove empty elements from XML that might interfere with XSD schema validation.

Overall formatting is maintained.

This is an example on how to use the DataONE Science Metadata library for Python. It shows how to:

- Deserialize, process and serialize XML docs.
- Apply an XSLT transform which removes empty elements from XML.
- Display or save the resulting XML doc.

```
d1_util.xml_remove_empty_elements.main()
```

```
exception d1_util.xml_remove_empty_elements.ResolveError
    Bases: Exception
```

d1_util.xml_strip_whitespace module

Strip whitespace that might interfere with XSD schema validation.

Overall formatting is maintained. Note that pretty printing the doc is likely to add the stripped whitespace back in.

This is an example on how to use the DataONE Science Metadata library for Python. It shows how to:

- Deserialize, process and serialize XML docs.
- Apply an XSLT transform which strips potentially problematic whitespace.
- Display or save the resulting XML doc.

```
d1_util.xml_strip_whitespace.main()
```

```
exception d1_util.xml_strip_whitespace.ResolveError
    Bases: Exception
```

4.1.2 Glossary

DataONE Terms

DataONE Data Observation Network for Earth

<https://dataone.org>

DataONE Common Library for Python Part of the DataONE *Investigator Toolkit (ITK)*. Provides functionality commonly needed by projects that interact with the *DataONE* infrastructure via Python. It is a dependency of *DataONE Client Library for Python*, *GMN* and currently all other DataONE components written in Python.

DataONE Client Library for Python Part of the DataONE *Investigator Toolkit (ITK)*. Provides programmatic access to the DataONE infrastructure and may be used to form the basis of larger applications or to extend existing applications to utilize the services of DataONE.

DataONE Test Utilities for Python A framework for testing and validation of DataONE components implemented in Python.

GMN DataONE Generic Member Node.

A DataONE Member Node (*MN*). It provides an implementation of MN APIs and can be used by organizations to expose their science data to DataONE if they do not wish to create their own, native MN.

Metacat Metacat is a flexible, open source metadata catalog and data repository that targets scientific data, particularly from ecology and environmental science. Metacat accepts XML as a common syntax for representing the large number of metadata content standards that are relevant to ecology and other sciences. Thus, Metacat is a generic XML database that allows storage, query, and retrieval of arbitrary XML documents without prior knowledge of the XML schema.

Metacat provides a complete implementation of all *MN* APIs.

<http://www.dataone.org/software-tools/metacat>

Replication target A *MN* that accepts replicas (copies) of science data from other MNs and thereby helps ensuring that science data remains available.

Vendor specific extensions Functionality that is not part of the DataONE APIs but is supported by a DataONE component. Vendor specific extensions are activated by adding custom HTTP headers when calling the existing DataONE API methods. When activated, they modify the behavior of the method in a vendor specific way. DataONE has reserved the namespace starting with *VENDOR_* for such custom headers.

Investigator Toolkit (ITK) The Investigator Toolkit provides a suite of software tools that are useful for the various audiences that DataONE serves. The tools fall in a number of categories, which are further developed here, with examples of potential applications that would fit into each category. <https://releases.dataone.org/online/api-documentation-v2.0.1/design/itk-overview.html>

MN DataONE Member Node.

CN DataONE Coordinating Node.

Node DataONE Member Node or Coordinating Node

client An application that accesses the DataONE infrastructure on behalf of a user.

Science Data An object (file) that contains scientific observational data.

Science Metadata An object (file) that contains information about a Science Data object.

System Metadata An object (file) that contains system level information about a Science Data or a Science Metadata object.

PID Persistent Identifier. An identifier that is unique within DataONE and references an immutable object.

SID Series Identifier. An identifier that is unique within DataONE and references one or more objects that have been linked together by a series of updates.

Workspace The Workspace is an online storage area where users can store search filters and references to DataONE objects. It follows the files and folders metaphor of regular filesystems. Objects are added to the Workspace from the ONEMercury search engine.

Authentication and security

X.509 An ITU-T standard for a public key infrastructure (PKI) for single sign-on (SSO) and Privilege Management Infrastructure (PMI). X.509 specifies, amongst other things, standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm.

<http://en.wikipedia.org/wiki/X509>

CA Certificate Authority

A certificate authority is an entity that issues digital *certificate* s. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the public key that is certified. In this model of trust relationships, a CA is a trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes.

http://en.wikipedia.org/wiki/Certificate_authority

CA signing key The private key which the *CA* uses for signing *CSRs*.

Server key The private key that Apache will use for proving that it is the owner of the *certificate* that it provides to the client during the SSL handshake.

CSR Certificate Signing Request

A message sent from an applicant to a [CA](#) in order to apply for a [certificate](#).

http://en.wikipedia.org/wiki/Certificate_signing_request

Certificate A public key certificate (also known as a digital certificate or identity certificate) is an electronic document which uses a digital signature to bind a public key with an identity – information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

http://en.wikipedia.org/wiki/Public_key_certificate

CA certificate A certificate that belongs to a [CA](#) and serves as the root certificate in a term: *chain of trust*.

Self signed certificate A [certificate](#) that is signed by its own creator. A self signed certificate is not a part of a *chain of trust* and so, it is not possible to validate the information stored in the certificate. Because of this, self signed certificates are useful mostly for testing in an implicitly trusted environment.

http://en.wikipedia.org/wiki/Self-signed_certificate

Chain of trust The Chain of Trust of a Certificate Chain is an ordered list of certificates, containing an end-user subscriber certificate and intermediate certificates (that represents the Intermediate CA), that enables the receiver to verify that the sender and all intermediates certificates are trustworthy.

http://en.wikipedia.org/wiki/Chain_of_trust

DN Distinguished Name.

OpenSSL Toolkit implementing the [SSL](#) v2/v3 and [TLS](#) v1 protocols as well as a full-strength general purpose cryptography library.

SSL Secure Sockets Layer

A protocol for transmitting private information via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data – a public key known to everyone and a private or secret key known only to the recipient of the message.

SSL handshake The initial negotiation between two machines that communicate over SSL.

<http://developer.connectopensource.org/display/CONNECTWIKI/SSL+Handshake>

http://developer.connectopensource.org/download/attachments/34210577/Ssl_handshake_with_two_way_authentication_with_certificates.png

TLS Transport Layer Security

Successor of [SSL](#).

Client side authentication [SSL](#) Client side authentication is part of the [SSL handshake](#), where the client proves its identity to the web server by providing a [certificate](#) to the server. The certificate provided by the client must be signed by a [CA](#) that is trusted by the server. Client Side Authentication is not a required part of the handshake. The server can be set up to not allow Client side authentication, to require it or to let it be optional.

Server Side Authentication [SSL](#) Server Side Authentication is part of the [SSL handshake](#), where the server proves its identity to the client by providing a [certificate](#) to the client. The certificate provided by the server must be signed by a [CA](#) that is trusted by the client. Server Side Authentication is a required part of the handshake.

Client side certificate [Certificate](#) that is provided by the client during *client side authentication*.

Server side certificate [Certificate](#) that is provided by the server during *server side authentication*.

Identity Provider A service that creates, maintains, and manages identity information for principals while providing authentication services to relying party applications within a federation or distributed network.

4.1.3 ONEDrive

FUSE Filesystem in Userspace.

<http://fuse.sourceforge.net/>

macfuse <http://code.google.com/p/macfuse/>

fusepy <http://code.google.com/p/fusepy/>

Dokan User mode file system for windows.

<http://dokan-dev.net/en/>

Misc

Subversion Version control system

<http://subversion.apache.org/>

Bash GNU Bourne-Again Shell

<http://www.gnu.org/software/bash/>

Apache HTTP server

<http://httpd.apache.org/>

MPM Multi-Processing Module

The component within Apache that manages the processes and threads used for serving requests.

<http://httpd.apache.org/docs/2.0/mpm.html>

Python A dynamic programming language.

<http://www.python.org>

Django High-level Python Web framework that encourages rapid development and clean, pragmatic design.

<https://www.djangoproject.com/>

WSGI Web Server Gateway Interface

<http://www.wsgi.org/wsgi/>

mod_wsgi An *Apache* module that implements *WSGI*.

mod_ssl An *Apache* module that interfaces to *OpenSSL*.

PyXB Python XML Schema Bindings

<http://pyxb.sourceforge.net/>

minixsv A Lightweight XML schema validator

<http://www.familieleuthe.de/MiniXsv.html>

python-dateutil Extends the standard datetime module

<http://labix.org/python-dateutil>

PostgreSQL A freely available object-relational database management system (ORDBMS).

<http://www.postgresql.org/>

MySQL A freely available object-relational database management system (ORDBMS).

<http://www.mysql.com/>

SQLite3 A freely available object-relational database management system (ORDBMS).

<http://www.sqlite.org/>

Oracle A object-relational database management system (ORDBMS) that is available in both free and commercial versions.

<http://www.oracle.com/>

Psycopg2 Psycopg is a PostgreSQL database adapter for *Python*.

<http://initd.org/psycopg/>

OpenSSL An open source implementation of the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.

<http://www.openssl.org/>

cron cron is a time-based job scheduler in Unix-like computer operating systems. cron enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates.

python-setuptools A package manager for Python

<http://pypi.python.org/pypi/setuptools>

ISO8601 International standard covering the exchange of date and time-related data

http://en.wikipedia.org/wiki/ISO_8601

python-iso8601 Python library implementing basic support for *ISO8601*

<http://pypi.python.org/pypi/iso8601/>

CILogon The CILogon project facilitates secure access to CyberInfrastructure (CI).

<http://www.cilogon.org/>

LOA Levels of Assurance

CILogon operates three Certification Authorities (CAs) with consistent operational and technical security controls. The CAs differ only in their procedures for subscriber authentication, identity validation, and naming. These differing procedures result in different Levels of Assurance (LOA) regarding the strength of the identity contained in the certificate. For this reason, relying parties may decide to accept certificates from only a subset of the CILogon CAs.

<http://ca.cilogon.org/loa>

REST Representational State Transfer

A style of software architecture for distributed hypermedia systems such as the World Wide Web.

http://en.wikipedia.org/wiki/Representational_State_Transfer

Solr Apache Solr

Solr is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites.

<http://lucene.apache.org/solr/>

OAI-ORE Resource Map Open Archives Initiative Object Reuse and Exchange (OAI-ORE) defines standards for the description and exchange of aggregations of Web resources.

<http://www.openarchives.org/ore/1.0/>

4.1.4 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.2 DataONE ONEDrive

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

DataONE ONEDrive enables DataONE objects stored in [Zotero citation manager](#) libraries to be accessed like regular files on Windows, Mac OS X and Linux systems. This allows users to open remote DataONE objects locally and work with them as if they reside on the user's computer. For instance, a spread sheet that is stored on a Member Node can be opened directly in Excel.

DataONE objects can be added to a Zotero library via the ONEMercury search tool. Objects can also be added in all the other ways that Zotero supports. ONEDrive connects to a Zotero library and makes all DataONE objects within the library accessible as regular files. Zotero collections are represented as folders in the ONEDrive filesystem.

Contents:

4.2.1 Installation

Microsoft Windows

1. Download the [latest ONEDrive for Windows setup](#).
1. Start the setup and follow the prompts.
2. Start ONEDrive from the Windows Start menu.
3. See [Using ONEDrive](#) for notes on how to customize and access ONEDrive.

By default, ONEDrive uses the drive letter "O:". If this drive letter is already in use, it can be changed in the `settings.py` file.

Mac OS X

1. Install FUSE

Development of ONEDrive on OS X has been done using the [Fuse for OS X](#) distribution. Download the latest installer (currently 2.5.4) and follow the instructions to install.

1. Install Python dependencies

`fusepy` provides the Python bindings to the FUSE library. To install `fusepy`, use the commands:

```
$ cd Downloads
$ git clone git://github.com/terencehonles/fusepy.git fusepy
$ cd fusepy
$ sudo python setup.py install
```

1. Install ONEDrive

There is currently no setup script for ONEDrive, so installation means simply downloading to a local folder:

```
$ cd ~/opt
$ svn co https://repository.dataone.org/software/cicore/trunk/itk/d1_client_onedrive
$ cd d1_client_onedrive
```

- Set PYTHONPATH to include d1_common_python/src and d1_libclient_python/src
- On OS X, set DYLD_LIBRARY_PATH=/usr/lib:\$DYLD_LIBRARY_PATH
- Make sure option ‘user_allow_other’ is set in /etc/fuse.conf.

If the library search path is incomplete, an exception such as the following may occur:

```
OSError: dlopen(/opt/local/lib/libfuse.dylib, 6): Symbol not found: _iconv
  Referenced from: /opt/local/lib/libfuse.dylib
```

To work around this, run onedrive.py with:

```
export DYLD_LIBRARY_PATH=/usr/lib:$DYLD_LIBRARY_PATH
```

Linux

Make sure the system is up to date:

```
sudo -H bash -c '
  apt update --yes && apt dist-upgrade --yes
'
```

- Reboot if necessary.

Set up server packages:

- The build environment for DataONE Python extensions
- Commands used in the install

```
$ sudo apt install --yes build-essential python-dev libxml2-dev \
libxslt-dev
```

Install pip:

```
$ sudo apt install --yes python-pip; sudo pip install pip --upgrade;
```

Install ONEDrive, and its dependencies from PyPI, into a Python virtual environment. The virtual environment is set up under onedrive_bin in the user’s home folder.

```
$ sudo pip install virtualenv;
$ cd; mkdir onedrive_bin; virtualenv --distribute onedrive_bin;
cd onedrive_bin; . bin/activate; pip install dataone.onedrive
```

- Press ctrl-d to exit the virtualenv.

ONEDrive expects to find a workspace.xml file in your home folder. Copy one of the example workspaces there:

```
$ cp onedrive_bin/workspace.xml ~
```

By default, ONEDrive uses a folder named “one” in your home folder as the mount point. Create it:

```
$ mkdir ~/one
```

Start ONEDrive:

```
$ ~/onedrive_bin/bin/onedrive
```

Open ~/one to access your DataONE objects.

4.2.2 Using ONEDrive

The default settings for ONEDrive are in the `settings.py` file that resides in the same location as the `onedrive.py` script. To modify the default settings, edit `settings.py`.

To launch ONEDrive with the default settings, simply start ONEDrive. See the OS specific sections below for how to start ONEDrive on your computer.

Most of the defaults can be overridden at launch time by adding options on the command line. The options are listed below, together with their default values (from `settings.py`):

```
Usage: onedrive.py [options]
```

Options:

```
-h, --help            show this help message and exit
-v, --version          Display version information and exit
--disable-fuse-foreground
--directory-max-cache-items=10000
--macfuse-icon=/home/user/.dataone/dl.icon
--sci-obj-cache-path=/home/user/.dataone/onedrive/sci_obj
--attribute-max-cache-items=10000
--disable-fuse-nothreads
--resource-map-size=size
--max-solr-query-cache-size=1000
--region-tree-max-cache-items=1000
--disable-macfuse-local-disk
--disable-fuse-nonempty
--log-level=DEBUG
--object-tree-cache-path=/home/user/.dataone/onedrive/object_tree
--sys-meta-cache-path=/home/user/.dataone/onedrive/sys_meta
--region-tree-cache-path=/home/user/.dataone/onedrive/region_tree
--base-url=https://cn.dataone.org/cn
--max-objects-for-query=50
--sci-obj-max-cache-items=10000
--zotero-cache-path=/home/user/.dataone/onedrive/zotero_library
--folder-size-for-resource-maps=zero
--fuse-filesystem-name=ONEDrive
--disable-debug
--mountpoint=/home/user/one
--sys-meta-max-cache-items=10000
--disable-solr-debug
--log-file-path=/home/user/.dataone/onedrive/onedrive.log
--mount-drive-letter=O:
--onedrive-cache-root=/home/user/.dataone/onedrive
--solr-query-path=/v1/query/solr/
```

Zotero Library Integration

ONEDrive uses the [Zotero citation manager](#) as an online repository of references to DataONE objects. As DataONE objects are added to a Zotero library via the ONEMercury search tool or any other method supported by Zotero, they become available as files in the ONEDrive filesystem. The files can then be opened directly in applications running on your own computer.

ONEDrive shows up in your computer as an extra storage device, much like a CD drive or a USB flash drive. Like your regular storage devices, ONEDrive contains folders that can contain files or other folders. The folders represent collections in Zotero. To make DataONE objects appear in a given folder in ONEDrive, add them to the corresponding collection in Zotero.

Folders can contain objects that have been specified directly and search queries that can specify any number of objects. Search queries are dynamically resolved to their matching DataONE objects and those objects become available within the ONEDrive filesystem, in the same folder in which the search query is stored.

ONEDrive recognizes DataONE objects in the Zotero library by their URLs. Zotero library items that have URLs that reference the DataONE Coordinating Node resolve endpoint at `https://cn.dataone.org/cn/v1/resolve/<identifier>` appear directly as DataONE objects. Library items that have URLs that reference the query endpoint at `https://cn.dataone.org/cn/v1/query/solr/<query>` will cause the queries to be executed on the Coordinating Node and the resulting DataONE objects will appear in the ONEDrive filesystem.

Notes

ONEDrive checks for updates in the Zotero library each time it is started. If the library has been updated, ONEDrive will refresh its local cache of the Zotero library and the metadata for the DataONE objects exposed through the filesystem.

Zotero can have multiple root level collections while a filesystem can have only one root. ONEDrive handles this by adding an additional level, so that root level collections in Zotero are the first level directories in the filesystem root.

Items in the Zotero library don't have to be in a collection. Any objects not in a collection are displayed in the root of the filesystem.

The folders in the ONEDrive filesystem contain readme files that describe the contents of the folders.

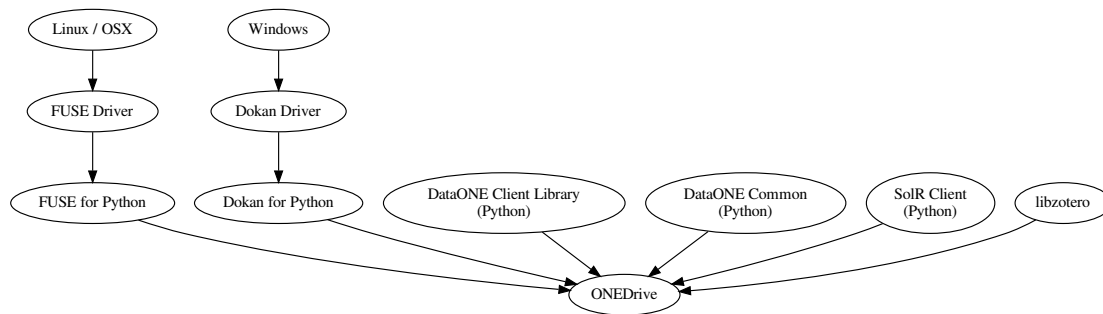
Because the DataONE API currently does not specify a way for Member Nodes to allow partial downloads of objects, ONEDrive downloads the entire object the first time it is accessed through the filesystem. If the object is large, the filesystem will appear to freeze while this download is being performed in the background. When the entire object has been downloaded to ONEDrive's cache, the filesystem becomes responsive again. ONEDrive caches objects across runs, so this will only happen the first time an object is accessed.

FlatSpace

In the root of the ONEDrive filesystem, there are two folders, FlatSpace and ObjectTree. ObjectTree exposes the Zotero based functionality described above. FlatSpace exposes functionality that allows DataONE objects to be accessed without first having to add them to the Zotero library. To access objects directly through FlatSpace, simply type the object identifier at the end of the filesystem path after entering the FlatSpace folder.

After an object has been accessed through FlatSpace, ONEDrive will start rendering a folder for the object in FlatSpace so that the identifier does not have to be typed the next time the object is accessed. ONEDrive caches this information across runs.

4.2.3 Architecture



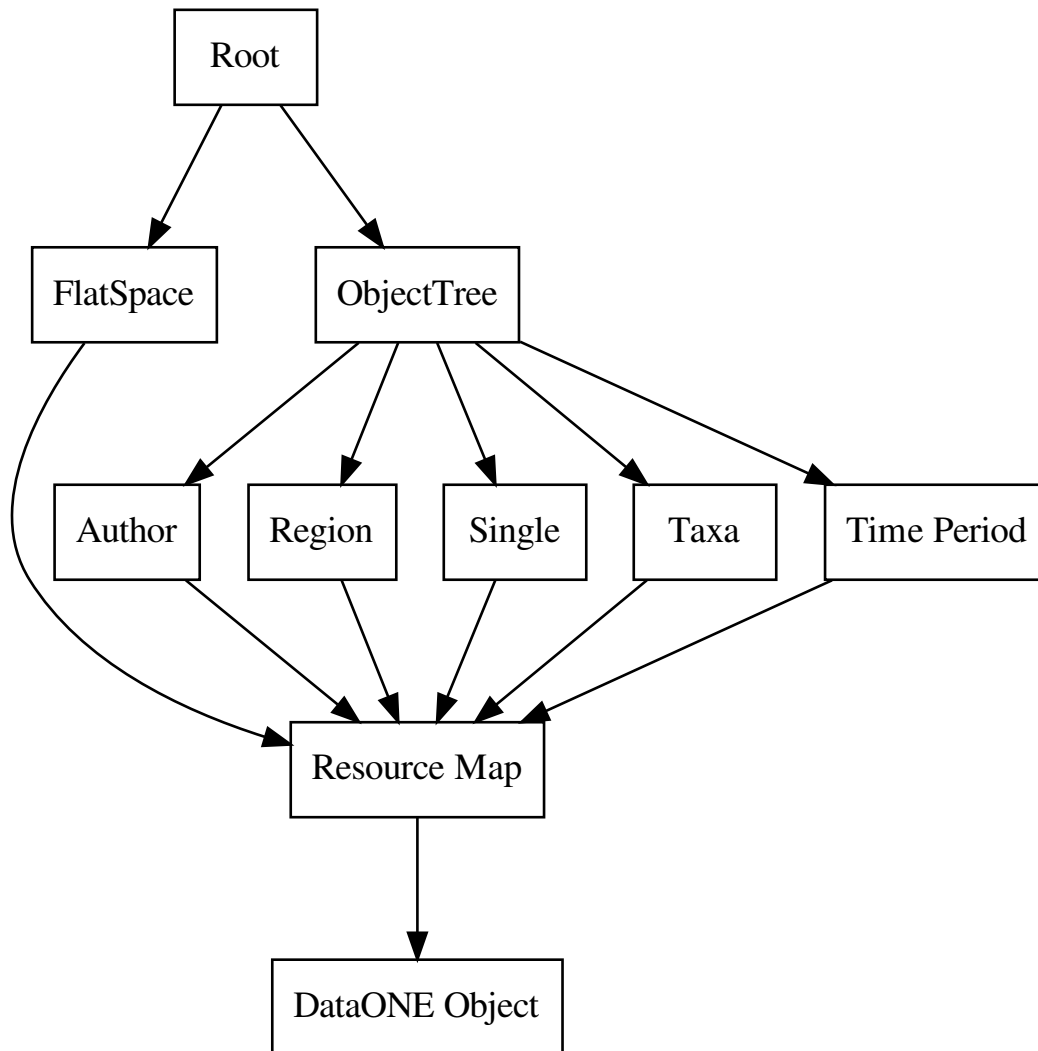
Resolvers

The resolvers are classes that “resolve” filesystem paths to lists of files and folders for those paths. The resolvers are arranged into a hierarchy. Each resolver examines the path and may resolve the path itself or pass control to another resolver.

Resolvers deeper in the hierarchy corresponds to sections that are further to the right in the path. If a resolver passes control to another resolver, it first removes the section of the left side of the path that it processed. Thus, each resolver needs to know only how to parse the section of the path that it is designed to handle. This also enables the same functionality to be exposed several places in the filesystem. For instance, the resolver for the object package level can be reached though each of the root level search types.

If a resolver determines that the path that it has received is invalid, it can abort processing of the path by raising a `PathException`.

The hierarchy of resolvers



- The resolvers are all derived from the `Resolver` class, not from each other.
- Each resolver has three public methods, `get_attributes()`, `get_directory()` and `read_file()`. `get_attributes()` returns the attributes for a file or folder. `get_directory()` returns the directory contents for a folder. `read_file()` returns sections of a DataONE object.
- The `Root` resolver renders the root directory, which contains a set of directories designating different types of interactions which can be performed with the DataONE infrastructure. It also parses the root elements of paths and transfers control to the appropriate path resolver.
- All the resolvers handle paths as lists of path segments. The root resolver performs the conversion of the path string to a list of path segments by splitting the path on the path separator and unescaping the segments. This allows the path segments to contain DataONE identifiers that include the path separator and simplifies path

handling in the resolvers.

- `ObjectTree` `ObjectTree` renders a filesystem folder structure that corresponds with the hierarchy of collections in the Zotero library. It takes a source tree generator as input and that generator is currently the Zotero client. This abstraction makes it easy to support additional online libraries, sky drives and reference managers in the future.
- `FlatSpace` enables direct access to objects and enables users to share short ONEDrive paths to directly access specific objects.
- `Resource Map` renders the contents of a OAI-ORE Resource Map.
- `DataONE Object` renders the folder view of a single DataONE object.

The Root resolver

As an example of the pattern that the resolvers follow, consider the Root resolver. The Root resolver is responsible for rendering the root directory, `/`, and for dispatching paths out to the other resolvers. Only the root folder is handled by the Root resolver.

`get_attributes("/"): Return the attributes for / (0 size, directory).`

`get_attributes("/ObjectTree"):` Not handled by the Root resolver. The Root resolver strips off `/ObjectTree`, and passes the remaining path, `/` to the `ObjectTree` resolver. So, even though `/ObjectTree` is returned by `get_directory("/")` (see below) of the Root resolver, that same path is not handled by the Root resolver.

`get_attributes("/ObjectTree/some/other/path"):` Same as `get_attributes("/ObjectTree")`, except that the path passed to the `ObjectTree` resolver is now `/some/other/path`.

`get_attributes("/invalid"):` This invalid path is handled by the Root resolver, which raises an `InvalidPath` exception.

`get_directory("/"): Return directories for all of the valid 1st level resolvers, such as ObjectTree.`

`get_directory("/ObjectTree"):` Not handled by the Root resolver. As with the equivalent `get_attributes()` call, the path is actually the root for the `ObjectTree` resolver.

`get_directory("/ObjectTree/some/other/path"):` Same as `get_directory("/ObjectTree")`, except that the path passed to the `ObjectTree` is now `/some/other/path`.

Path representation

Only the driver specific part of ONEDrive handles paths as strings. The bulk of the code handles paths as lists of path elements. The elements are strings or Unicode. They do not contain any escaped characters. The elements may contain characters that have special meaning in the filesystem, such as the path separator character ("`/`" on `*nix`). If so, these characters do NOT have the special meaning that they would have in a normal path string. When joining the segments together to a path string, the special characters would be escaped.

Normally, when splitting the root path, "`/`", one ends up with a list of two empty strings. The first empty string shows that the path is absolute (starting at root), and the second that there is nothing after root. In ONEDrive, all paths represented as lists of path segments are assumed to be rooted, so the first, empty, element is removed.

Callbacks

The FUSE callbacks and how these are handled.

getattr()

`getattr()` gets called on any path that the user attempts to access and any path that has previously been returned by `readdir()`. `getattr()` returns information, such as size, date and type, for a single item. In ONEDrive, the type of an item is either a file or a folder.

ONEDrive handles `getattr()` calls as follows:

1. The keys in the attribute cache are searched for a match to the path. If a match is found, the attributes for the file or folder are returned.
2. If the path was not found in the cache, `get_attributes()` is called in the root resolver.
3. `getattr()` caches the result, then returns it.

readdir()

`readdir()` is only called for folders. It returns the names of items in a folder. It does not return any other information, such as the type of the item. FUSE calls `getattr()` for each of the items returned by `readdir()` to get their type, size and other information.

FUSE assumes that the root, “/”, is a folder, so `getattr()` is not called for the root before `readdir()` is called on the root. This is the only exception to the general pattern of interactions between `getattr()` and `readdir()`.

By calling `getattr()` and `readdir()` in a cyclic pattern, FUSE recursively discovers the folder tree in the filesystem, the contents of the folders, and the sizes of both files and folders.

FUSE only calls `readdir()` on folders that were previously designated as folders and valid paths by `getattr()`.

ONEDrive handles `readdir()` calls as follows:

1. The keys in the directory cache (see [readdir\(\)](#)) are searched for a match to the path. If a match is found, the names of the contents for the folder are returned.
2. `readdir()` caches the result in the directory cache and returns it to FUSE.

Debugging

When first mounting ONEDrive, the filesystem will be hit with various automated requests in order for the OS to learn about the filesystem. This causes trouble when debugging. On Ubuntu, the automated requests can be disabled temporarily by killing the gvfs processes:

```
$ sudo pkill -9 -f gvfs
```

Future improvements

There's a lot more that can be done with Zotero integration if desired. For instance, ONEDrive could enable access to other information that can be stored in Zotero libraries, such as tags, notes and attached objects.

ONEDrive could detect updates in Zotero while it is running and dynamically update itself. Currently, ONEDrive only refreshes its caches during startup.

4.2.4 ONEDrive Mockups

For reference, these alternatives were considered for how ONEDrive should be implemented.

The initial implementation of ONEDrive was a simple proof of concept that enabled access to objects on a specific Member Node.

The second implementation was based on the `standalone` mockup below and allowed the user to perform searches by manipulating the filesystem path. Instead of representing a folder hierarchy, the path was used for specifying a faceted search. This system proved to be too complex to use. It also caused the filesystem to be virtually infinitely recursive, which caused problems for file managers and filesystem searches.

The third implementation was based on the `onemercury-integration` and implemented the concept of a DataONE Workspace.

The fourth implementation replaced the DataONE Workspace concept with the Zotero citation manager.

Contents:

Overview of mockups

In the mockups, filter operations and filter values are prefixed with “@” and “#” respectively. These decorators have two purposes. The first is to cause filter operations, filter values and results from previously applied filters to be displayed as separate groups in the filesystem when the files and folders are sorted alphabetically. The second is to make it easier for ONEDrive to parse the path when the file and folder names are returned to ONEDrive as path elements by the client. The filesystem path serves as the only channel of communication from the client to ONEDrive and there is no opportunity to do interpretation or translation on the client. Without the decorations, ONEDrive would have to keep track of more context to determine the semantics embedded in the path.

Member Node

Member Node filtering fits well in the filesystem. The mockup exposes it as a `@MemberNode` folder that appears in all folders in which a new filter can be started. Opening the folder exposes a list of Member Nodes. Selecting a Member Node applies the filter and brings the user back to a folder in which the resulting objects appear and the `@MemberNode` filter is no longer available. We can also implement an “OR” filter by leaving the Member Node filter available to be selected again.

Geographical Area

ONEMercury exposes the geographical area search in two ways, as names of continents/states/countries and as a bounding box defined by latitude and longitude. The first type maps pretty well to the folder hierarchy and the mockup exposes it as two hierarchies, one which allows the user to first select a continent then a state/country in that continent and another that allows selecting a state/country directly. Letting the user select latitude and longitude floating point numbers in a filesystem is tricky. It might involve having the user open one folder for each digit. Letting the user select the coordinates as degrees, minutes and seconds is more feasible. We could expose a system which lets the user define coordinates only to the granularity that they need. The user would first select the degrees for upper left and lower right coordinates in a list of numbers between 0 and 359. Then, if they wish, they can refine that by selecting the minutes in a list between 0 and 59, then the same for seconds. Also, only the numbers for which there are actually results within the currently filtered objects are displayed. The mockup illustrates these ideas.

Keyword

The only way to let the user type a keyword in filesystem based search/discovery would be to have them type it directly into the path, which I don't think is feasible. So the mockup shows a system where the user must know up front which keyword he wants. The idea is to have the user click through a hierarchy of groups until there are few enough keywords that they can be displayed directly in a list. The groups are displayed as folders named after the first and last keyword in the group. If the keyword filter is the first one that the user applies, my guess is that it will normally be 2-3 levels deep. If the keyword filter is applied after other filters, it may be just 0 or 1 levels deep (where 0 levels means that the keywords are displayed directly, without having to select groups first).

Date-time filtering

Date-time filtering is implemented in a way similar to the bounding box geographical area filtering. The goal is have the user filter only to the granularity that they need and select only from date-times for which there are existing objects. So a user that is searching for data up to and including 2005 can select @EndYear/#2005, but does not have to refine with month and day selections and if the current set of objects contain data only up to January and February 2005, only those months are displayed, with the same for year and day (Solr faceting is used for retrieving the options in a single query). The mockup illustrates this and shows how to make the user aware that refinements are available but optional. This is done by displaying the currently filtered list of objects plus other options for filtering together with each optional date-time refinement step.

As both start and end date-time filtering is available and objects can contain data for a period of time, I think that filtering should be applied in such a way that objects that contain data for a time period that has any overlap with the specified start and end date-time filter should be included in the filter. So, for instance, an object with data for 2005-2007 would be included in a search for objects for 2000-2005. And objects with data for 2000-2010 would be included in a search for objects for 2005.

From an implementation standpoint, the mockup also shows how ONEDrive can parse the path in such a way that an optional elements in the path are seen in the context of earlier elements.

Data package

The data package mockup aims to show the following:

- When a data package folder is opened, the entire contents of the package can be exposed as a bagit or zip file.
 - The same object can be exposed in multiple formats. For instance, an EML file may also be exposed as an HTML file.
 - The system metadata for all the objects is in a separate folder in order to keep the main folder from getting too crowded.

Transformations can be implemented as CN services or can be implemented directly in ONEDrive, maybe as a plugin system.

OS specific integration

As an extension of the standard filesystem, ONEDrive is by nature a platform specific user interface. Here are several user interface mockups that take advantage of the user interface approach on each platform on which we plan to operate.

Mac OS X

Mac OS X displays files using the Finder. The standard Finder view would be used to display a set of browse hierarchies representing the authors, locations, time periods, taxa, etc. associated with the data sets. At the leaves of this

folder hierarchy are Data Packages, each of which is represented as a folder containing the science metadata for the package in an HTML file and each of the data files in their natural format, and with an appropriate filename (e.g., with proper extension).

Basic folder view

The basic folder view shows only browse hierarchies that are sensible from a user perspective. These will need to be controlled hierarchies that are cleaned up from our metadata indexing corpus. The ONEDrive mount point is shown in the Finder window on the left.

Get Info Dialog used to display System Metadata

Package and file-oriented metadata that comes from the DataONE SystemMetadata corpus would be added as extended attributes to the file and folders, so that the standard Get Info dialog box can show these metadata fields. Note the presence of the DOI identifier listed in the middle.

Filtering approach 1: Spotlight

In our first UI for filtering, we use the built-in Mac OS X Spotlight filtering system that provides an UI for specifying search filters and applies them to the extended attributes of the items.

Filtering approach 2: Filter Dialog

Many users are unaware of the Spotlight filter UI described above, and don't naturally find it in the Finder interface. A potential alternative is to provide our own Filter Dialog that is accessed via a ONEDrive dropdown menu. The following two screenshots show first the dropdown menu and then a mockup of a potential filtering dialog box.

The use of a filtering dialog box gives us a lot of flexibility in laying out filter widgets, including the capability to use map widgets and other UI widgets to make constructing filters powerful. Once a filter is applied, the view in the ONEDrive window is constrained to display only packages that match the filter criteria.

ONEMercury Integration

Search/discovery for ONEDrive can be exposed in a web interface based on ONEMercury. The interface would let users search for and discover objects. After finding the objects, they would be accessed through ONEDrive.

ONEMercury

Opening objects

- Single objects in the search results can be opened directly with the local application to which the given filetype is associated.

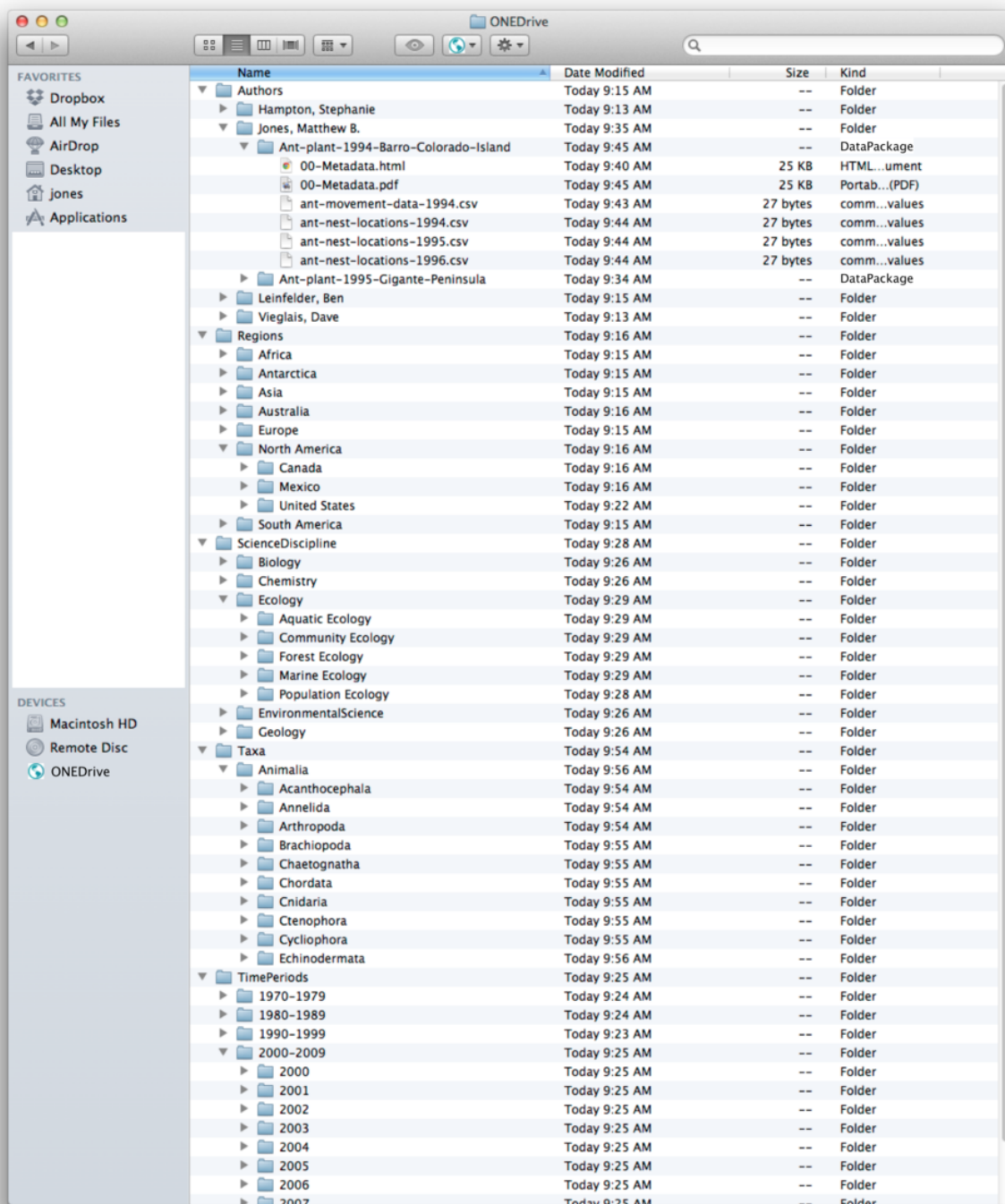


Fig. 1: *Figure 1.* Mac OS X Mockup of the hierarchical folder view.

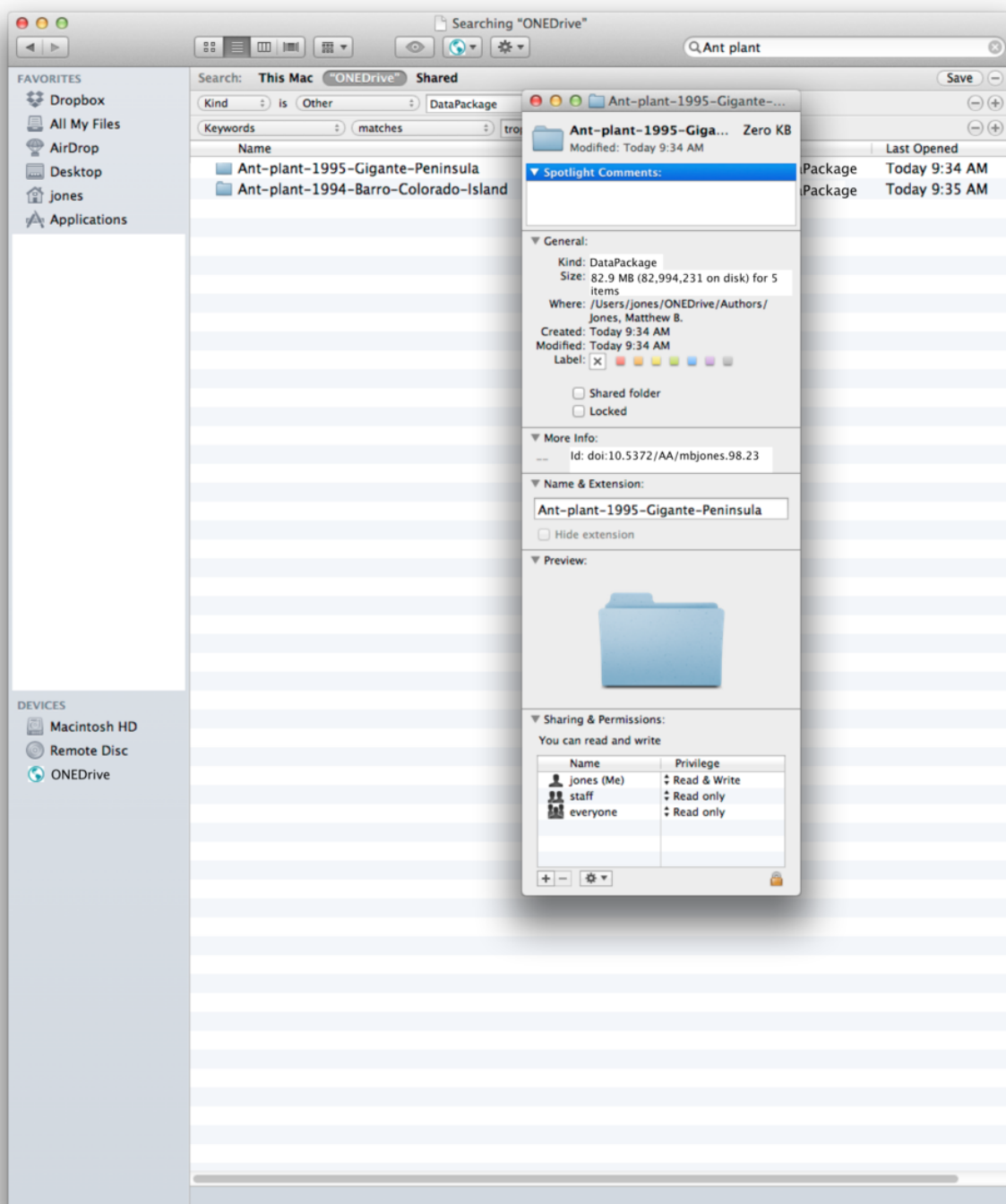


Fig. 2: Figure 2. Mac OS X Get Info dialog is used to display system metadata.

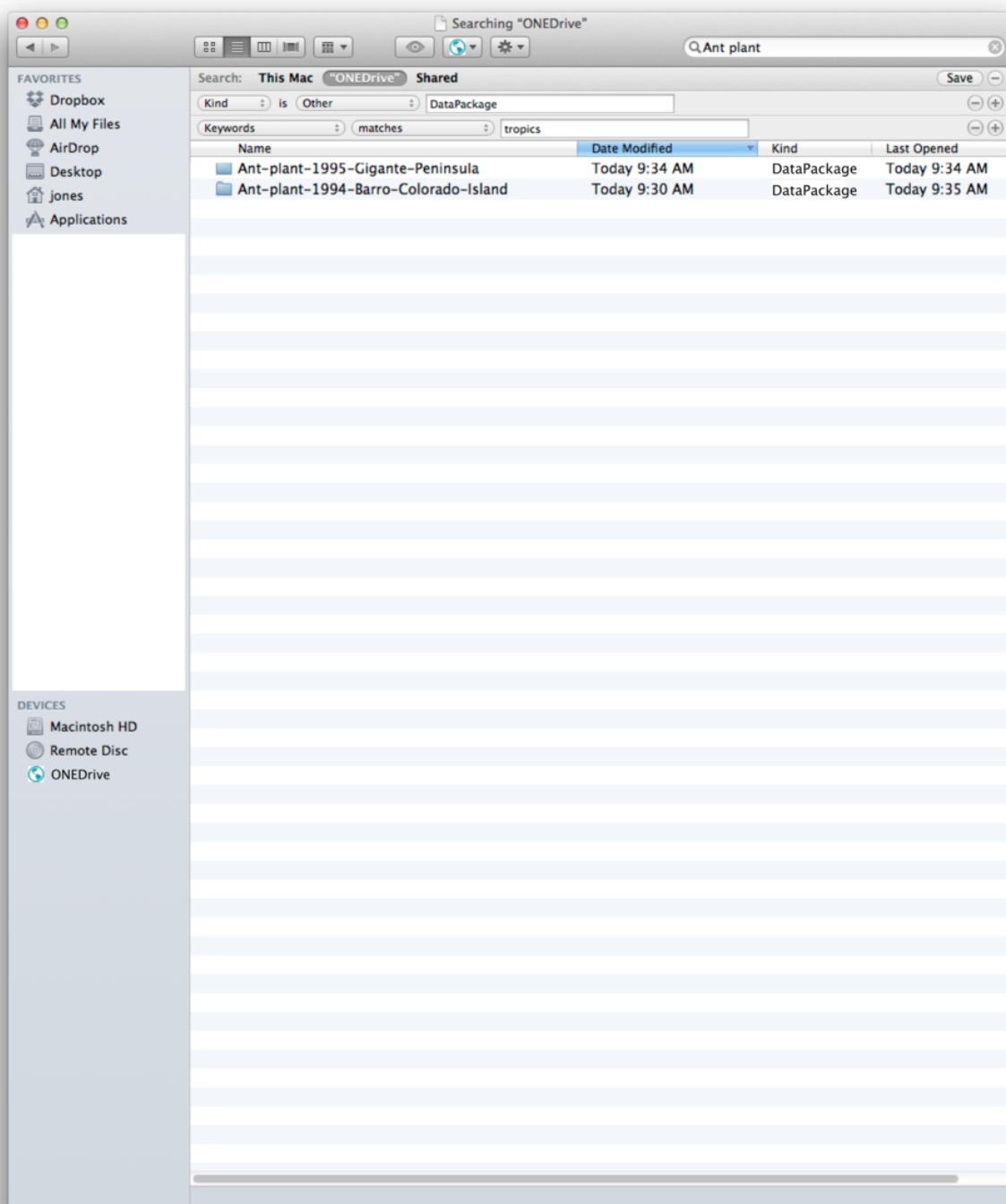


Fig. 3: *Figure 3.* Mac OS X Mockup of the filter UI showing a restricted set of data packages.

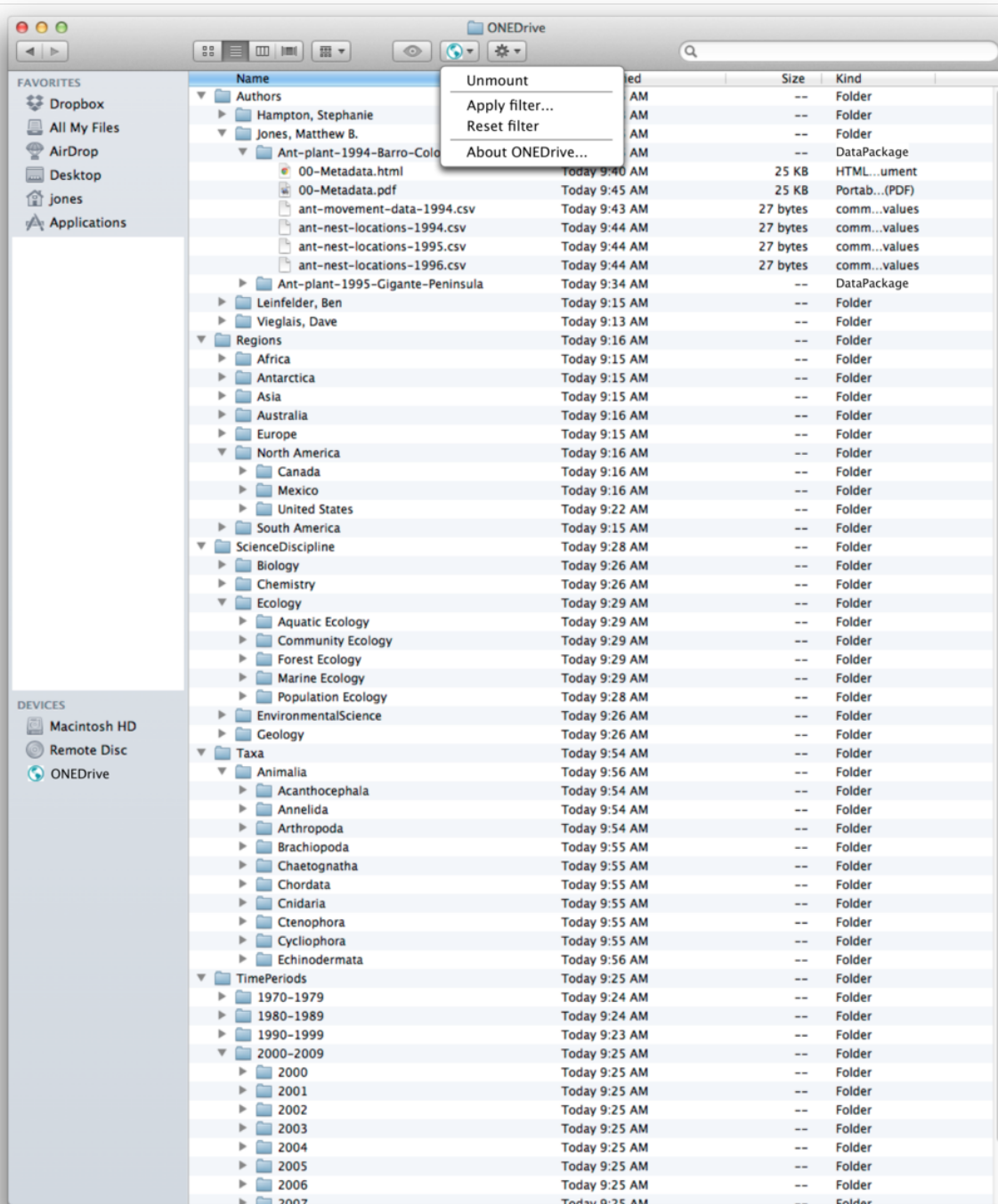


Fig. 4: Figure 1. Mac OS X Mockup showing the ONEDrive dropdown menu.

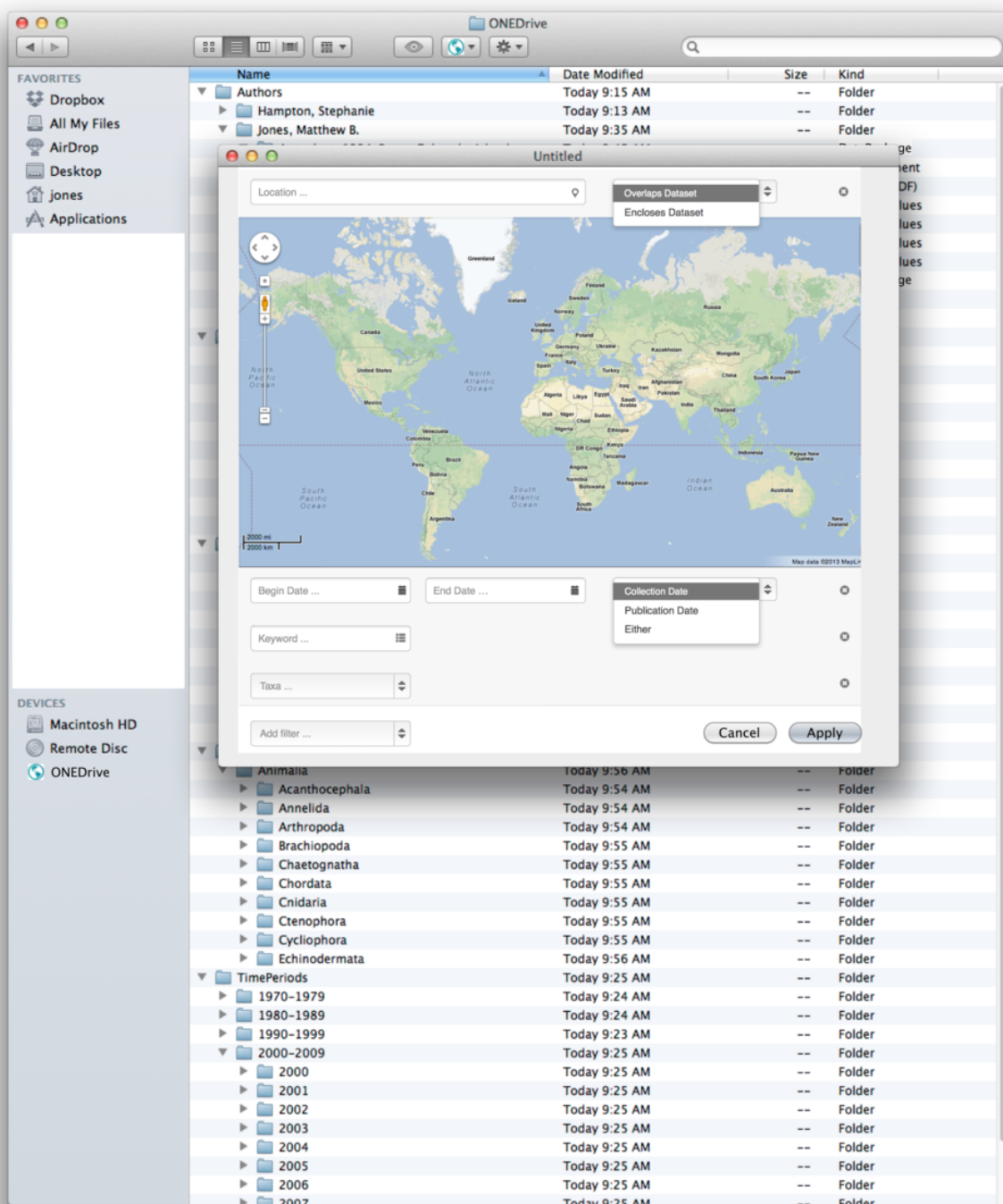
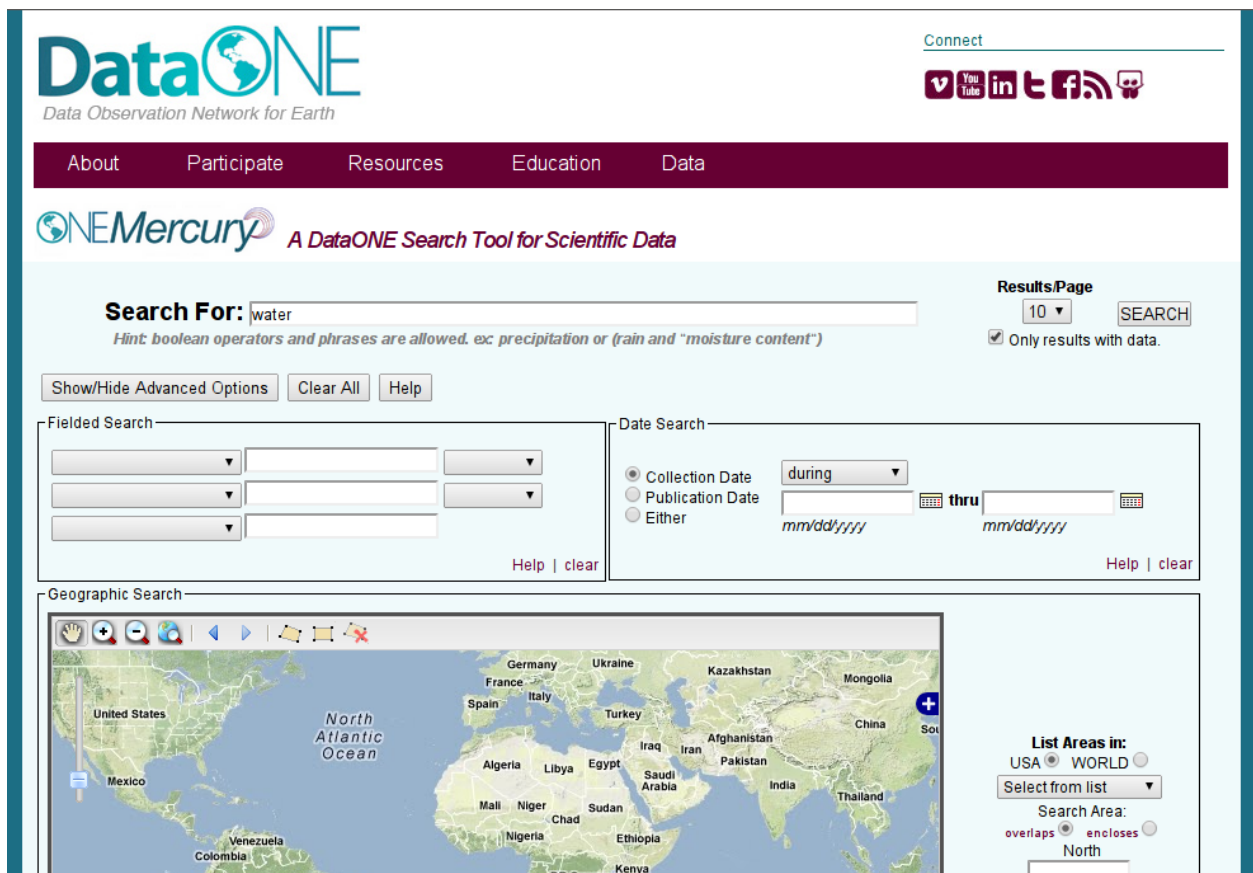


Fig. 5: *Figure 1.* Mac OS X Mockup showing the filter dialog for constraining what is shown in the window.



DataONE
Data Observation Network for Earth

Connect

About Participate Resources Education Data

ONEMercury A DataONE Search Tool for Scientific Data

Search For: water
Hint: boolean operators and phrases are allowed, ex: precipitation or (rain and "moisture content")

Results/Page: 10 **SEARCH**
☒ Only results with data.

Show/Hide Advanced Options Clear All Help

Fielded Search

Date Search

- ☒ Collection Date during
- ☐ Publication Date
- ☐ Either

mm/dd/yyyy thru mm/dd/yyyy

Help | clear

Geographic Search

List Areas in:
USA ☒ WORLD ☐
Select from list

Search Area:
☒ overlaps ☐ encloses
North

Fig. 6: Figure 1. Regular search/discovery in ONEMercury

The screenshot displays the DataONE web interface. At the top, there are two filter panels: 'Filter by author' and 'Filter by keywords'. The 'Filter by author' panel lists several California Department of Public Health (CDPH) districts and counties with their respective document counts. The 'Filter by keywords' panel lists terms like 'California', 'contaminants', 'drinking water', 'health', 'San Bernardino County', 'Los Angeles County', and 'Fresno County' with their counts. Below the filters, there are sorting options: 'Sort By: Relevance', 'Start Date', and 'Most Recent'. To the right, it says 'Viewing Documents 1 - 10 out of 16790' with pagination links 'Prev 1 2 3 4 5 6 7 8 9 10 Next'. The main content area shows a list of documents. The first document is 'Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-005: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 05 - GREENHILLS - UNTREATED.' The second document is 'Cal Water Service. 04/05/2002. DWSAP Assessment for 4900546-001: HAWKINS WATER CO-CAL WATER SERVICE (PUC) / WELL 02.' The third document is 'Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-006: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 06 - RUSHDEN - RAW.' The fourth document is 'Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-002: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 02 - WATT - RAW.' The fifth document is partially visible: 'Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-008: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 08 - MORSE - RAW.' Each document entry includes an identifier, a data source (Merritt Repository), and a brief description. At the bottom of each entry are four buttons: 'View full metadata', 'Download', 'Open', and 'Add to Folder'. The 'Open' button for the second document is highlighted with a red circle and a mouse cursor.

Hide Filters

Filter by author

- CDPH Merced District (810)
- CDPH San Bernardino District (730)
- Cal Water Service (713)
- CDPH Sonoma District (652)
- CDPH Tehachapi District (477)
- San Bernardino County (466)

Filter by keywords

- California (16788)
- contaminants (16788)
- drinking water (16788)
- health (16788)
- San Bernardino County (1391)
- Los Angeles County (1297)
- Fresno County (954)

Sort By: **Relevance** Start Date Most Recent

Viewing Documents 1 - 10 out of 16790
Prev 1 2 3 4 5 6 7 8 9 10 Next

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-005: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 05 - GREENHILLS - UNTREATED.
Identifier: ark:/13030/m5z60p9f/1/cadwsap-s3410003-005.xml **Datasource:** Merritt Repository
A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
View full metadata Download **Open** Add to Folder

Cal Water Service. 04/05/2002. DWSAP Assessment for 4900546-001: HAWKINS WATER CO-CAL WATER SERVICE (PUC) / WELL 02.
Identifier: ark:/13030/m5mk6bww/1/cadwsap-s4900546-001.xml **Datasource:** Merritt Repository
A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-006: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 06 - RUSHDEN - RAW.
Identifier: ark:/13030/m5513zdc/1/cadwsap-s3410003-006.xml **Datasource:** Merritt Repository
A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-002: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 02 - WATT - RAW.
Identifier: ark:/13030/m5qv3mm/1/cadwsap-s3410003-002.xml **Datasource:** Merritt Repository
A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-008: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 08 - MORSE - RAW.

Fig. 7: Figure 2. Opening a single object

Saving objects

- A single object can be added to user defined folders.
- The user can use the “New folder” option to a new folder and add the first object to it in the same operation.

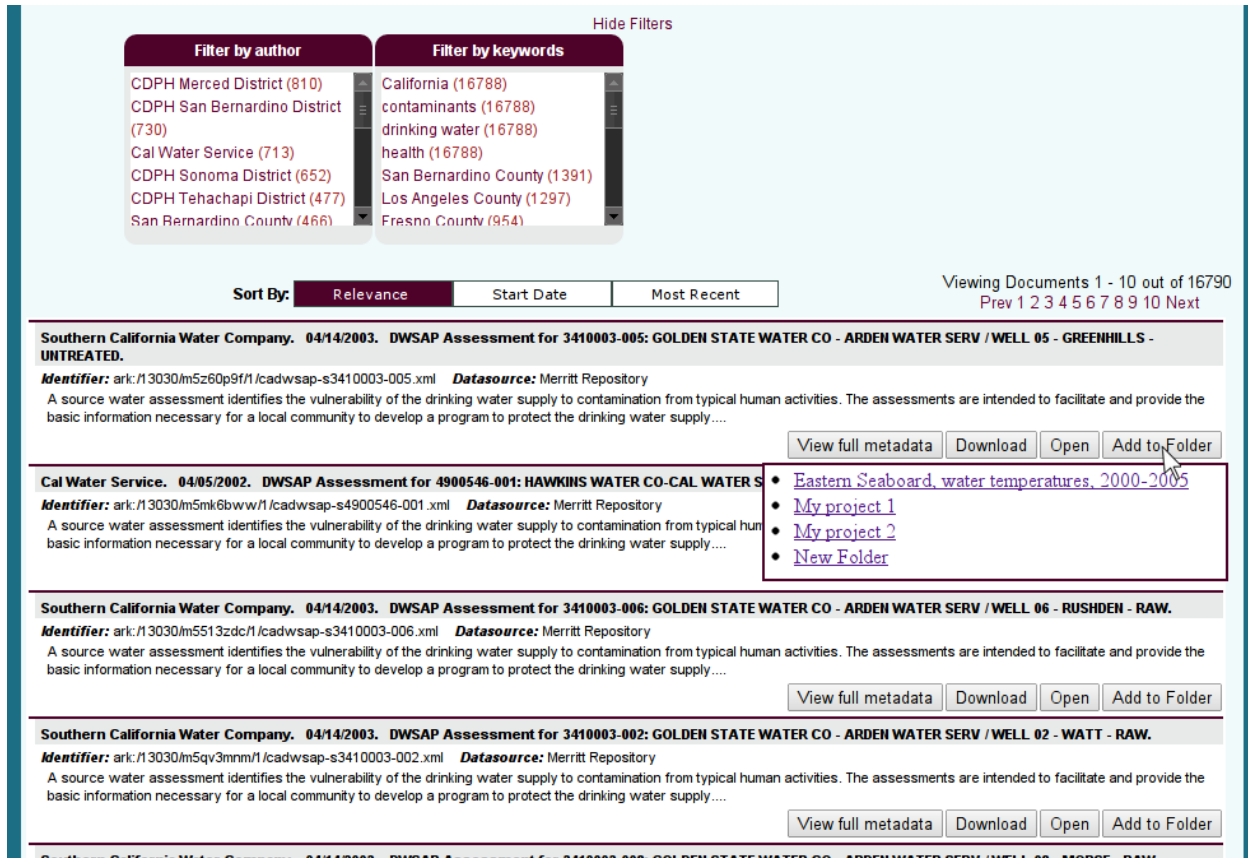


Fig. 8: Figure 3. Saving a single object

- All of the search results can be added to a folder.
- It is also possible to add a function to create an RSS or Atom feed for the search results so that the user can be notified when new objects that match the search parameters are added to DataONE.
- The folders created in ONEMercury and the search results stored in them become visible in ONEDrive.

View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-008: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 08 - MORSE - RAW.
Identifier: ark:/13030/m5sq90m7/1/cadwsap-s3410003-008.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Cal Water Service. 04/05/2002. DWSAP Assessment for 4900546-002: HAWKINS WATER CO-CAL WATER SERVICE (PUC) / WELL 01 - STANDBY.
Identifier: ark:/13030/m5gb22vz/1/cadwsap-s4900546-002.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-004: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 04 - WINDSOR - RAW.
Identifier: ark:/13030/m5m372c/1/cadwsap-s3410003-004.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Southern California Water Company. 04/14/2003. DWSAP Assessment for 3410003-001: GOLDEN STATE WATER CO - ARDEN WATER SERV / WELL 01 - SHADOWGLEN - RAW.
Identifier: ark:/13030/m5p26z95/1/cadwsap-s3410003-001.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Cal Water Service. 03/13/2002. DWSAP Assessment for 0410002-048: CAL-WATER SERVICE CO.-CHICO / WELL 48-01.
Identifier: ark:/13030/m5mg7ngk/1/cadwsap-s0410002-048.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Twentynine Palms Water District. 10/20/2003. DWSAP Assessment for 3610049-013: TWENTYNINE PALMS WATER DIST / WELL 13.
Identifier: ark:/13030/m5zs2ww5/1/cadwsap-s3610049-013.xml **Datasource:** Merritt Repository
 A source water assessment identifies the vulnerability of the drinking water supply to contamination from typical human activities. The assessments are intended to facilitate and provide the basic information necessary for a local community to develop a program to protect the drinking water supply....
 View full metadata Download Open Add to Folder

Viewing Documents 1 - 10 out of 16790
 Prev 1 2 3 4 5 6 7 8 9 10 Next

Add All To Folder

DataONE is a collaboration among many partner organizations, and is funded by the US National Science Foundation (NSF) under a Cooperative Agreement.
 1312 Basehart SE 1 University of New Mexico Albuquerque, NM 87106 [Contact Us](#) | [Site Map](#)

Fig. 9: Figure 4. Adding search results to folder

Tree:

Name

fs

dahl

Easter Seaboard, water temper...

My project 1

My project 2

Name	Ext	Size	Modified	rw
ark_13030_m5mk6bww_1_cad...	xml	4,676	2005-05-23 11:12	r...
ark_13030_m5z60p9f_1_cadws...	xml	34,822	2005-05-23 11:12	r...
scimeta_1067	xml	9,573	2005-05-23 11:12	r...

Fig. 10: Figure 5. ONEDrive view of files and folders

Standalone

In this approach, search and discovery is exposed directly as a filesystem hierarchy within ONEDrive. This is done by changing the semantics of files and folders. Instead of folders being containers of files, they are used as filters and filter parameters. The files become DataONE objects that match the currently applied filters.

Main advantages:

- Search and discovery functionality is in the same place as object access.
- Virtually the same code base can be used for all supported platforms.

Main disadvantages:

- Giving folders different semantics than they have in a regular filesystem is unintuitive.
- There is a learning curve in interpreting and navigating the search/discovery filesystem.
- Input data such as keywords cannot be typed in – they can only be selected.

Mockups

In the mockups, filter operations and filter values are prefixed with “@” and “#” respectively. These decorators have two purposes. The first is to cause filter operations, filter values and results from previously applied filters to be displayed as separate groups in the filesystem when the files and folders are sorted alphabetically. The second is to make it easier for ONEDrive to parse the path when the file and folder names are returned to ONEDrive as path elements by the client. The filesystem path serves as the only channel of communication from the client to ONEDrive and there is no opportunity to do interpretation or translation on the client. Without the decorations, ONEDrive would have to keep track of more context to determine the semantics embedded in the path.

Member Node

Member Node filtering fits well in the filesystem. The mockup exposes it as a @MemberNode folder that appears in all folders in which a new filter can be started. Opening the folder exposes a list of Member Nodes. Selecting a Member Node applies the filter and brings the user back to a folder in which the resulting objects appear and the @MemberNode filter is no longer available. We can also implement an “OR” filter by leaving the Member Node filter available to be selected again.

Geographical Area

ONEMercury exposes the geographical area search in two ways, as names of continents/states/countries and as a bounding box defined by latitude and longitude. The first type maps pretty well to the folder hierarchy and the mockup exposes it as two hierarchies, one which allows the user to first select a continent then a state/country in that continent and another that allows selecting a state/country directly. Letting the user select latitude and longitude floating point numbers in a filesystem is tricky. It might involve having the user open one folder for each digit. Letting the user select the coordinates as degrees, minutes and seconds is more feasible. We could expose a system which lets the user define coordinates only to the granularity that they need. The user would first select the degrees for upper left and lower right coordinates in a list of numbers between 0 and 359. Then, if they wish, they can refine that by selecting the minutes in a list between 0 and 59, then the same for seconds. Also, only the numbers for which there are actually results within the currently filtered objects are displayed. The mockup illustrates these ideas.

Keyword

The only way to let the user type a keyword in filesystem based search/discovery would be to have them type it directly into the path, which I don't think is feasible. So the mockup shows a system where the user must know up front which keyword he wants. The idea is to have the user click through a hierarchy of groups until there are few enough keywords that they can be displayed directly in a list. The groups are displayed as folders named after the first and last keyword in the group. If the keyword filter is the first one that the user applies, my guess is that it will normally be 2-3 levels deep. If the keyword filter is applied after other filters, it may be just 0 or 1 levels deep (where 0 levels means that the keywords are displayed directly, without having to select groups first).

Date-time filtering

Date-time filtering is implemented in a way similar to the bounding box geographical area filtering. The goal is have the user filter only to the granularity that they need and select only from date-times for which there are existing objects. So a user that is searching for data up to and including 2005 can select @EndYear/#2005, but does not have to refine with month and day selections and if the current set of objects contain data only up to January and February 2005, only those months are displayed, with the same for year and day (Solr faceting is used for retrieving the options in a single query). The mockup illustrates this and shows how to make the user aware that refinements are available but optional. This is done by displaying the currently filtered list of objects plus other options for filtering together with each optional date-time refinement step.

As both start and end date-time filtering is available and objects can contain data for a period of time, I think that filtering should be applied in such a way that objects that contain data for a time period that has any overlap with the specified start and end date-time filter should be included in the filter. So, an object with data for 2005-2007 would be included in a search for objects for 2000-2005. And an object with data for 2000-2010 would be included in a search for objects for 2005.

From an implementation standpoint, the mockup also shows how ONEDrive can parse the path in such a way that an optional elements in the path are seen in the context of earlier elements.

4.2.5 Creating the installer for Windows

ONEDrive supports Microsoft Windows. The Windows distribution installs ONEDrive as a regular application, independent of any existing Python environment on the computer. These instructions detail how to create the installer and is intended as a reference for DataONE and 3rd party developers.

The regular distribution channel for DataONE's Python products is PyPI, but the PyPI distribution does not include various files needed for building the installer.

To create the ONEDrive installer for Windows, ONEDrive is first installed from the DataONE Subversion repository into a regular Python environment.

Then, a stand-alone, executable version of ONEDrive is created with py2exe.

Finally, an installer is built for the executable and all dependencies.

1. If you do not already have a working 32-bit Python 3.6 environment, download the latest 32-bit Python 3.6 Windows installer from <http://www.python.org/download/> and install it.

Open a command prompt:

```
> setx path "%path%;C:\Python27;C:\Python27\Scripts"
```

Close then reopen the command prompt (to activate the new path).

1. Install pip:

Download get-pip.py:

```
https://pip.pypa.io/en/latest/installing.html
```

Go to the download location of get-pip.py.

E.g.:

```
> cd \Users\Myself\Downloads
```

Install and update pip:

```
> python get-pip.py
> python -m pip install -U pip
```

1. Open a Command Prompt.
2. Install the DataONE Client Library for Python and dependencies:

```
> pip install dataone.libclient
```

1. Download and install Subversion for Windows from <http://sourceforge.net/projects/win32svn/>
2. Download and install the 32-bit Python 3.6 py2exe from <http://sourceforge.net/projects/py2exe/files/py2exe/>
3. Open a new Command Prompt (to pick up new path to the svn command).
4. Create a work area on disk. Below, C:\onedrive is used for this. To use another folder replace all the references to the folder below.

```
> mkdir c:\onedrive
> c:
> cd \onedrive
> svn co https://repository.dataone.org/software/cicore/trunk/itk/dl_client_onedrive/
↪src/ .
```

1. Start ONEDrive and verify that it works:

```
> src\dl_client_onedrive\onedrive.py
```

Access the ONEDrive filesystem and check that the folder hierarchy can be traversed and that the DataONE objects can be accessed.

Exit with ctrl-break.

1. Build a stand-alone version of ONEDrive:

```
> cd src
> setup.py py2exe
```

A list of missing modules will be printed. These are not used by ONEDrive.

1. Verify that the exe version of ONEDrive works:

```
> cd dist
> onedrive.exe
```

Access the ONEDrive filesystem and check that the folder hierarchy can be traversed and that the DataONE objects can be accessed.

Exit with ctrl-break.

1. Download and install the stable release of Inno Setup from: <http://www.jrsoftware.org/isdl.php#stable>

Open the Inno Setup script:

```
> cd \onedrive
> onedrive-setup.iss
```

In the script, update the version number so that it matches the version number displayed when ONEDrive was started in a previous step.

Build the installer by selecting **Compile and Build** in the main menu.

The finished installer will be in `C:\onedrive\src\Output`.

4.2.6 API

d1_onedrive package

DataONE ONEDrive.

Subpackages

d1_onedrive.impl package

Subpackages

d1_onedrive.impl.clients package

Submodules

d1_onedrive.impl.clients.onedrive_d1_client module

DataONE Client with caching.

```
class d1_onedrive.impl.clients.onedrive_d1_client.DataONEClient (options)
    Bases: object
```

```
    get_science_object (pid)
```

```
    get_system_metadata (pid)
```

This method causes an implicit validation of the retrieved System Metadata.

```
    get_system_metadata_as_string (pid)
```

This method does not include validation of the System Metadata.

```
    describe (pid)
```

d1_onedrive.impl.clients.onedrive_solr_client module

Extend `d1_client.solr_client.SolrClient` for OneDrive.

```
class d1_onedrive.impl.clients.onedrive_solr_client.OneDriveSolrClient (options,
                                                                    try_count=3)
    Bases: d1_client.solr_client.SolrClient
```

```
run_solr_query (query, filter_queries=None, fields=None)  
get_solr_record (pid)
```

d1_onedrive.impl.clients.onedrive_zotero_client module

Hold a local cache of the online Zotero library.

Provide overview of what has changed in the online Zotero library since last sync.

Expose a simple API to query and refresh the cache.

```
class d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient (options)  
    Bases: object  
  
    refresh ()  
        Refresh the local cache of the online Zotero library if stale.  
  
    force_refresh ()  
  
    get_filtered_sub_tree (path)  
        Get a sub-tree rooted at [path] that contains only DataONE items.  
        The path is a list of collection names.  
  
    iterate_collection_trees ()  
  
    iterate_collection_tree (collection_tree, path=None)  
  
    iterate_filtered_tree (filtered_tree=None, path=None)  
  
    cache_is_stale ()  
  
    create_filtered_tree ()
```

d1_onedrive.impl.clients.query_engine_description module

Cache and manipulate a queryEngineDescription.

d1_onedrive.impl.drivers package

Subpackages

d1_onedrive.impl.drivers.dokan package

ONEDrive filesystem drivers.

Submodules

d1_onedrive.impl.drivers.dokan.const module

Constants used by the DataONE drive for Windows which is built using Dokan.

d1_onedrive.impl.drivers.dokan.d1_dokan module

d1_onedrive.impl.drivers.dokan.dokan module

d1_onedrive.impl.drivers.dokan.fs_util module

d1_onedrive.impl.drivers.dokan.solrclient module

d1_onedrive.impl.drivers.fuse package

ONEDrive FUSE driver.

Submodules

d1_onedrive.impl.drivers.fuse.callbacks module

d1_onedrive.impl.drivers.fuse.d1_fuse module

d1_onedrive.impl.resolver package

ONEDrive resolvers.

Submodules

d1_onedrive.impl.resolver.author module

Resolve a filesystem path pointing into an Authors controlled hierarchy.

```
class d1_onedrive.impl.resolver.author.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver
    get_attributes (object_tree_folder, path)
    get_directory (object_tree_folder, path)
    read_file (object_tree_folder, path, size, offset)
```

d1_onedrive.impl.resolver.d1_object module

Resolve a DataONE object.

Determine what type of DataONE object a given PID references and branch out to a resolver that is specialized for that type.

```
class d1_onedrive.impl.resolver.d1_object.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver
    get_attributes (object_tree_root, path)
    get_directory (object_tree_root, path)
    read_file (object_tree_root, path, size, offset)
```

d1_onedrive.impl.resolver.flat_space module

Resolve flat space.

Resolve a filesystem path that points to a directory to the contents of the directory by querying the query engine.

```

class d1_onedrive.impl.resolver.flat_space.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver
    get_attributes (object_tree_root, path)
    get_directory (object_tree_root, path)
    read_file (object_tree_root, path, size, offset)

```

d1_onedrive.impl.resolver.object_tree_resolver module

Resolve object tree.

Resolve a filesystem path that points to a directory to the contents of the directory by querying the query engine.

```

class d1_onedrive.impl.resolver.object_tree_resolver.Resolver (options,      ob-
                                                                ject_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver
    get_attributes (object_tree_root, path)
    get_directory (object_tree_root, path, preconfigured_query=None)
    read_file (object_tree_root, path, size, offset)

```

d1_onedrive.impl.resolver.region module

Resolve region.

Resolve a filesystem path pointing into a Region controlled hierarchy.

```

class d1_onedrive.impl.resolver.region.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver
    get_attributes (object_tree_folder, path)
    get_directory (object_tree_folder, path)
    read_file (object_tree_folder, path, size, offset)

```

d1_onedrive.impl.resolver.resolver_base module

Base resolver.

Abstract Base Class (ABC) for the resolvers.

The resolvers are a class of objects that translate filesystem paths to their corresponding files and folders.

```

class d1_onedrive.impl.resolver.resolver_base.Resolver (options, object_tree)
    Bases: object
    is_root (path)

```

d1_onedrive.impl.resolver.resource_map module

Resolve resource map.

Resolve a filesystem path pointing to a resource map.

```
class d1_onedrive.impl.resolver.resource_map.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver

    get_attributes (object_tree_root, path)
    get_directory (object_tree_root, path)
    read_file (object_tree_root, path, size, offset)
```

d1_onedrive.impl.resolver.root module

Resolve root.

The root of ONEDrive is a list of folders, designating different types of interactions which can be performed with the DataONE infrastructure. The root resolver renders the folders and transfers control to the appropriate path resolver, based on the path which is entered.

The root resolver unescapes path entries before they are passed into the resolver hierarchy and escapes entries that are received.

```
class d1_onedrive.impl.resolver.root.RootResolver (options, object_tree_client)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver

    get_attributes (path)
    get_directory (path)
    read_file (path, size, offset)
```

d1_onedrive.impl.resolver.single module

Resolve single.

This resolver simply renders all objects into a single folder.

```
class d1_onedrive.impl.resolver.single.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver

    get_attributes (object_tree_root, path)
    get_directory (object_tree_root, path)
    read_file (object_tree_root, path, size, offset)
```

d1_onedrive.impl.resolver.taxa module

Resolve a filesystem path pointing into a Taxa controlled hierarchy.

```
class d1_onedrive.impl.resolver.taxa.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver

    get_attributes (object_tree_folder, path)
    get_directory (object_tree_folder, path)
```


`read_file (object_tree_folder, path, size, offset)`

d1_onedrive.impl.resolver.time_period module

Resolve time period.

Resolve a filesystem path pointing into a TimePeriod controlled hierarchy.

```
class d1_onedrive.impl.resolver.time_period.Resolver (options, object_tree)
    Bases: d1_onedrive.impl.resolver.resolver_base.Resolver

    get_attributes (object_tree_folder, path)
    get_directory (object_tree_folder, path)
    read_file (object_tree_folder, path, size, offset)
```

d1_onedrive.impl.tests package

ONEDrive tests.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

d1_onedrive.impl.tests.command_echoer module

Echo commands back for unit testing / TDD.

```
class d1_onedrive.impl.tests.command_echoer.CommandEchoer
    Bases: object

    solr_query_raw (query_string)
    solr_query (applied_facets=None, filter_queries=None)
    get_science_object_through_cache (pid)
    get_system_metadata_through_cache (pid)
```

d1_onedrive.impl.tests.object_tree_test_sample module

d1_onedrive.impl.tests.test_author_resolver module

d1_onedrive.impl.tests.test_d1_object_resolver module

d1_onedrive.impl.tests.test_disk_cache module

d1_onedrive.impl.tests.test_flat_space_resolver module

d1_onedrive.impl.tests.test_os_escape module

d1_onedrive.impl.tests.test_region_resolver module

d1_onedrive.impl.tests.test_resource_map_resolver module

d1_onedrive.impl.tests.test_root_resolver module

d1_onedrive.impl.tests.test_solr_client module

d1_onedrive.impl.tests.test_taxa_resolver module

d1_onedrive.impl.tests.test_time_period_resolver module

d1_onedrive.impl.tests.test_util module

d1_onedrive.impl.tests.test_zotero_client module

Submodules

d1_onedrive.impl.attributes module

Hold the size and other attributes for a file or folder.

```
class d1_onedrive.impl.attributes.Attributes (size=0, date=None, is_dir=False)
    Bases: object

    size()
    date()
    is_dir()
```

d1_onedrive.impl.cache_memory module

Cache Python objects in memory.

Cache Python objects with a limit on how many objects can be cached. When the cache reaches a configured size, adding a new object causes the oldest object to be removed. The oldest object is the object that was added first of the objects still in the cache.

```
class d1_onedrive.impl.cache_memory.Cache (max_items)
    Bases: dict

    keys() → a set-like object providing a view on D's keys
    copy() → a shallow copy of D
    log_dump()
```

d1_onedrive.impl.directory module

Hold the list of files and folders for a directory.

```
class d1_onedrive.impl.directory.Directory (init_list=None, disable_cache=False)
    Bases: collections.abc.MutableSequence
```

```
insert (i, v)
    S.insert(index, value) – insert value before index
```

d1_onedrive.impl.disk_cache module

Cache Python objects on disk.

Cache Python objects with a limit on how many objects can be cached. When the cache reaches a configured size, adding a new object causes the oldest object to be removed. The oldest object is the object that was added first of the objects still in the cache.

```
class d1_onedrive.impl.disk_cache.DiskCache (max_items, cache_directory_path)
    Bases: dict

    clear () → None. Remove all items from D.
```

d1_onedrive.impl.log_decorator module

```
class d1_onedrive.impl.log_decorator.log_func
    Bases: object
```

d1_onedrive.impl.object_tree module

Object Tree.

Based on a source tree that contains only PIDs and queries, maintain the object tree that is browsed through the ONEDrive filesystem.

Cache the information on disk between runs of ONEDrive.

```
class d1_onedrive.impl.object_tree.ObjectTree (options, source_tree)
    Bases: object

    refresh ()
        Synchronize the local tree of Solr records for DataONE identifiers and queries with the reference tree.

    get_folder (path, root=None)
        Get the contents of an object tree folder.

    get_object_tree_folder_name (object_tree_folder)

    get_object_record (pid)
        Get an object that has already been cached in the object tree.

        Caching happens when the object tree is refreshed.

    get_object_record_with_sync (pid)
        Get an object that may not currently be in the cache.

        If the object is not in the cache, an attempt is made to retrieve the record from a CN on the fly. If the object is found, it is cached before being returned to the user. This allows the object tree caching system to be used for objects that are not in the object tree. ONEDrive uses this functionality for the FlatSpace folder.

    add_object_to_cache (pid)
        Attempt to add a specific object to the cache.

        Objects are normally only added to the object tree during refresh. This method is used by the FlatSpace resolver.
```

```
get_science_object(pid)
get_system_metadata(pid)
get_source_tree_folder(path)
sync_cache_with_source_tree()
```

d1_onedrive.impl.onedrive_exceptions module

Type that gets raised as exception for invalid paths.

```
exception d1_onedrive.impl.onedrive_exceptions.PathException(message)
    Bases: Exception

exception d1_onedrive.impl.onedrive_exceptions.ONEDriveException(message)
    Bases: Exception

exception d1_onedrive.impl.onedrive_exceptions.NoResultException(message="")
    Bases: Exception
```

d1_onedrive.impl.os_escape module

OS specific escaping.

Escape DataONE identifiers, so that they follow the filesystem conventions and restrictions for valid file- and folder names on a given OS.

Unescape the escaped file- and folder names to arrive at the original DataONE identifiers.

DataONE supports identifiers containing characters that cannot (or should not) be represented directly within the file- and folder names in a filesystem, such as the path separator character (“/” on *nix). This module contains functions for converting between DataONE identifiers and names that are suitable for use as file- and folder names on the operating systems that are supported by ONEDrive.

It is necessary to be able to get the original DataONE identifier given an escaped name, so the escaping must be reversible. This is because the filesystem driver (e.g., FUSE), passes the escaped name back to ONEDrive when the user interacts with the file or folder.

The current implementation simply uses URL percent-encoding of an utf-8 encoding of the Unicode strings.

Quote and unquote are somewhat borrowed from python urllib standard library.

```
d1_onedrive.impl.os_escape.unquote('abc%20def') → 'abc def'.
```

```
d1_onedrive.impl.os_escape.quote(s, unsafe='/')
```

Pass in a dictionary that has unsafe characters as the keys, and the percent encoded value as the value.

```
d1_onedrive.impl.os_escape.posix_filename_from_identifier(identifier)
```

```
d1_onedrive.impl.os_escape.posix_identifier_from_filename(filename)
```

```
d1_onedrive.impl.os_escape.windows_filename_from_identifier(identifier)
```

On Windows, the following characters are not allowed:

```
/ : * ? " < > |
```

```
d1_onedrive.impl.os_escape.windows_identifier_from_filename(filename)
```

```
d1_onedrive.impl.os_escape.identifier_from_filename(filename)
```

```
d1_onedrive.impl.os_escape.filename_from_identifier(identifier)
```

d1_onedrive.impl.util module

Misc utilities that don't fit anywhere else.

```
d1_onedrive.impl.util.log_dump(s)
d1_onedrive.impl.util.ensure_dir_exists(path)
d1_onedrive.impl.util.string_from_path_elements(path)
d1_onedrive.impl.util.is_root(path)
d1_onedrive.impl.util.os_format(txt)
```

Submodules

d1_onedrive.onedrive module

Top level module for ONEDrive.

- Parse arguments
- Instantiate the Root resolver
- Mount FUSE / Dokan

```
d1_onedrive.onedrive.main()
d1_onedrive.onedrive.log_setup(options)
d1_onedrive.onedrive.log_version()
d1_onedrive.onedrive.log_startup_parameters(options, arguments)
```

4.2.7 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.3 DataONE Command Line Interface (CLI)

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

The *DataONE Command Line Interface (CLI)* enables operations to be performed against the DataONE infrastructure from the command line. Supported operations include creating and retrieving DataONE objects, searching, updating access control rules and retrieving statistics.

Contents:

4.3.1 Installing the DataONE CLI

The *DataONE Command Line Interface (CLI)* enables operations to be performed against the DataONE infrastructure from the command line. Supported operations include creating and retrieving DataONE objects, searching, updating access control rules and retrieving statistics.

Install

The CLI is distributed via PyPI, the Python Package Index.

Set up OS packages required by some of the CLI's PyPI distributed dependencies. This includes a build environment for DataONE Python extensions.

```
$ sudo apt install --yes build-essential python-dev libssl-dev \
libxml2-dev libxslt-dev openssl curl
```

Install pip:

```
$ sudo apt install --yes python-pip; sudo pip install pip --upgrade
```

Install the CLI from PyPI:

```
$ sudo pip install dataone.cli
```

4.3.2 Examples

Note: Use the Arrow Up and Arrow Down keys to find commands in the command history. These can then be edited and run again.

Viewing and manipulating the session variables

Viewing and manipulating the *session variables* used when performing operations against the *DataONE* infrastructure via the DataONE Command Line Interface (CLI).

If desired, the session variables can be *reset* back to their defaults (useful if they were modified by an existing `.dataone_cli.conf` file at startup):

```
> reset
```

Set the authentication session variables for authenticated access using a certificate from CILogon (downloaded to the default location in `/tmp`):

```
> set anonymous false
> set cert-file none
> set key-file none
```

Or set to use a certificate in a non-standard location:

```
> set cert-file /etc/dataone/d1_client/certs/myclientcert.pem
```

View all the session variables:

```
> set
```

Save the session variables to a file for later use:

```
> save ~/d1/mysettings
```

Exit the CLI:

```
> exit
```

Searching for Science Data

A scientist can discover and download Science Data to leverage them in their own research.

Load the session variables from the file created in the previous step:

```
> load ~/dl/mysettings
```

View the session variables:

```
> set
```

Perform an unlimited search:

```
> set query *.*
> search
```

Restrict the search to a specific time or later:

```
> set from-date 1998-01-01T05:00:00
> search
```

Modify the search parameters to find only Science Data that originated from the “DEMO3” *MN* and search:

```
> set query origin_mn:DEMO3
> search
```

The search terms can also be specified after the “search” command:

```
> search barnacle
```

Modify the search parameters to find only Science Data that are of type text/csv and search again:

```
> set format-id text/csv
> search barnacle
```

Downloading Science Data Objects

View the session variables:

```
> set
```

Set *MN* from which to download the Science Data Object:

```
> set mn-url https://dataone.member.node.com/mn/
```

Download Science Data Object and save to local file:

```
> get hdl:10255/dryad.669/mets.xml ~/my_dataone_files/dryad669.xml
```

Downloading System Metadata

System Metadata is an XML document that contains additional information about a Science Data Object.

Retrieve the System Metadata and display it:

```
> meta hdl:10255/dryad.669/mets.xml
```

Retrieve the System Metadata and save it to a file:

```
> meta hdl:10255/dryad.669/mets.xml ~/dl/dryad669_system_metadata.xml
```

Downloading an access restricted object

- Authenticate with CILogon, at <https://cilogon.org/?skin=DataONE>

Tell the CLI that you wish to use authentication:

```
> set anonymous False
```

- Download an object for which you have read access:

```
> get my-access-controlled-identifier
```

See *Authentication* for more information.

Uploading Science Data Objects

A scientist can upload a set of Science Data to benefit from the services provided by DataONE.

Select *MN* to which to upload the Science Data Object:

```
> set mn-url https://dataone.member.node.com/mn/
```

Configure the session variables used when generating *System Metadata*:

```
> set rights-holder CN=MATTJTEMP,DC=dataone,DC=org
> set origin-mn DEMO1
> set authoritative-mn DEMO1
```

Create an Access Policy that has only public read permissions:

```
> clearaccess
> allowaccess public read
```

Add a create (upload) operation of the Science Data Object to the write operation queue:

```
> create mynewpid ~/path/to/my/file
```

View the queue:

```
> queue
```

Edit the queue if there are any mistakes in the create operation:


```
> edit
```

Perform all operations in the queue:

```
> run
```

Store the settings in `.dataone_cli.conf` for use when creating similar Science Data Objects later:

```
> save
```

Exit the CLI:

```
> exit
```

Misc operations

Find replicas of Science Data Objects:

```
> resolve hdl:10255/dryad.669/mets.xml
```

Display list of Science Data Objects on a *MN* or *CN*:

```
> set mn-url https://mn.dataone.org/mn
> set start 100
> set count 10
> list
```

Display event log on a *MN*:

```
> reset
> set anonymous false
> set cert-file /etc/dataone/dl_client/certs/myclientcert.pem
> set key-file None
> set mn-url https://dataone.org/mn
> log
```

Download the event log and save it to a file:

```
> log events.xml
```

4.3.3 Reference

Overview of operation

The DataONE Command Line Interface enables basic interactions with the DataONE infrastructure via the command line.

Session

The CLI is built around the concept of session variables. Session variables are analogous to environment variables in an operating system. The session variables store information that is often reused by multiple commands and minimize the amount of typing that is required. For instance, one session variable is the Base URL to a Member Node. Whenever a command is typed that will access a Member Node, the URL for the Member Node is pulled from the session.

The session can be saved and loaded from files. This allows easy switching between different roles by the user. For instance, if a user often works with two different Member Nodes and creates different types of objects on them, he can save the session after setting it up for each role, after which he can easily switch between them by loading the appropriate session.

Read vs. write operations

The commands that cause operations to be issued against Coordinating Nodes and Member Nodes are divided into two broad categories; read commands and write commands. These two categories are handled differently by the CLI. Read operations are issued immediately and their results displayed on screen or saved to a file. Write operations, however, are added to a queue, called the write operation queue, to be issued later.

Write operation queue

The DataONE infrastructure does not allow science objects to be modified or deleted once created. Objects can be updated, but the original object stays in the system forever. The write operation queue allows operations to be viewed and edited, thus adding a buffer where mistakes, such as typos, can be caught before the permanent operations are issued.

Like read commands, write commands use session variables. Each time an operation is added to the write operation queue, the relevant session variables are used for creating parameters for the operation. When an operation is later issued, it uses the parameters stored in the operation, not the current session variables.

When the commands have been verified, the queue is issued with a single command, after which each of the operations in the queue are automatically performed in sequence. If any operation fails, the process is stopped. The failed operation and all subsequent operations remain in the queue and can be manipulated before the queue is issued again. The successfully performed operations cannot be undone.

Access Policy

The Access Policy is a list of subjects and their associated access levels. The Access Policy is applied to new objects as they are created. The Access Policy can also be updated on existing Science Data Objects.

Replication Policy

The Replication Policy contains a list of Member Nodes that are specifically preferred or blocked as replication targets. The Replication Policy also contains a setting that specifies if replication is allowed or disallowed, and the preferred number of replicas.

The Replication Policy is applied to new objects as they are created. The Replication Policy can also be updated on existing Science Data Objects.

The DataONE infrastructure will never replicate an object to a Member Node that is in the list of blocked Member Nodes for that object. If a Member Node is added to the list of blocked Member Nodes after an object has been replicated to that Member Node, the DataONE infrastructure will request that the Member Node in question remove its copy of the object.

If the preferred number of replicas is larger than the number of Member Nodes that have been specified as preferred replication targets, additional Member Nodes that are not in the blocked list will automatically be selected as replication targets.

If the preferred number of replicas is modified on an existing Science Data Object, DataONE will adjust the number of existing replicas by creating and deleting replicas of that object as needed.

Authentication

A user that accesses a *Node* may connect either anonymously or as an authenticated subject. The Node to which the user connects will allow access to operations, Science Objects and other data based on the permissions that have been granted to the subject for which the user has authenticated.

A user that connects anonymously is granted access only to publicly available operations and data. Access is typically denied for operations that create or update data, such as the *create* operation.

When the CLI connects to a Node on a user's behalf, it passes authentication information for that user via a *certificate*. The certificate enables the user to act as a specific subject within a Node.

The user obtains a certificate for the subject with which to access a Node from *CILogon*. When the user downloads a certificate from CILogon, the CILogon download process stores the certificate in a standard location. The CLI can automatically find certificates in this location. In some cases, certificates may be stored in custom locations. In such cases, the automatic location of certificates can be bypassed by setting the `cert-path` session parameter to the filesystem path of the certificate. Because CILogon provides a certificate that holds both the public and private keys in the same file, only `cert-path` is required and `key-path` should be set to `None`. If the certificate was obtained in some other way, and the certificate's private key is stored in a separate file, the `key-path` session parameter must be set to the filesystem path of the private key.

When a user types a command that requires the CLI to connect to a Node, the CLI starts by examining the value of the `anonymous` session parameter. If the `anonymous` session parameter is **True**, the CLI ignores any available certificate and connects to the DataONE Node without providing a certificate. This causes the Node to allow access only to publicly available operations and data.

If the `anonymous` session parameter is **False**, the CLI attempts to locate the user's certificate as described above. If a certificate is not found, the operation is aborted. If a certificate is found, the CLI passes the certificate to the Node when establishing the connection. The Node validates the certificate and may reject it, causing the operation to be aborted. If the certificate is successfully validated, the Node grants access to the user, authenticated as the subject designated in the certificate, and the CLI proceeds with the operation.

Startup

When the CLI is started, it attempts to load the *Session* variables from a default configuration file named `.dataone_cli.conf`, located in the user's home directory. If the configuration file is not present, the session variables are set to default values as shown in the *Default* column in the *overview of session variables*.

The CLI then applies any options and executes any commands specified on the *command line*, in the specified order. This includes any *set* commands that modify the session variables.

Command line arguments

The CLI accepts a set of options and arguments on the command line. The options are used for setting the session variables. The arguments are executed as CLI commands. By default, the CLI will enter interactive mode after modifying the session according to the options and executing any commands provided as arguments. This can be prevented by passing the `--no-interactive` option or giving the `exit` command as the last argument. When the CLI enters interactive mode, the session that was set up with command line options remains active.

The command line arguments can also include commands that alter the session. E.g., the following examples are equivalent. Each will load the session from the user's `~/.dataone_cli.conf` file, download the `mypid` object from `mymembernode`, store it in `myfile` and exit.

```
$ dataone --no-interactive --mn-url http://mymembernode.org/mn 'get mypid myfile'
```

```
$ dataone --no-interactive 'set mn-url http://mymembernode.org/mn' 'get mypid myfile'
```

```
$ dataone 'set mn-url http://mymembernode.org/mn' 'get mypid myfile' exit
```

Commands that contain spaces or other symbols that have special meaning to the shell must be quoted. The examples use single quotes. Double quotes can also be used if it's desired to have the shell expand variables.

Since any CLI command is accepted on the command line, sessions can also be loaded with the *load [file]* command. If the CLI is called from a script, it may be desirable to start with a known, default session. This can be accomplished by issuing the *reset* command before any other commands.

When the session variables are set with the options, they are all applied before any of the commands specified as arguments are executed. When the session variables are specified with arguments, such as *set [variable [value]]*, they become active when they are specified and only apply to arguments specified later on the command line.

Also see *Overview of command line options*.

Commands

Table of Contents

- *Commands*
 - *Syntax*
 - *CLI*
 - * *history*
 - * *exit*
 - * *exit*
 - *Session, General*
 - * *set [variable [value]]*
 - * *load [file]*
 - * *save [config_file]*
 - * *reset*
 - *Session, Access Control*
 - * *allowaccess <subject> [access level]*
 - * *denyaccess <subject>*
 - * *clearaccess*
 - *Session, Replication Policy*
 - * *allowrep*
 - * *denyrep*
 - * *preferrep <member node> [member node ...]*
 - * *blockrep <member node> [member node ...]*
 - * *removerep <member node> [member node ...]*

- * *numberrep* <number of replicas>
- * *clearrep*
- *Read Operations*
 - * *get* <identifier> <file>
 - * *meta* <identifier> [file]
 - * *list*
 - * *log*
 - * *resolve* <pid>
- *Write Operations*
 - * *create* <pid> <file>
 - * *update* <old-pid> <new-pid> <file>
 - * *package* <package-pid> <science-metadata-pid> <science-pid> [science-pid ...]
 - * *archive* <identifier> [identifier ...]
 - * *updateaccess* <identifier> [identifier ...]
 - * *updatereplication* <identifier> [identifier ...]
- *Utilities*
 - * *listformats*
 - * *listnodes*
 - * *ping* [base-url ...]
- *Write Operation Queue*
 - * *queue*
 - * *run*
 - * *edit*
 - * *clearqueue*

Syntax

< . . . > denotes required arguments.

[. . .] denotes optional arguments.

file is the filesystem path to a local file.

Commands are case sensitive.

CLI

Commands that relate to the operation of the Command Line Interface itself.

help — Get help on commands

help or ? with no arguments displays a list of commands for which help is available

`help <command>` or `? <command>` gives help on `<command>`

history

Display a list of commands that have been entered

exit

Exit from the CLI

exit

Exit from the CLI

Session, General

Commands that view and manipulate the session and the session variables.

set [variable [value]]

`set` (without parameters): Display the value of all session variables

`set <session variable>`: Display the value of a single session variable.

`set <session variable> <value>`: Set the value of a session variable.

See [Access Policy](#) and [Replication Policy](#) for information about how to set the Access Policy and Replication Policy session variables.

An unset session variable has its value displayed as `None`. A session variable can either be a Boolean (`True` / `False`), numeric or string value. See [set \[variable \[value\]\]](#) for more information on how to set session variables.

Also see [overview of session variables](#).

load [file]

Load session variables from file

`load` (without parameters): Load session from default file `~/.dataone_cli.conf`

`load <file>`: Load session from specified file

save [config_file]

Save session variables to file

`save` (without parameters): Save session to default file `~/.dataone_cli.conf`

`save <file>`: Save session to specified file

reset

Set all session variables to their default values

The defaults are listed in the `Default` column in the [overview of session variables](#).

Session, Access Control

The Access Policy is a list of subjects and their associated access levels. The Access Policy is applied to new objects as they are *created*. The Access Policy can also be updated on existing Science Data Objects with *updateaccess* `<identifier> [identifier ...]`.

Use the *set* `[variable [value]]` command without any parameters to view the current Access Policy.

allowaccess <subject> [access level]

Set the access level for subject

Access level is `read`, `write` or `changePermission`.

Access level defaults to `read` if not specified.

Special subjects:

`public`: Any subject, authenticated and not authenticated

`authenticatedUser`: Any subject that has authenticated with CILogon

`verifiedUser`: Any subject that has authenticated with CILogon and has been verified by DataONE

Any access level implicitly includes less permissive levels. E.g., giving `changePermission` to a subject implicitly gives `read` and `write` permissions as well.

To make objects accessible to the general public, give `read` access to the public user. In some cases, it is desirable to obtain log records that include information about who accessed a given object while still making the object publicly accessible. This can be accomplished by giving `read` access only to `authenticatedUser`. Access higher than `read` should not be given to any of the special subjects.

denyaccess <subject>

Remove subject from Access Policy.

clearaccess

Remove all subjects from Access Policy.

Only the submitter will have access to the object.

Session, Replication Policy

allowrep

Allow new objects to be replicated

denyrep

Prevent new objects from being replicated

preferrep <member node> [member node ...]

Add one or more preferred Member Nodes to replication policy

blockrep <member node> [member node ...]

Add Member Node to list of blocked replication targets.

removerep <member node> [member node ...]

Remove Member Node from list of preferred or blocked replication targets.

numberrep <number of replicas>

Set preferred number of replicas for new objects

If the preferred number of replicas is set to zero, replication is also disallowed.

clearrep

Set the replication policy to default

The default replication policy has no preferred or blocked member nodes, allows replication and sets the preferred number of replicas to 3.

Read Operations

Commands that cause read operations to be issued against Coordinating Nodes and Member Nodes.

Commands for retrieving *Science Data Objects*, *System Metadata* and related information.

get <identifier> <file>

Get an object from a Member Node

The object is saved to `file`.

Active session variables: *mn-url*, *Authentication*

meta <identifier> [file]

Get the System Metadata that is associated with a Science Object

If the metadata is not on the Coordinating Node, the Member Node is checked.

Provide `file` to save the System Metadata to disk instead of displaying it.

Active session variables: *cn-url, Authentication*

list

Retrieve a list of available Science Data Objects from Member Node

The response is filtered by the from-date, to-date, search, start and count session variables.

Active session variables: *mn-url, start, count, from-date, to-date, search-format-id, Authentication*

See also: `search`

log

Retrieve event log from Member Node

The response is filtered by the from-date, to-date, start and count session parameters.

Active session variables: *mn-url, start, count, from-date, to-date, search-format-id, Authentication*

resolve <pid>

Find all locations from which the given Science Object can be downloaded.

Active session variables: *cn-url, Authentication*

Write Operations

Commands that cause write operations to be issued against Coordinating Nodes and Member Nodes.

create <pid> <file>

Create a new Science Object on a Member Node.

The System Metadata that becomes associated with the new Science Object is generated from the session variables.

The algorithm set in *algorithm* is used for calculating the checksum for the new object. If the value is unset, it defaults to the DataONE system wide default, which is currently SHA1.

Active session variables: *mn-url, format-id, submitter, rights-holder, origin-mn, authoritative-mn, algorithm, Access Policy, Replication Policy, Authentication*

update <old-pid> <new-pid> <file>

Replace an existing Science Object in a *MN* with another.

The existing Science Object becomes obsoleted by the new Science Object. obsoleted by the new values in the *System Metadata*, *Access Policy* and *Replication Policy* session variables.

The algorithm set in *algorithm* is used for calculating the checksum for the new object. If the value is unset, it defaults to the DataONE system wide default, which is currently SHA1.

Active session variables: *mn-url*, *format-id*, *submitter*, *rights-holder*, *origin-mn*, *authoritative-mn*, *algorithm*, *Access Policy*, *Replication Policy*, *Authentication*

package <package-pid> <science-metadata-pid> <science-pid> [science-pid ...]

Create a simple *OAI-ORE Resource Map* on a Member Node

archive <identifier> [identifier ...]

Mark one or more existing Science Objects as archived

updateaccess <identifier> [identifier ...]

Update the Access Policy on one or more existing Science Data Objects

Requires that the calling subject has *authenticated* and has changePermission access level on the object for which Access Policy is to be updated.

Active session variables: *cn-url*, *Authentication*, *Access Policy*

updatereplication <identifier> [identifier ...]

Update the Replication Policy on one or more existing Science Data Objects

Requires that the calling subject has *authenticated* and has changePermission access level on the object for which Replication Policy is to be updated.

Active session variables: *cn-url*, *Replication Policy*, *Authentication*

Utilities

listformats

Display all known Object Format IDs

listnodes

Display all known DataONE Nodes

search [query] Comprehensive search for Science Data Objects across all available MNs

See <https://releases.dataone.org/online/api-documentation-v2.0.1/design/SearchMetadata.html> for the available search terms.

ping [base-url ...]

Check if a server responds to the DataONE ping() API method ping (no arguments): Ping the CN and MN that is specified in the session ping <base-url> [base-url ...]: Ping each CN or MN

If an incomplete base-url is provided, default CN and MN base URLs at the given url are pinged.

Write Operation Queue

Commands that view and manipulate the write operation queue.

queue

Print the queue of write operations.

run

Perform each operation in the queue of write operations

edit

Edit the queue of write operations

clearqueue

Remove the operations in the queue of write operations without performing them

Overview of session variables

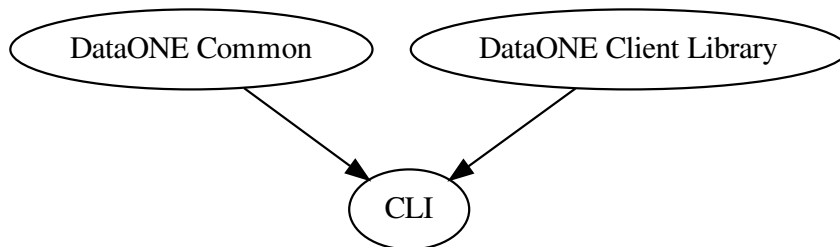
Name	Default	Type	Description
CLI configuration			
verbose	False	Boolean	Display more information editor nano String I
Target Nodes			
cn-url	https://cn.dataone.org/cn	String	Node to which to connect for operations that a
mn-url	https://localhost/mn/	String	Node to which to connect for operations that a
Authentication			
anonymous	True	Boolean	Ignore any installed certificates and connect a
key-file	None	String	Filesystem path to the client certificate private
Slicing			
start	0	Integer	First item to display for operations that display
count	1000	Integer	Maximum number of items to display for oper
Searching			
query	*.*	String	Query string (SOLR or Lucene query syntax)

Table 1 – continued from previous

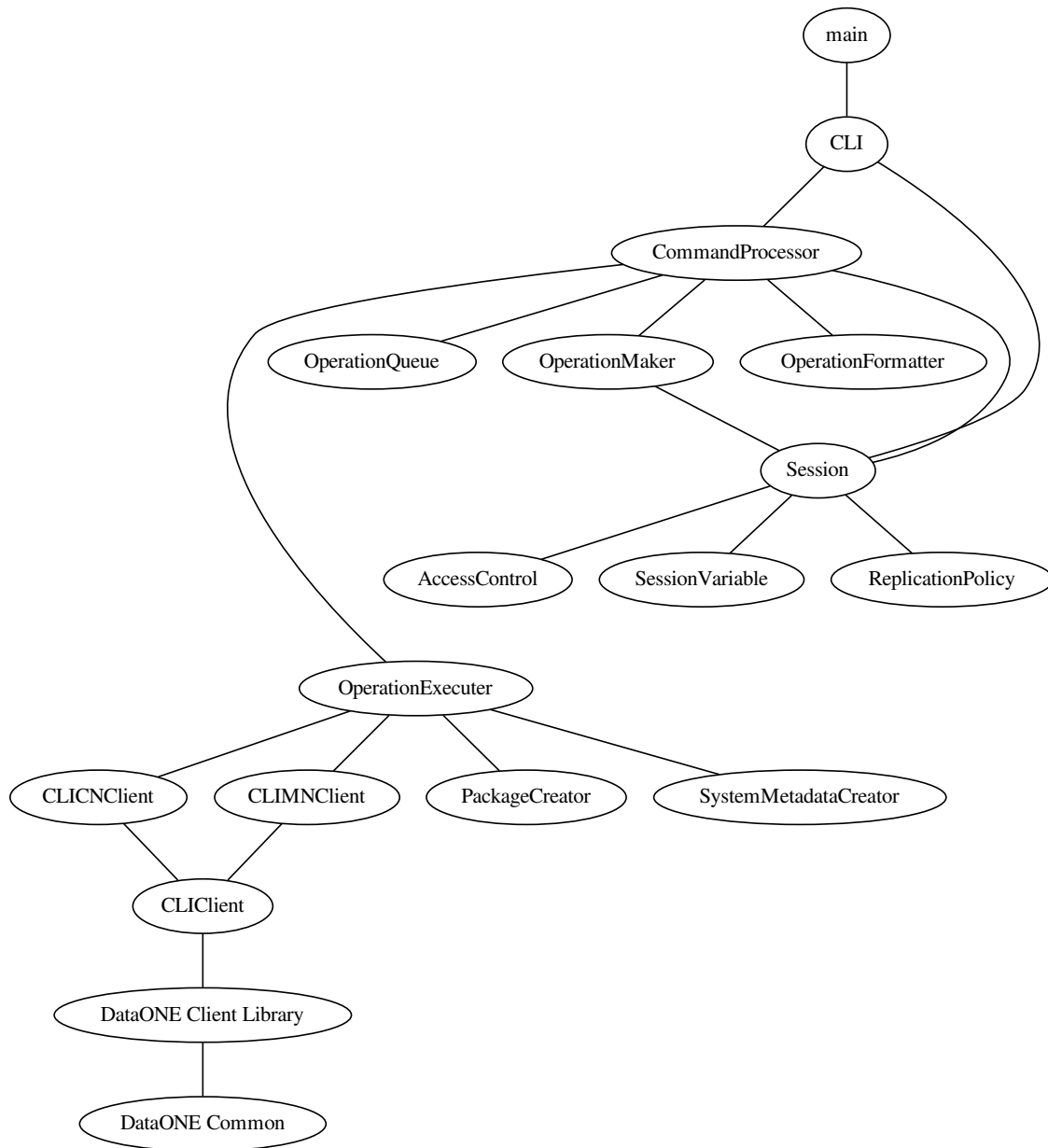
Name	Default	Type	Description
query-type	solr	String	Select search engine (currently, only SOLR is supported)
from-date	None	String	Start time used by operations that accept a time range
to-date	None	String	End time used by operations that accept a time range
search-format-id	None	String	Include only objects of this format
Parameters Misc			
algorithm	SHA-1	String	Checksum algorithm to use when calculating object checksums
format-id	None	String	ID for the Object Format to use when generating object IDs
Parameters Reference Nodes			
authoritative-mn	None	String	Authoritative Member Node to use when generating object IDs
origin-mn	None	String	Originating Member Node to use when generating object IDs
Parameters Subjects			
rights-holder	None	String	Subject of the rights holder to use when generating object IDs
submitter	None	String	Subject of the submitter to use when generating object IDs
Access Control			
Access Control Policy parameters managed by a <i>separate set of commands</i> .			
Replication			
Replication Policy parameters managed by a <i>separate set of commands</i> .			

4.3.4 Implementation

Dependencies



Class hierarchy



Command An action that causes changes only internal to the CLI.

Operation An action that causes one or more reads or writes against a DataONE Node.

main:

- Handle command line options.
- Capture and display internal and external exceptions.

CLI:

- Generic boiler plate for Python CLI apps.
- Simple command tokenizing and validation.

CommandProcessor:

- Manipulate the session.
- Create, then execute DataONE Read Operations.
- Create, then queue DataONE Write Operations.
- Execute queue of DataONE Write Operations.
- Display the results of DataONE Operations.

OperationMaker:

- Combine parameters from CommandProcessor and from the session into a DataONE Read or Write Operation.

OperationQueue:

- Hold a queue of DataONE Write Operations.
- Edit the queue.
- Display the queue.

OperationExecuter:

- Execute a DataONE Read Operation or a queue of Write Operations.

Utility classes

These are used throughout the main classes and so are kept out of main hierarchy for readability.



Notes

- Read operations are executed immediately.
- Write operations are queued and executed in a batch. The write queue can be edited.
- Write operations are decoupled from the session. Each write operation contains a copy of the relevant session variables at the time the operation was issued. Those variables are then used when the operation is executed.

4.3.5 Overview of command line options

```
Usage: dataone.py [command] ...
```

```
Options:  
  --algorithm=ALGORITHM
```

(continues on next page)

(continued from previous page)

```

Checksum algorithm used for a Science Data Object.
--anonymous          Ignore any installed certificates and connect
                      anonymously
--no-anonymous        Use the installed certificates and do not connect
                      anonymously
--authoritative-mn=MN-URI
                      Authoritative Member Node for generating System
                      Metadata.
--cert-file=FILE      Path to client certificate
--count=COUNT        Maximum number of items to display
--cn-url=URI          URI to use for the Coordinating Node
--from-date=DATE       Start time used by operations that accept a date range
--key-file=FILE       File of client private key (not required if key is in
                      cert-file)
--mn-url=URI          Member Node URL
--format-id=OBJECT-FORMAT
                      ID for the Object Format to use when generating System
                      Metadata
--formatId=OBJECT-FORMAT
                      ID for the Object Format to use when generating System
                      Metadata
--origin-mn=MN-URI    Originating Member Node to use when generating System
                      Metadata
--query=QUERY         Query string (SOLR or Lucene query syntax) for
                      searches
--rights-holder=SUBJECT
                      Subject of the rights holder to use when generating
                      System Metadata
--search-format-id=OBJECT-FORMAT
                      Include only objects of this format when searching
--start=START         First item to display for operations that display a
                      list_objects of items
--submitter=SUBJECT   Subject of the submitter to use when generating System
                      Metadata
--to-date=DATE        End time used by operations that accept a date range
-v, --verbose         Display more information
--no-verbose          Display less information
--editor             Editor to use for editing operation queue
--no-editor           Use editor specified in EDITOR environment variable
--allow-replication   Allow objects to be replicated.
--disallow-replication
                      Do not allow objects to be replicated.
--replicas=#replicas Set the preferred number of replicas.
--add_blocked=MN      Add blocked Member Node to access policy.
--add_preferred=MN    Add Member Node to list_objects of preferred
                      replication targets.
--cn=HOST             Name of the host to use for the Coordinating Node
--mn=HOST             Name of the host to use for the Member Node
-i, --interactive     Allow interactive commands
--no-interactive      Don't allow interactive commands
-q, --quiet           Display less information
--debug              Print full stack trace and exit on errors
-h, --help            show this help message and exit

```

4.3.6 API

d1_cli package

DataONE Command-line Client.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_cli.impl package

DataONE Command-line Client.

Submodules

d1_cli.impl.access_control module

Create and manipulate access control objects.

```
class d1_cli.impl.access_control.AccessControl
    Bases: object

    get_list ()

    add_allowed_subject (subject, permission)

    remove_allowed_subject (subject)

    clear ()
```

d1_cli.impl.check_dependencies module

Check the dependencies by attempting to import them.

```
d1_cli.impl.check_dependencies.are_modules_importable (module_list=None)
```

d1_cli.impl.client module

CN and MN clients of the DataONE Command Line Interface.

The logic was factored out of these and reimplemented in more natural locations. This is just a placeholder for now. May remove.

```
class d1_cli.impl.client.CLIClient
    Bases: object

class d1_cli.impl.client.CLIMNClient (*args, **kwargs)
    Bases: d1_cli.impl.client.CLIClient, d1_client.mnclient_2_0.MemberNodeClient_2_0

class d1_cli.impl.client.CLICNClient (*args, **kwargs)
    Bases: d1_cli.impl.client.CLIClient, d1_client.cnclient_2_0.CoordinatingNodeClient_2_0
```



```
class d1_cli.impl.client.CLIBaseClient(*args,**kwargs)
    Bases: d1_client.baseclient_2_0.DataONEBaseClient_2_0
```

d1_cli.impl.command_parser module

- Generic boiler plate for Python CLI apps.
- Simple command tokenizing and validation.

```
class d1_cli.impl.command_parser.CLI
    Bases: cmd.Cmd
```

preloop()

Initialization before prompting user for commands.

Despite the claims in the Cmd documentation, Cmd.preloop() is not a stub.

postloop()

Take care of any unfinished business.

Despite the claims in the Cmd documentation, Cmd.postloop() is not a stub.

precmd(line)

This method is called after the line has been input but before it has been interpreted.

If you want to modify the input line before execution (for example, variable substitution) do it here.

postcmd(stop, line)

If you want to stop the console, return something that evaluates to true.

If you want to do some post command processing, do it here.

emptyline()

Do nothing on empty input line.

default(line)

Called on an input line when the command prefix is not recognized.

run_command_line_arguments(cmd_line_list)

do_help(line)

Get help on commands “help” or “?” with no arguments displays a list_objects of commands for which help is available “help <command>” or “?”

<command>” gives help on <command>

do_history(line)

history Display a list of commands that have been entered.

do_exit(line)

exit Exit from the CLI.

do_quit(line)

quit Exit from the CLI.

do_eof(line)

Exit on system EOF character.

do_set(line)

set [parameter [value]] set (without parameters): Display the value of all session variables.

set <session variable>: Display the value of a single session variable. set <session variable> <value>: Set the value of a session variable.

do_load (*line*)
load [file] Load session variables from file load (without parameters): Load session from default file
~/.dataone_cli.conf load.

<file>: Load session from specified file.

do_save (*line*)
save [config_file] Save session variables to file save (without parameters): Save session to default file
~/.dataone_cli.conf save.

<file>: Save session to specified file.

do_reset (*line*)
reset Set all session variables to their default values.

do_allowaccess (*line*)
allowaccess <subject> [access-level] Set the access level for subject Access level is “read”, “write” or
“changePermission”.

Access level defaults to “read” if not specified. Special subjects: public: Any subject, authenticated and
not authenticated authenticatedUser: Any subject that has authenticated with CILogon verifiedUser: Any
subject that has authenticated with CILogon and has been verified by DataONE

do_denyaccess (*line*)
denyaccess <subject> Remove subject from access policy.

do_clearaccess (*line*)
clearaccess Remove all subjects from access policy Only the submitter will have access to the object.

do_allowrep (*line*)
allowrep Allow new objects to be replicated.

do_denyrep (*line*)
denyrep Prevent new objects from being replicated.

do_preferrep (*line*)
preferrep <member node> [member node ...] Add one or more preferred Member Nodes to replication
policy.

do_blockrep (*line*)
blockrep <member node> [member node ...] Add one or more blocked Member Node to replication
policy.

do_removerep (*line*)
removerep <member node> [member node ...] Remove one or more Member Nodes from replication
policy.

do_numberrep (*line*)
numberrep <number of replicas> Set preferred number of replicas for new objects If the preferred number
of replicas is set to zero, replication is also disallowed.

do_clearrep (*line*)
clearrep Set the replication policy to default.

The default replication policy has no preferred or blocked member nodes, allows replication and sets the
preferred number of replicas to 3.

do_get (*line*)
get <identifier> <file> Get an object from a Member Node.

The object is saved to <file>.

do_meta (*line*)

meta <identifier> [file] Get the System Metadata that is associated with a Science Object.

If the metadata is not on the Coordinating Node, the Member Node is checked.

Provide `file` to save the System Metadata to disk instead of displaying it.

do_list (*line*)

list [path] Retrieve a list of available Science Data Objects from Member Node The response is filtered by the from-date, to-date, search, start and count session variables.

See also: `search`

do_log (*line*)

log [path] Retrieve event log from Member Node The response is filtered by the from-date, to-date, start and count session parameters.

do_resolve (*line*)

resolve <identifier> Find all locations from which the given Science Object can be downloaded.

do_create (*line*)

create <identifier> <file> Create a new Science Object on a Member Node.

The System Metadata that becomes associated with the new Science Object is generated from the session variables.

do_update (*line*)

update <old-pid> <new-pid> <file> Replace an existing Science Object in a.

MN with another.

do_package (*line*)

package <package-pid> <science-metadata-pid> <science-pid> [science- pid.

...] Create a simple OAI-ORE Resource Map on a Member Node.

do_archive (*line*)

archive <identifier> [identifier ...] Mark one or more existing Science Objects as archived.

do_updateaccess (*line*)

updateaccess <identifier> [identifier ...] Update the Access Policy on one or more existing Science Data Objects.

do_updatereplication (*line*)

updatereplication <identifier> [identifier ...] Update the Replication Policy on one or more existing Science Data Objects.

do_listformats (*line*)

listformats Display all known Object Format IDs.

do_listnodes (*line*)

listnodes Display all known DataONE Nodes.

do_search (*line*)

search [query] Comprehensive search for Science Data Objects across all available MNs.

See <https://releases.dataone.org/online/api-documentation-v2.0.1/design/SearchMetadata.html> for the available search terms.

do_ping (*line*)

ping [base-url ...] Check if a server responds to the DataONE ping() API method ping (no arguments): Ping the CN and MN that is specified in the session ping <base-url> [base-url ...]: Ping each CN or MN.

If an incomplete base-url is provided, default CN and MN base URLs at the given url are pinged.

do_queue (*line*)
queue Print the queue of write operations.

do_run (*line*)
run Perform each operation in the queue of write operations.

do_edit (*line*)
edit Edit the queue of write operations.

do_clearqueue (*line*)
clearqueue Remove the operations in the queue of write operations without performing them.

d1_cli.impl.command_processor module

Process and execute CLI operations.

```
class d1_cli.impl.command_processor.CommandProcessor
    Bases: object

    get_session ()

    get_operation_queue ()

    get_nodes ()

    get_format_ids ()

    ping (hosts)

    search (line)
        CN search.

    list_format_ids ()

    list_nodes ()

    resolve (pid)
        Get Object Locations for Object.

    update_access_policy (pids)

    update_replication_policy (pids)

    science_object_get (pid, path)
        First try the MN set in the session.

        Then try to resolve via the CN set in the session.

    system_metadata_get (pid, path)

    log (path)

    list_objects (path)

    science_object_create (pid, path, format_id=None)
        Create a new Science Object on a Member Node.

    science_object_update (pid_old, path, pid_new, format_id=None)
        Obsolete a Science Object on a Member Node with a different one.

    create_package (pids)

    science_object_archive (pids)
```

d1_cli.impl.const module

Constants (mostly names of things).

```
class d1_cli.impl.const.SessionVariable (sect, name)  
    Bases: object
```

d1_cli.impl.exceptions module

Classes used for passing exceptions within and between components of the DataONE Command Line Client.

```
exception d1_cli.impl.exceptions.InvalidArguments (value)  
    Bases: Exception
```

```
exception d1_cli.impl.exceptions.CLIErrror (value)  
    Bases: Exception
```

d1_cli.impl.format_ids module

Retrieve, cache, manipulate list of known Object Format IDs.

```
class d1_cli.impl.format_ids.FormatIDs  
    Bases: object  
  
    get (cn_base_url)  
  
    format (cn_base_url)
```

d1_cli.impl.nodes module

Retrieve, cache, manipulate list of known DataONE nodes.

```
class d1_cli.impl.nodes.Nodes  
    Bases: object  
  
    get (cn_base_url)  
  
    format (cn_base_url)
```

d1_cli.impl.operation_executer module

Execute queued write operations.

```
class d1_cli.impl.operation_executer.OperationExecuter  
    Bases: object  
  
    execute (operation)
```

d1_cli.impl.operation_formatter module

Pretty print an operation.

```
class d1_cli.impl.operation_formatter.OperationFormatter
```

```
    Bases: object
```

```
    Print an operation according to the template.
```

```
    The template contains all parameters that can be in any of the operations and determines the relative position of each parameter that is present in the operation.
```

```
    print_operation (operation)
```

d1_cli.impl.operation_maker module

Put together all the information required for executing a given operation.

```
class d1_cli.impl.operation_maker.OperationMaker (session)
```

```
    Bases: object
```

```
    create (pid, path, format_id=None)
```

```
    update (pid, path, pid_new, format_id=None)
```

```
    create_package (pids)
```

```
    archive (pid)
```

```
    update_access_policy (pid)
```

```
    update_replication_policy (pid)
```

d1_cli.impl.operation_queue module

Hold a queue of operations and perform commands on the queue.

```
class d1_cli.impl.operation_queue.OperationQueue (session)
```

```
    Bases: object
```

```
    append (operation)
```

```
    display ()
```

```
    edit ()
```

```
    execute ()
```

```
    clear ()
```

d1_cli.impl.operation_validator module

Sanity checking of the values which are required by a given operation.

```
class d1_cli.impl.operation_validator.OperationValidator
```

```
    Bases: object
```

```
    assert_valid (operation)
```

```
    assert_valid_create (operation)
```

```
    assert_valid_update (operation)
```

```
    assert_valid_create_package (operation)
```

```
    assert_valid_archive (operation)
```

```

assert_valid_update_access_policy (operation)
assert_valid_update_replication_policy (operation)

```

d1_cli.impl.replication_policy module

Create and manipulate replication policies.

```

class d1_cli.impl.replication_policy.ReplicationPolicy
    Bases: object

    clear ()

    get_preferred ()

    get_blocked ()

    add_preferred (mns)

    remove (mns)

    add_blocked (mns)

    set_replication_allowed (replication_allowed)

    get_replication_allowed ()

    get_number_of_replicas ()

    set_number_of_replicas (number_of_replicas)

    print_replication_policy ()

```

d1_cli.impl.session module

Hold and manipulate session variables.

```

class d1_cli.impl.session.Session (nodes, format_ids)
    Bases: object

    reset ()

    get (variable)

    get_access_control ()

    get_replication_policy ()

    set (variable, value)

    set_with_conversion (variable, value_string)
        Convert user supplied string to Python type.

        Lets user use values such as True, False and integers. All variables can be set to None, regardless of type.
        Handle the case where a string is typed by the user and is not quoted, as a string literal.

    print_variable (variable)

    print_single_variable (variable)

    print_all_variables ()

    load (pickle_file_path=None, suppress_error=False)

    save (pickle_file_path=None, suppress_error=False)

```

```
is_verbose()  
get_default_pickle_file_path()
```

d1_cli.impl.system_metadata module

Create System Metadata documents based on session variables.

```
class d1_cli.impl.system_metadata.SystemMetadataCreator  
    Bases: object  
  
    create_system_metadata(operation)  
    create_system_metadata_for_update(operation)  
    create_system_metadata_for_package(resource_map, create_package_operation)  
    create_access_policy(operation)  
    create_replication_policy(operation)
```

d1_cli.impl.util module

Utilities shared between components of the DataONE Command Line.

```
d1_cli.impl.util.confirm(prompt, default='no', allow_blank=False)  
d1_cli.impl.util.output(file_like_object, path, verbose=False)  
    Display or save file like object.  
d1_cli.impl.util.assert_file_exists(path)  
d1_cli.impl.util.copy_file_like_object_to_file(file_like_object, path)  
d1_cli.impl.util.copy_requests_stream_to_file(response, path)  
d1_cli.impl.util.print_debug(msg)  
d1_cli.impl.util.print_error(msg)  
d1_cli.impl.util.print_warn(msg)  
d1_cli.impl.util.print_info(msg)
```

d1_cli.tests package

Submodules

d1_cli.tests.test_access_control module

d1_cli.tests.test_cli module

d1_cli.tests.test_cli_exceptions module

d1_cli.tests.test_cli_util module

d1_cli.tests.test_command_processor module

d1_cli.tests.test_replication_policy module

d1_cli.tests.test_session module

Submodules

d1_cli.dataone module

DataONE Command Line Interface.

`d1_cli.dataone.main()`

`d1_cli.dataone.log_setup(debug)`

`d1_cli.dataone.handle_options(cli, options)`

`d1_cli.dataone.handle_unexpected_exception(max_traceback_levels=100)`
Suppress stack traces for common errors and provide hints for how to resolve them.

d1_cli.pids2ore module

d1_cli.version module

4.3.7 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.4 Generic Member Node (GMN)

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

The *Generic Member Node (GMN)* is a DataONE Member Node *MN*). It provides an implementation of MN APIs and can be used by organizations to expose their science data to DataONE if they do not wish to create their own, native MN.

GMN can be used as a standalone MN or it can be used for exposing data that is already available on the web, to DataONE. When used in this way, GMN provides a DataONE compatible interface to existing data and does not store the data.

GMN can also be used as a workbone or reference for a 3rd party MN implementation. If an organization wishes to donate storage space to DataONE, GMN can be set up as a replication target.

Contents:

4.4.1 GMN setup overview

Setting up the DataONE Generic Member Node (*GMN*).

Verified setup procedures are provided for Ubuntu 16.04 LTS (Server and Desktop) and CentOS 7.3.

It may be possible to deploy GMN using a different stack, such as one based on [nginx](#) and [uWSGI](#). Such setups are currently untested, but if they are attempted and prove to have benefits, please let us know.

The GMN setup process has been broken down into two sections, each containing a series of steps. The first section describes how to set up an instance of GMN which can be used only locally. The second section describes how to join the GMN instance to DataONE. For testing GMN and learning about Member Nodes, only the first section need be completed. For exposing data to the DataONE federation and providing storage for replicas, both the first and second sections must be completed.

Along with the steps in each section, some background information is provided. The actual steps that need to be performed are indented to separate them from the background information.

Commands that need to be run from the shell are prefixed with “\$”.

The instructions describe an installation into subfolders of `/var/local/dataone/`. To install into another location, all related paths must be adjusted accordingly.

The instructions describe how to set GMN up to run in a separate Apache Virtual Host on a fresh install of Ubuntu. General setup and configuration issues, such as issues that may be encountered when adding GMN to an existing server, are not covered.

The GMN software stack is installed into a Python virtual environment to avoid potential conflicts with other Python software on the server.

Use the [Next](#) link in the sidebar to get to the next page of steps after completing the current page.

Contents:

Hardware requirements and configuration

Setting up the hardware.

GMN is installed on a physical or virtual machine. Network connectivity is arranged so that GMN can be reached from the DataONE *CNs* and from *clients*. Typically, this means that GMN is set up to be globally accessible from the web.

GMN can be used in a mode where it provides a DataONE interface to data that is already available on the web. When used in this way, GMN must also have access to the web site which holds the data.

The requirements for RAM, CPU, disk and network resources are workload dependent. Below is benchmarks for two different setups.

Benchmarks

To give an indication of the hardware that may be required for hosting GMN, some benchmarks are provided.

Configuration of benchmarking scripts:

- Concurrent calls per API: 5
- Science object size: 1024 bytes.
- Allow rules per object: 10
- listObjects / getLogRecords page size: 1000 objects

Hardware configuration 1

Machine type	Physical
CPU	Intel Core2 Quad Q6600 @ 2.40GHz
RAM	4GiB
Disk	5400 RPM SATA (I/O @ 60 MiB/s)

API	Transactions per second
MNStorage.create()	9.8
MNRead.get()	35.3
MNRead.listObjects()	0.5
MNCore.getLogRecords(), paged, called by CN	0.36
MNCore.getLogRecords(), specific object, called by regular subject	40.6
Combination of MNStorage.create(), MNRead.get(), MNRead.listObjects()	4.4
Combination of MNCore.getLogRecords(), MNRead.get()	36.2

Hardware configuration 2

Machine type	Virtual
CPU	Intel Xeon E7540 @ 2.00GHz
RAM	32GiB
Disk	NFS (I/O @ 45MiB/s)

API	Transactions per second
MNStorage.create()	9.3
MNRead.get()	5.6
MNRead.listObjects()	0.35
MNCore.getLogRecords(), paged, called by CN	0.2
MNCore.getLogRecords(), specific object, called by regular subject	6.0
Combination of MNStorage.create(), MNRead.get(), MNRead.listObjects()	2.8
Combination of MNCore.getLogRecords(), MNRead.get()	5.24

Setup on Ubuntu

This section describes the initial steps in setting up *GMN*. The procedure has been verified on Server and Desktop versions of Ubuntu 18.04 LTS.

Instructions for CentOS are *also available*.

Complete this section to set up a stand-alone test instance of GMN. The stand-alone instance can be used for performance testing, developing scripts for populating the MN and for learning about MNs in general.

The stand-alone test instance can then be joined to the DataONE production environment as an official Member Node or to one of DataONE's test environments for further testing by completing *Registering the new MN in a DataONE environment*.

Contents:

GMN Dependencies and Platform Installation

The following steps require sudo access. They prepare the server for the part of the GMN installation that can be performed as the `gmh` user.

Contents:

Update OS and install APT dependencies

Run the following commands to:

- Upgrade all installed packages to latest versions
- Install APT packaged GMN dependencies
- Set server clock to UTC timezone
- Open for HTTPS in the firewall

```
sudo -H bash -c '  
  apt update --yes  
  apt dist-upgrade --yes  
'
```

Reboot if necessary.

```
sudo -H bash -c '  
  apt install --yes build-essential libssl-dev libxml2-dev libxslt1-dev \  
  libffi-dev postgresql openssl curl python-pip python3-venv \  
  python3-dev apache2 libapache2-mod-wsgi-py3 acl  
  
  pip install --upgrade pip virtualenv  
  
  timedatectl set-timezone Etc/UTC  
  ufw allow 443  
'
```

Create `gmh` Account and Configure Permissions

Run the following commands to:

- Create the `gmh` user account (with password login disabled)
- Add or update permissions allowing the `gmh` user to
 - Create and edit Apache configuration files
 - Restart the Apache and Postgres services
 - Read Apache and Postgres logs
- Prepare the DataONE root directory
- Create Postgres role and database

Note: These commands can safely be run multiple times. Any missing permissions will be restored. Existing permissions will not be duplicated.

```

sudo -H bash -c '
# Create the gmn user account with password login disabled
id -u gmn 1>/dev/null 2>&1 || adduser --ingroup www-data \
    --gecos "DataONE Generic Member Node" --disabled-password gmn

ERR=$(sudo -u postgres createuser gmn 2>&1)
[[ ${ERR} =~ "already exists" ]] || echo ${ERR}
ERR=$(sudo -u postgres createdb -E UTF8 gmn3 2>&1)
[[ ${ERR} =~ "already exists" ]] || echo ${ERR}

mkdir -p /var/local/dataone
chown -R gmn:www-data /var/local/dataone
chmod -R 00755 /var/local/dataone

# Allow the gmn user to create and edit Apache configuration files
setfacl -Rm gmn:rwX /etc/apache2 /var/lib/apache2/site/enabled_by_admin/

# Allow the gmn user to start and stop the Apache and Postgres services
for s in postgresql apache2; do
    grep -q $s /etc/sudoers \
        || echo "gmn ALL=NOPASSWD:/etc/init.d/$s" >> /etc/sudoers
done

# Allow the gmn user to read existing Postgres and Apache logs
setfacl -Rm gmn:rx /var/log/postgresql /var/log/apache2

# Allow the gmn user to read future Postgres and Apache logs
P="/etc/logrotate.d/gmn"
echo >$P "postrotate"
echo >>$P "    setfacl -Rm gmn:rx /var/log/postgresql /var/log/apache2"
echo >>$P "endscript"
'

```

GMN Software Stack Installation

The following steps describe the part of the GMN installation that can be performed as the *gmn* user. It is assumed that *GMN Dependencies and Platform Installation* has already been performed.

Contents:

Install the GMN software stack

Run the following commands to:

- Install the GMN software stack from PyPI into a Python virtual environment
- Install standard `.bashrc` for the *gmn* user

```

sudo -Hu gmn bash -c '
python3 -m venv /var/local/dataone/gmn_venv_py3
. /var/local/dataone/gmn_venv_py3/bin/activate
GMN_PKG_DIR=$(python -c "import site; print(site.getsitepackages()[0])")
pip install --upgrade pip virtualenv
pip install dataone.gmn
cp ${GMN_PKG_DIR}/d1_gmn/deployment/bashrc ~/.bashrc
'

```

(continues on next page)

(continued from previous page)

```
chmod go+x ${GMN_PKG_DIR}/dl_gmn/manage.py
,
```

Install and configure Apache

Run the commands below to:

- Install default GMN configuration for Apache
- Set correct ServerName in GMN VirtualHost file

```
sudo -Hu gmn bash -c '
. /var/local/dataone/gmn_venv_py3/bin/activate
GMN_PKG_DIR=`python -c "import site; print(site.getsitepackages()[0])"`
FQDN=`python -c "import socket; print(socket.getfqdn())"`
CONF_PATH=/etc/apache2/sites-available/gmn3-ssl.conf
DELIMITER=`printf "%0s" {1..100}`

cp ${GMN_PKG_DIR}/dl_gmn/deployment/gmn3-ssl.conf ${CONF_PATH}

sed -Ei "s/www\.example\.com/${FQDN}/" ${CONF_PATH}

a2enmod wsgi ssl alias
a2dissite 000-default
a2ensite gmn3-ssl

printf "%s\nUsing FQDN: %s\nIf this is incorrect, correct it in %s\n%s\n" \
  ${DELIMITER} ${FQDN} ${CONF_PATH} ${DELIMITER}
,
```

Configure the GMN asynchronous processes

Run the commands below to:

- Set up cron jobs for GMN's asynchronous processes

```
sudo -Hu gmn bash -c '
. /var/local/dataone/gmn_venv_py3/bin/activate
GMN_PKG_DIR=`python -c "import site; print(site.getsitepackages()[0])"`
crontab -u gmn ${GMN_PKG_DIR}/dl_gmn/deployment/crontab
,
```

Altering the schedule

By default, the processes are set to run once every hour, with a random delay that distributes network traffic and CN load over time. To alter the schedule, consult the crontab manual:

```
man 5 crontab
```

Local Certificate Authority (CA)

Authentication and authorization in the DataONE infrastructure is based on [X.509](#) v3 certificates.

This section describes how to set up a [CA](#), configure GMN to trust the new CA and how to use the CA to generate *client side certificates* that can then be used for creating authenticated connections to GMN.

MNs that are registered with DataONE must trust the [CILogon](#) CAs. But, for security, CILogon issues certificates that are only valid for 18 hours, and stand-alone nodes do not need to trust CILogon. So both stand-alone and registered instances of GMN are set up to trust a locally generated CA. For stand-alone instances, this is typically the only trusted CA. Registered instances also trust CILogon and DataONE. The local CA enables the administrator of the MN to generate long lived certificates that can be used for creating authenticated connections to the MN. Common uses for these certificates on both stand-alone and registered GMN instances include enabling automated connections to the MN for performing tasks such as populating the Node with Science Objects. In addition, these certificates are used for regular user oriented tasks such as accessing the node via the the DataONE Command Line Client on stand-alone nodes.

Setting up the local CA

The local CA used for signing certificates that will be trusted by this (and no other) instance of GMN.

Run the commands below to:

- Generate local CA folders
- Copy custom OpenSSL configuration file
- Create the certificate database file
- Generate the private key and certificate signing request (CSR)
- Self-sign the CA
- Remove the CA CSR

```
sudo -Hu gmn bash -c '
. /var/local/dataone/gmn_venv_py3/bin/activate
GMN_PKG_DIR=`python -c "import site; print(site.getsitepackages()[0])"`
mkdir -p /var/local/dataone/certs/local_ca/{certs,newcerts,private}
cd /var/local/dataone/certs/local_ca
cp ${GMN_PKG_DIR}/d1_gmn/deployment/openssl.cnf .
touch index.txt
openssl req -config ./openssl.cnf -new -newkey rsa:2048 -keyout private/ca_key.pem -
→out ca_csr.pem
'
```

- Enter a password for the private key. Anyone who gains access to the key can create certificates that will be trusted by your MN unless you protect it with a strong password.
- You will be prompted for the information that will become the DN of your CA certificate. All fields should be filled with valid information. For Common Name, use something like “CA for GMN Client Side Certificates”. Since the DN of the signing CA is included in all signed certificates, this helps indicate the intended use for certificates signed by this CA. The Organization Name indicates where the client side certificates are valid.

```
sudo -Hu gmn bash -c '
cd /var/local/dataone/certs/local_ca
openssl ca -config ./openssl.cnf -create_serial \
-keyfile private/ca_key.pem -selfsign -extensions v3_ca_has_san \
```

(continues on next page)

(continued from previous page)

```
-out ca_cert.pem -infile ca_csr.pem
,
```

Answer “y” on the prompts.

Generate a client side certificate

Generate a client side certificate that is signed by the local CA.

- This certificate will be used in any outgoing connections made by the GMN instance while it is operating in stand-alone mode and for initial tests.
- If more client side certificates are needed in the future, just repeat this section, changing the filenames of the client_*.pem files.
- GMN does not include a system for securely managing the password for the private key of the client side certificate so the password is removed.
- The private key implicitly contains the public key. For some use cases, it can be convenient to split out the public key.

Run the commands below to:

- Generate the private key and certificate signing request (CSR)
- Remove the password from the private key
- Split public key from private key
- Sign the CSR for the client side certificate with the local CA
- Remove the client side certificate CSR

```
sudo -Hu gmn bash -c '
cd /var/local/dataone/certs/local_ca
openssl req -config ./openssl.cnf -new -newkey rsa:2048 -nodes \
-keyout private/client_key.pem -out client_csr.pem
,
```

- You will be prompted for the information that will become the DN of your client side certificate. All fields should be filled with valid information. For the Common Name, provide a brief and unique name such as, “localClient”.

```
sudo -Hu gmn bash -c '
cd /var/local/dataone/certs/local_ca
openssl rsa -in private/client_key.pem -out private/client_key_nopassword.pem
openssl rsa -in private/client_key_nopassword.pem -pubout -out client_public_key.pem
openssl ca -config ./openssl.cnf -in client_csr.pem -out client_cert.pem
,
```

Answer “y” on the prompts.

```
sudo -Hu gmn bash -c '
cd /var/local/dataone/certs/local_ca
rm client_csr.pem
rm ca_csr.pem
,
```

You now have a local CA root certificate and a certificate signed by that root:

ca_cert.pem: The CA root certificate
 private/ca_key.pem: The CA root cert private key

client_cert.pem: The client side certificate
 private/client_key.pem: The client side certificate private key
 private/client_key_nopassword.pem: The client side certificate private key without password
 client_public_key.pem: The client side certificate public key

Set GMN up to trust the local CA root certificate

Add the local CA that was just created to the CAs trusted by GMN.

```
sudo -Hu gmn bash -c '
  cd /var/local/dataone/certs/local_ca
  mkdir -p ../ca
  cp ca_cert.pem ../ca/local_ca.pem
'
sudo -H bash -c '
  cd /var/local/dataone/certs/ca
  c_rehash .
'
```

Install non-trusted client side certificate

Run the following commands to:

- Set GMN up to use the previously created locally signed client side certificate for outgoing connections.

```
sudo -Hu gmn bash -c '
  cd /var/local/dataone/certs/local_ca
  mkdir -p ../client
  cp client_cert.pem private/client_key_nopassword.pem ../client
'
```

Install non-trusted server side certificate

Run the commands below to:

- Ensure that the `ssl-cert` package is installed
- Copy the certificate and key to the GMN standard locations

```
sudo -H bash -c '
  apt install --yes ssl-cert
  mkdir -p /var/local/dataone/certs/server
  cp /etc/ssl/certs/ssl-cert-snakeoil.pem /var/local/dataone/certs/server/server_cert.
↪pem
  cp /etc/ssl/private/ssl-cert-snakeoil.key /var/local/dataone/certs/server/server_
↪key_nopassword.pem
'
```

Final configuration and startup

Run the following commands to:

- Configure the GMN settings that are required for running a local instance of GMN.
- Initialize the GMN database

```

sudo -Hu gmn bash -c '
. /var/local/dataone/gmn_venv_py3/bin/activate
GMN_PKG_DIR=`python -c "import site; print(site.getsitepackages()[0])"`
FQDN=`python -c "import socket; print(socket.getfqdn())"`
DELIMITER=`printf "%.0s" {1..100}`
SETTINGS_PATH=${GMN_PKG_DIR}/d1_gmn/settings.py
cp ${GMN_PKG_DIR}/d1_gmn/settings_template.py ${SETTINGS_PATH}
sed -Ei "s/MIDDLEWARE_CLASSES/MIDDLEWARE/" ${SETTINGS_PATH}
sed -Ei "s/'gm2'/'gm2'/'gm3'/'gm3'/" ${SETTINGS_PATH}
sed -Ei "s/(\s*)(.*)\.server\.name\.com\.*/\1'${FQDN}'/'/" ${SETTINGS_PATH}
python ${GMN_PKG_DIR}/d1_gmn/manage.py migrate --run-syncdb
printf "%s\nUsing FQDN: %s\nIf this is incorrect, correct it in %s\n%s\n" \
    ${DELIMITER} ${FQDN} ${SETTINGS_PATH} ${DELIMITER}

```

Starting GMN

GMN should now be ready to start. Simply restart Apache:

```
sudo service apache2 restart
```

Check the Apache logs for error messages. In case of any issues, refer to [/gmnl/setup/troubleshooting](#)

Continue to the next section to test your new node.

Test the installation on Ubuntu

The new stand-alone GMN instance is now ready to be tested.

After a successful installation, GMN exposes the complete REST interface that DataONE has defined for Member Nodes.

The default installation makes GMN accessible both on the server's loopback (localhost) and external interface(s). So the tests outlined below can be performed on the local server or from a different host by replacing `localhost` with the server's external name or address.

Basic testing via web browser or curl

These initial tests can be performed via a web browser or the `curl` command line utility. By default, stand-alone instances of GMN use a non-trusted “snakeoil” self-signed certificate. The browser will warn about this and may require you to create a security exception. `curl` will need to be started with the `--insecure` switch. For example, `curl --insecure <url>`.

After the stand-alone GMN instance passes the tests, it can be joined to DataONE by performing the `/gm/setup/dl env/env` section of the installation, in which the non-trusted certificate is replaced with a publicly trusted certificate from a 3rd party CA.

Node document

Open:

```
https://localhost/mn/v2
```

You should see an XML document such as this:

```
<?xml version="1.0" ?>
<ns1:node replicate="false" state="up" synchronize="true" type="mn" xmlns:ns1="http://
↪ns.dataone.org/service/types/v2.0">
  <identifier>urn:node:MyMemberNode</identifier>
  <name>My Member Node</name>
  <description>Test Member Node</description>
  <baseURL>https://localhost/mn</baseURL>
  <services>
    <service available="true" name="MNCORE" version="v1"/>
    <service available="true" name="MNRead" version="v1"/>
    <service available="true" name="MNAuthorization" version="v1"/>
    <service available="true" name="MNStorage" version="v1"/>
    <service available="true" name="MNReplication" version="v1"/>
    <service available="true" name="MNCORE" version="v2"/>
    <service available="true" name="MNRead" version="v2"/>
    <service available="true" name="MNAuthorization" version="v2"/>
    <service available="true" name="MNStorage" version="v2"/>
    <service available="true" name="MNReplication" version="v2"/>
  </services>
  <synchronization>
    <schedule hour="*" mday="*" min="42" mon="*" sec="0" wday="?" year="*" />
  </synchronization>
  <subject>CN=urn:node:MyMemberNode,DC=dataone,DC=org</subject>
  <contactSubject>CN=My Name,O=Google,C=US,DC=cilogon,DC=org</contactSubject>
</ns1:node>
```

This is your Node document. It exposes information about your Node to the DataONE infrastructure. It currently contains only default values. The /gm/setup/d1env/env section of the installation includes information on how to customize this document for your node.

Home page

GMN also has a home page with some basic statistics, available at:

```
https://localhost/mn/home
```

Note that the home endpoint is not part of DataONE's API definition for Member Nodes, and so does not include a DataONE API version designator (/v1/ or /v2/) in the URL.

Continue with the next installation section if the node is to be registered with DataONE.

Upgrade OS and GMN to latest versions

Contents:

Upgrade Ubuntu 14.04 or 16.04 to 18.04

OS Upgrade

Ubuntu 16.04 LTS can be upgraded directly to 18.04 LTS with `do-release-upgrade`. 14.04 LTS must first be upgraded to 16.04 LTS, which means that the following procedure must be performed twice.

The `dist-upgrade` steps may seem redundant, but `do-release-upgrade` is more likely to complete successfully when the system is on the latest available kernel and packages.

```
sudo -H bash -c '  
  apt dist-upgrade  
  apt autoremove  
  reboot  
'
```

```
sudo -H bash -c '  
  do-release-upgrade  
  reboot  
'
```

```
sudo -H bash -c '  
  apt dist-upgrade  
  apt autoremove  
  reboot  
'
```

Postgres Upgrade

Ubuntu	Postgres
14.04	9.3
16.04	9.5
18.04	10

As the table shows, upgrading from Ubuntu 14.04 or 16.04 to 18.04 causes Postgres to be upgraded from major version 9 to 10. The database storage formats are not compatible between major versions of Postgres, so the databases must be migrated from the old to the new version of Postgres.

The Ubuntu 18.04 upgrade process will install Postgres 10 side by side with Postgres 9.x. The `pg_upgradecluster` migration script is installed as well. However, the migration itself is not performed.

Run the commands below in order to migrate the databases over to Postgres 10 and remove the old database services.

If upgrading from Ubuntu 14.04, replace 9.5 with 9.3. It is not necessary to perform a database migration after upgrading to 16.04.

```
sudo -H bash -c '  
  pg_dropcluster --stop 10 main  
  pg_upgradecluster 9.5 main  
  apt remove postgresql-9.5*  
  reboot  
'
```

Upgrading

This section describes how to migrate an existing, operational MN to GMN.

instance of GMN v1. If you are working on a fresh install, start at setup.

Because of changes in how later versions of GMN store System Metadata and Science Objects, there is no direct *pip* based upgrade path from 1.x. Instead, 3.x is installed side by side with 1.x and an automatic process migrates settings and contents from v1 to 3.x and switches Apache over to the new version.

The automated migration assumes that GMN v1 was installed with the default settings for filesystem locations and database settings. If this is not the case, constants in the migration scripts must be updated before the procedure will work correctly. Contact us for assistance.

The existing v1 instance is not modified by this procedure, so it is possible to roll back to v1 if there are any issues with the migration or 3.x.

Install GMN 3.x and migrate settings and contents

Prepare pip from PyPI:

```
$ sudo apt install --yes python-pip; \
sudo pip install --upgrade pip; \
sudo apt remove --yes python-pip;
```

Prepare dependencies:

```
$ sudo pip install --upgrade pip virtualenv
$ sudo apt install --yes libffi-dev
```

Create virtual environment for GMN 3.x:

```
$ sudo -u gmn virtualenv /var/local/dataone/gmn_venv_py3
```

Install GMN 3.x from PyPI:

```
$ sudo -u gmn --set-home /var/local/dataone/gmn_venv_py3/bin/pip install dataone.gmn
```

Configure GMN 3.x instance and migrate settings from GMN v1:

```
$ sudo /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/dl_gmn/deployment/
↪migrate_v1_to_v2.sh
```

Migrate contents from GMN v1:

```
$ sudo -u gmn /var/local/dataone/gmn_venv_py3/bin/python \
/var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/dl_gmn/manage.py \
migrate_v1_to_v2
```

Verify successful upgrade:

- Seen from the user side, the main improvement in GMN v2 is that it adds support for v2 of the DataONE API. For any clients that continue to access GMN via the v1 API, there should be no apparent difference between v1 and v2. Clients that access GMN via the v2 API gain access to the new v2 functionality, such as Serial IDs.
- A quick way to check if the node is now running GMN 3.x is to open the v2 Node document in a browser, at <https://your.node.edu/mn/v2>. An XML document which announces both v1 and v2 services should be displayed.

Roll back to GMN v1

If there are any issues with GMN v2 or the migration procedure, please contact us for assistance. While the issues are being resolved, the following procedure will roll back to v1.

This procedure should not be performed after any new objects have been added to v2, as they will become unavailable in v1.

Switch the GMN version served by Apache to v1:

```
$ sudo a2dissite gmn3-ssl
$ sudo a2ensite gmn-ssl
```

Disable v2 services for this MN in the CN Node registry:

```
$ sudo -u gmn /var/local/dataone/gmn/bin/python \
/var/local/dataone/gmn/lib/python3.6/site-packages/gmn/manage.py \
register_node_with_dataone --update
```

APT based install

This avoids dependencies on PyPI, which may result in a more secure deployment.

```
sudo apt update -y
sudo apt dist-upgrade -y

sudo apt install -y \
python3 \
python3-asn1crypto \
python3-certifi \
python3-cffi \
python3-chardet \
python3-contextlib \
python3-cryptography \
python3-django \
python3-idna \
python3-iso8601 \
python3-isodate \
python3-lxml \
python3-msgpack \
python3-pkg-resources \
python3-psycpg2 \
python3-pyasnl \
python3-pycparser \
python3-jwt \
python3-pyparsing \
python3-tz \
python3-pyxb \
python3-rdflib \
python3-requests-toolbelt \
python3-requests \
python3-six \
python3-urllib3 \
python3-zipstream
```

```
# apt install python3-setuptools
```

```
nano /usr/local/lib/python3.6/dist-packages/dataone.pth
```

```
/var/local/dataone/dl_python/lib_client/src
/var/local/dataone/dl_python/lib_common/src
/var/local/dataone/dl_python/gmn/src
/var/local/dataone/dl_python/lib_scimeta/src
```

```
GMN_PKG_DIR=/var/local/dataone/dl_python/gmn/src
FQDN=`python -c "import socket; print(socket.getfqdn())"`
```

```
sudo -H
```

```
apt install --yes apache2 apache2-dev libapache2-mod-wsgi-py3
```

```
cp ${GMN_PKG_DIR}/dl_gmn/deployment/gmn3-ssl.conf /etc/apache2/sites-available
sed -Ei "s/www.example.com/${FQDN}/" ${CONF_PATH}
```

```
a2enmod --quiet wsgi ssl alias
a2ensite --quiet gmn3-ssl
```

```
apt install --yes postgresql
```

```
passwd -d postgres
su postgres -c passwd
```

```
su postgres -c 'createuser gmn'
su postgres -c 'createdb -E UTF8 gmn2'
```

```
mkdir -p /var/local/dataone/certs/local_ca/{certs,newcerts,private}
cd /var/local/dataone/certs/local_ca
cp ${GMN_PKG_DIR}/dl_gmn/deployment/openssl.cnf .
touch index.txt
```

```
openssl req -config ./openssl.cnf -new -newkey rsa:2048 \
-keyout private/ca_key.pem -out ca_csr.pem
```

```
openssl ca -config ./openssl.cnf -create_serial \
-keyfile private/ca_key.pem -selfsign -extensions v3_ca_has_san \
-out ca_cert.pem -infiles ca_csr.pem
```

```
cd /var/local/dataone/certs/local_ca
```

```
openssl req -config ./openssl.cnf -new -newkey rsa:2048 -nodes \
-keyout private/client_key.pem -out client_csr.pem
```

Install non trusted certs here.

```
cd /var/local/dataone/certs/local_ca
mkdir -p ../ca
cp ca_cert.pem ../ca/local_ca.pem
sudo c_rehash ../ca
```

```
cp ${GMN_PKG_DIR}/dl_gmn/settings_template.py ${GMN_PKG_DIR}/dl_gmn/settings.py
```

```
chown -R gmn:www-data /var/local/dataone/  
chmod -R g+w /var/local/dataone/  
timedatectl set-timezone Etc/UTC  
ufw allow 443
```

```
python3 ${GMN_PKG_DIR}/dl_gmn/manage.py migrate --run-syncdb
```

```
settings.py:  
  
MIDDLEWARE -> MIDDLEWARE
```

Setup on CentOS

This section describes the initial steps in setting up *GMN*. It has been verified CentOS 7.3. Instructions for Ubuntu are *also available*.

If only this section is completed, the resulting installation is a stand-alone test instance of GMN. The stand-alone instance can be used for performance testing, developing scripts for populating the MN and for learning about MNs in general.

By completing *Registering the new MN in a DataONE environment*, the stand-alone test instance can then be joined to DataONE as an official Member Node.

Contents:

CentOS 7.3 Firewall Setup

Install firewalld

GMN will require ports 80 and 443 to be opened. So after logging in to your server as a user with sudoer privileges, the first step is to setup. We begin by ensuring that the firewall management package is installed on your server and started.

Update yum.:

```
$ sudo yum -y update
```

Install firewalld:

```
$ sudo yum install firewalld  
$ sudo systemctl unmask firewalld  
$ sudo systemctl start firewalld
```

Configure Firewall with Network Interfaces

Next we want to achieve the binding of network interfaces to firewalld zones. This example uses the default public zone. First we need to identify your network interfaces.:

```
$ ifconfig -a
```

The interfaces described in response will look something like this:


```

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 138.197.100.216 netmask 255.255.240.0 broadcast 138.197.111.255
inet6 fe80::3c64:d3ff:fe95:187b prefixlen 64 scopeid 0x20<link>
ether 3e:64:d3:95:18:7b txqueuelen 1000 (Ethernet)
RX packets 467254 bytes 268127560 (255.7 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 335825 bytes 72203530 (68.8 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4098<BROADCAST,MULTICAST> mtu 1500
ether f2:ac:61:7b:73:10 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1 (Local Loopback)
RX packets 81687 bytes 26998580 (25.7 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 81687 bytes 26998580 (25.7 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

There should be one or more network interfaces available, such as “eth0” or “eth1”. Ignore an entry such as “*LOOPBACK,RUNNING*”. The firewall management system we are using binds these network interfaces to something called a “zone”. There is the potential for multiple zones which can have different configuration options, but we aren’t going to worry about that here. We just need the simplest configuration using the default zone. The *public zone* will be the default. So at this point we will check whether or not the network interfaces we identified with “ifconfig -a” are bound to the public zone. We can check that with this command:

```
$ sudo firewall-cmd --zone=public --list-all
```

Which return:

```

public (active)
target: default
icmp-block-inversion: no
interfaces:
sources:
services: dhcpv6-client http https ssh
ports: 443/tcp
protocols:
masquerade: no
forward-ports:
sourceports:
icmp-blocks:
rich rules:

```

If the space next to the “interfaces” line contains the network interfaces, such as eth0 and eth1 in this example, then they are already bound to the public zone. However, if that line is empty, you will need to bind your network interfaces to the firewall zone as follows.

Bind Network Interfaces to Zone:

```
$ sudo firewall-cmd --permanent --zone=public --change-interface=eth0
$ sudo firewall-cmd --permanent --zone=public --change-interface=eth1
$ sudo firewall-cmd --reload
```

Substituting the names of your interfaces in `--change-interface=`. Now, when you enter the command:

```
$ sudo firewall-cmd --zone=public --list-all
```

The network interfaces should be listed:

```
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0 eth1
  sources:
  services: dhcpv6-client ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  sourceports:
  icmp-blocks:
  rich rules:
```

Another way to confirm that everything is as it should be is to use this command:

```
$ firewall-cmd --get-active-zones
```

Which will return output similar to:

```
public
  interfaces: eth1 eth0
```

Open HTTP & HTTPS Ports

Now we can specify rules for handling specific ports and services, using the below commands.:

```
$ sudo firewall-cmd --permanent --add-service=http
$ sudo firewall-cmd --permanent --add-service=https
$ sudo firewall-cmd --permanent --add-port=80/tcp
$ sudo firewall-cmd --permanent --add-port=443/tcp
$ sudo firewall-cmd --reload

$ sudo systemctl enable firewalld
```

Install Web Server & Create GMN User

Apache

Install Apache:

```
$ sudo yum -y install httpd
```

Start Apache and configure to start on boot:

```
$ sudo systemctl start httpd
$ sudo systemctl enable httpd
$ sudo systemctl enable httpd.service
```

To confirm apache is running, check the status:

```
$ sudo systemctl status httpd
```

Now is a good time to check if Apache is listening on port 80 by default as it should be:

```
$ netstat -ln | grep -E :'80'
```

Which will return output similar to:

```
tcp6          0      0 :::80          :::*           LISTEN
```

If the command returns nothing, then something isn't right. You should go back and review the previous steps.

Create GMN User

Change ownership of document root so it and its contents are in the same group as the web server:

```
$ sudo chgrp -R apache /var/www/html
```

Now that apache is installed you can create a user and add it to the apache group:

```
$ sudo useradd -G apache gmn
$ sudo passwd gmn
```

Installing Postgres

Install Postgres:

```
$ sudo yum -y install postgresql postgresql-devel postgresql-libs postgresql-plpython_
↪ postgresql-server
```

Initialize the Database & Configure to start on boot:

```
$ sudo postgresql-setup initdb
$ sudo service postgresql start
$ sudo chkconfig postgresql on

$ sudo passwd -d postgres
$ sudo su postgres -c passwd

$ sudo -u postgres createuser gmn
$ sudo -u postgres createdb -E UTF8 gmn2
```

Installing GMN Software & Supporting Packages

Install Packages & Pip

Install development tools and other needed packages:

```
$ sudo yum groupinstall -y 'development tools'
$ sudo yum -y install python-devel openssl-devel libxml2-devel
$ sudo yum -y install libxslt-devel libffi-devel curl mod_ssl
$ sudo yum -y install openssl-perl gcc mod_wsgi
```

Install pip:

```
$ sudo easy_install pip
```

Install GMN Software in Virtual Environment

Install the virtualenv command. We can get this using pip:

```
$ pip install virtualenv
```

Setup directories:

```
$ sudo mkdir -p /var/local/dataone/{gmn_venv_py3,gmn_object_store}
$ cd /var/local/dataone
$ sudo chown gmn:apache gmn_venv_py3
```

Create and activate a virtual environment in the gmn_venv_py3 directory:

```
sudo -Hu gmn bash -c '
    virtualenv gmn_venv_py3
    source gmn_venv_py3/bin/activate
    pip install --upgrade setuptools==33.1.1
    pip install dataone.gmn
'
```

Configure the GMN Python virtual environment to be the default for the gmn user.:

```
$ su MySudoUser
$ sudo vi /home/gmn/.bashrc
```

This will take you into a text editor. Use the “i” key to enter insert mode. You will see the word ‘INSERT’ at the bottom when this is active, which means you can edit the contents. Add the following lines to the end of the file.:

```
# This next line added as part of GMN installation setup:
PATH="$PATH":/var/local/dataone/gmn_venv_py3/bin/
```

Then use Escape key and “:wq” to write the changes to the file and exit the editor.

Setup GMN Vhost & SSL Configuration

First we will remove default SSL.conf and add a custom config virtual host file. The basic unit of configuration for an individual website is called a “virtual host” or “vhost”. A virtual host directive will let the server know where to find your site’s files, where you want log files to go, which port the configuration applies to, and can contain a variety of other instructions for how the web server should handle this site. For your convenience, a virtual host file is already provided with your GMN installation files.

Remove the default ssl.conf file containing a default vhost:

```
$ cd /etc/httpd/conf.d/
$ sudo rm ssl.conf
```

Copy over and edit gmn3-ssl.conf virtual host file:

```
$ sudo cp /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/
→deployment/gmn3-ssl.conf /etc/httpd/conf.d/
$ sudo vi gmn3-ssl.conf
```

Change the ServerName to your domain, which should already be pointed at your server's IP. This must be consistent with the domain as it will be expressed when registering an SSL certificate.

Change server admin to appropriate email contact.

Replace {`$APACHE_DIR`} in log file declarations with "logs" because that is defined for Apache on Debian. So the log declarations should read:

```
ErrorLog          logs/error.log
CustomLog         logs/access.log combined
```

Add the below text to the top of the file, above the start of the `<IfModule mod_ssl.c>` directive and Virtualhost entry. This is basically combining the default ssl.conf configurations with the GMN virtual host configuration:

```
LoadModule ssl_module modules/mod_ssl.so

# These lines come from the default ssl.conf file:
Listen 443 https
SSLPassPhraseDialog exec:/usr/libexec/httpd-ssl-pass-dialog
SSLSessionCache      shmcb:/run/httpd/sslcache(512000)
SSLSessionCacheTimeout 300
SSLRandomSeed startup file:/dev/urandom 256
SSLRandomSeed connect builtin
SSLCryptoDevice builtin

# Disable for protection against vulnerabilities such as POODLE.
SSLProtocol all -SSLv3 -SSLv2

SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384
→EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 EECDH
→EDH+aRSA RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS !RC4"
```

Don't try to restart apache yet! Ordinarily, one might expect to restart apache at this point. However, the custom .conf file just copied over contains several references to certificate files and directories we have not yet created, so a restart would fail at this point.

Install Server SLL Certificate

SSL certificates accomplish several things. For example, they provide for encrypted communication. They also support a feature whereby browsers will recognize whether or not a server's SSL certificate has come from a trusted Certificate Authority.

These instructions cover three options for handling server SSL certificates. Please review all three options and select the one that is most appropriate for your installation.

Option 1: Install an Externally Generated SSL Cert from a Trusted CA

Either you or your IT department may have already acquired an SSL certificate registered for the domain that is pointing to the server where you are installing GMN. If so, then just create the directory:

```
$ sudo mkdir -p /var/local/dataone/certs/server
```

And copy your server certificate and corresponding key into it.

Option 2: Create a Self-Signed SSL Cert for Testing

It is possible to create a self-signed certificate. While this certificate will not be trusted by browsers, it will still be useful for testing that our SSL configurations are working. The below command will generate the certificate and key, putting them in the location where the gmn3-ssl.conf file has been told to look for it:

```
$ sudo mkdir -p /var/local/dataone/certs/server
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /var/local/dataone/
↪certs/server/server_key_nopassword.pem -out /var/local/dataone/certs/server/server_
↪cert.pem
```

You will be asked to enter some information as shown below. Be sure to enter the domain being pointed at your server's IP for the Common Name. Don't forget that this should be consistent with the domain we configured as the ServerName in the gmn3-ssl.conf file:

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:Tennessee
Locality Name (eg, city) [Default City]:Oak Ridge
Organization Name (eg, company) [Default Company Ltd]:GMN Test Org
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:gmnn.example.edu
Email Address []: admin@example.edu
```

At this point you should be able to navigate to your domain in a browser or with curl and see the apache default webpage. If using a browser, you'll have to exception the security complaint, as the browser won't trust a self-signed certificate as secure. If using curl, you'll need to add the `--insecure` option. For example:

```
$ curl --insecure https://gmnn.example.edu
```

Option 3: Create and Install a Free Cert using Let's Encrypt

If you don't have a certificate from a CA yet, you can make one for free. Let's Encrypt is "a free, automated, and open certificate authority (CA), run for the public's benefit". It is widely gaining traction as an alternative to the traditionally pricey cost of SSL certificates from trusted Certificate Authorities. More information is available at <https://letsencrypt.org/>. This section will walk you through the steps of acquiring an SSL Cert from Let's Encrypt. The instructions assume that (A) you have already pointed your domain/subdomain at the installation server, and (B) the virtual host for the domain has already been configured in the web server.

A package called Certbot is the easiest way to generate a certificate using Let's Encrypt. Certbot is found in EPEL (Extra Packages for Enterprise Linux). To use Certbot, you must first enable the EPEL repository:

```
$ sudo yum install epel-release
```

Install Certbot:

```
$ sudo yum install python-certbot-apache
```

Certbot has a fairly solid beta-quality Apache plugin, which is supported on many platforms, and automates both obtaining and installing certs. To generate the certificate:

```
$ sudo certbot --apache -d gmn.example.edu
```

You will be asked to enter a contact email. Choose the https only option when you get to that part. After the script is finished, you should be provided a link you can use to test your SSL configurations, such as:

```
-----
Congratulations! You have successfully enabled https://centos7-3gmn.kitty.ninja

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=gmn.example.edu
-----
```

Clicking on that link will take you to a recommended tool by SSL labs which will provide a rating for your server's SSL configurations. Following the recommended configurations outlined in these instructions should result in getting an A rating. If you get less than that, feel free to ask for suggestions on how to improve your rating.

Use the following command to show information about the certificate that was generated:

```
$ sudo certbot certificates
```

Which will provide output similar to:

```
-----
Found the following certs:
  Certificate Name: gmn.example.edu
    Domains: gmn.example.edu
    Expiry Date: 2017-06-21 18:22:00+00:00 (VALID: 89 days)
    Certificate Path: /etc/letsencrypt/live/gmn.example.edu/fullchain.pem
    Private Key Path: /etc/letsencrypt/live/gmn.example.edu/privkey.pem
-----
```

However, you should not need to update your vhost with this information. If you go back and look at the contents of the GMN virtual host file:

```
$ vi /etc/httpd/conf.d/gmn3-ssl.conf
```

You'll see that the paths for SSLCertificateFile and SSLCertificateKeyFile have automatically been updated for you.

Local Certificate Authority (CA)

Authentication and authorization in the DataONE infrastructure is based on [X.509](#) v3 certificates.

This section describes how to set up a [CA](#), configure GMN to trust the new CA and how to use the CA to generate *client side certificates* that can then be used for creating authenticated connections to GMN.

MNs that are registered with DataONE must trust the [CILogon](#) CAs. But, for security, CILogon issues certificates that are only valid for 18 hours, and stand-alone nodes do not need to trust CILogon. So both stand-alone and registered instances of GMN are set up to trust a locally generated CA. For stand-alone instances, this is typically the only trusted CA. Registered instances also trust CILogon and DataONE. The local CA enables the administrator of the MN to generate long lived certificates that can be used for creating authenticated connections to the MN. Common uses for these certificates on both stand-alone and registered GMN instances include enabling automated connections to the

MN for performing tasks such as populating the Node with Science Objects. In addition, these certificates are used for regular user oriented tasks such as accessing the node via the the DataONE Command Line Client on stand-alone nodes.

Setting up the local CA

The local CA used for signing certificates that will be trusted by this (and no other) instance of GMN.

Generate local CA folders:

```
$ sudo mkdir -p /var/local/dataone/certs/local_ca/{certs,newcerts,private}
$ cd /var/local/dataone/certs/local_ca
```

Copy custom OpenSSL configuration file:

```
$ sudo cp /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/dl_gmn/
→deployment/openssl.cnf .
```

Create the certificate database file:

```
$ sudo touch index.txt
```

Generate the private key and certificate signing request (CSR):

```
$ sudo openssl req -config ./openssl.cnf -new -newkey rsa:2048 \
-keyout private/ca_key.pem -out ca_csr.pem
```

Enter a password for the private key. Anyone who gains access to the key can create certificates that will be trusted by your MN unless you protect it with a strong password.

You will be prompted for the information that will become the DN of your CA certificate. All fields should be filled with valid information. For Common Name, use something like “CA for GMN Client Side Certificates”. Since the DN of the signing CA is included in all signed certificates, this helps indicate the intended use for certificates signed by this CA. The Organization Name indicates where the client side certificates are valid.

Self-sign the CA:

```
$ sudo openssl ca -config ./openssl.cnf -create_serial \
-keyfile private/ca_key.pem -selfsign -extensions v3_ca_has_san \
-out ca_cert.pem -infiles ca_csr.pem
```

Answer “y” on the prompts.

The CSR is no longer needed and can be removed:

```
$ sudo rm ca_csr.pem
```

Generate a client side certificate

Generate a client side certificate that is signed by the local CA.

This certificate will be used in any outgoing connections made by the GMN instance while it is operating in stand-alone mode and for initial tests.

If more client side certificates are needed in the future, just repeat this section, changing the filenames of the client_*.pem files.

Generate the private key and certificate signing request (CSR):

```
$ cd /var/local/dataone/certs/local_ca
$ sudo openssl req -config ./openssl.cnf -new -newkey rsa:2048 -nodes \
-keyout private/client_key.pem -out client_csr.pem
```

You will be prompted for the information that will become the DN of your client side certificate. All fields should be filled with valid information. For the Common Name, provide a brief and unique name such as, “localClient”.

GMN does not include a system for securely managing the password for the private key of the client side certificate so the password is removed.

Remove the password from the private key:

```
$ sudo openssl rsa -in private/client_key.pem \
-out private/client_key_nopassword.pem
```

The private key implicitly contains the public key. For some use cases, it can be convenient to split out the public key.

Split public key from private key:

```
$ sudo openssl rsa -in private/client_key_nopassword.pem -pubout \
-out client_public_key.pem
```

Sign the CSR for the client side certificate with the local CA:

```
$ sudo openssl ca -config ./openssl.cnf -in client_csr.pem \
-out client_cert.pem
```

Answer “y” on the prompts.

The CSR is no longer needed and can be removed:

```
$ sudo rm client_csr.pem
```

You now have a local CA root certificate and a certificate signed by that root:

```
ca_cert.pem: The CA root certificate
private/ca_key.pem: The CA root cert private key

client_cert.pem: The client side certificate
private/client_key.pem: The client side certificate private key
private/client_key_nopassword.pem: The client side certificate private key without
password
client_public_key.pem: The client side certificate public key
```

Set GMN up to trust the local CA root certificate

Add the local CA that was just created to the CAs trusted by GMN:

```
$ cd /var/local/dataone/certs/local_ca
$ sudo mkdir -p ../ca
$ sudo cp ca_cert.pem ../ca/local_ca.pem
$ sudo c_rehash ../ca
```

Install non-trusted client side certificate

In addition to acting as servers in the DataONE infrastructure, Member Nodes also act as clients, initiating connections to other Nodes. When connecting to other Nodes, Member Nodes authenticate themselves in a process called *client side authentication*, in which a client side certificate is provided over an LTS/SSL connection.

Nodes that are registered with DataONE will only trust Member Node connections where a client side certificate issued by the DataONE CA is provided. However, a stand-alone instance of GMN will not connect to registered Member Nodes, so a non-trusted client side certificate can be used instead.

These instructions use a non-trusted client side certificate for the first part of the install and describe how to upgrade the certificate to a certificate issued by the DataONE CA in the optional section on how to register the node.

If you already have a client side certificate issued by the DataONE CA, you can still install the non-trusted certificate here and just follow the instructions to upgrade it later.

Copy the previously created locally signed client side certificate for outgoing connections:

```
$ cd /var/local/dataone/certs/local_ca
$ sudo mkdir -p ../client
$ sudo cp client_cert.pem private/client_key_nopassword.pem ../client
```

Configure the GMN asynchronous processes

CNs may send various messages to MNs. These include replication requests and System Metadata update notifications. Such requests are queued by GMN and handled asynchronously.

The asynchronous processes are implemented as Django management commands that are launched at regular intervals by *cron*. The management commands examine the queues and process the requests.

The asynchronous processes connect to CNs and other MNs on behalf of your GMN instance. These connections are made over TLS/SSL and use the client side certificate stored in `/var/local/dataone/certs/client`.

Set up cron jobs

Edit the cron table for the gmn user:

```
$ sudo crontab -e -u gmn
```

Add:

```
GMN_ROOT = /var/local/dataone/gmn_venv_py3
SERVICE_ROOT = /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/
↳dl_gmn
PYTHON_BIN = /var/local/dataone/gmn_venv_py3/bin/python

* * * * * sleep $(expr $RANDOM \% $(30 * 60)) ; $PYTHON_BIN $SERVICE_ROOT/
↳manage.py process_replication_queue >> $SERVICE_ROOT/gmn_replication.log 1>
↳&1
# Process the System Metadata refresh queue
* * * * * sleep $(expr $RANDOM \% $(30 * 60)) ; $PYTHON_BIN $SERVICE_ROOT/
↳manage.py process_refresh_queue >> $SERVICE_ROOT/gmn_sysmeta.log 2>&1
```

This sets the processes to run once every hour, with a random delay that distributes network traffic and CN load over time. To alter the schedule, consult the crontab manual:

```
$ man 5 crontab
```

Basic Configuration

Configure the GMN settings that are required for running a local instance of GMN.

Create a copy of the GMN site settings template:

```
$ cd /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn
$ sudo cp settings_template.py settings.py
```

For a basic local standalone install, all the settings can be left at their defaults.

Final configuration and startup

Filesystem permissions

Set all the files to be owned by the gmn account, and to be writable by www-data:

```
$ sudo chown -R gmn:apache /var/local/dataone
$ sudo chmod -R g+w /var/local/dataone/
```

Initialize the database

```
sudo -Hu gmn bash -c '
  cd /var/local/dataone
  source gmn_venv_py3/bin/activate
  python /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/
↪manage.py migrate --run-syncdb
'
```

Set server to UTC timezone (recommended)

GMN translates incoming date-times to UTC and provides outgoing date-times in UTC. Because of this, it is convenient to run the server in UTC, so that server related timestamps, such as the ones in logs, match up with timestamps stored in the GMN database and provided in DataONE REST API responses.

To check your time format:

```
$ date
```

The output should specify that that time given is in UTC, for example:

```
[mySudoerUser@centos7 dataone]$ date
Thu Mar 23 20:27:52 UTC 2017
```

If not in UTC time, try:

```
$ rm -f /etc/localtime; ln -s /usr/share/zoneinfo/UTC /etc/localtime
$ echo 'ZONE="UTC"' > /etc/sysconfig/clock
```

Starting GMN

GMN should now be ready to start. Simply restart Apache:

```
$ sudo service httpd restart
```

Check the Apache logs for error messages. In case of any issues, refer to [Troubleshooting](#)

Continue to the next section to test your new node.

Test the installation on CentOS

The new stand-alone GMN instance is now ready to be tested.

After a successful installation, GMN exposes the [complete REST interface](#) that DataONE has defined for [Member Nodes](#).

The default installation makes GMN accessible both on the server's loopback (localhost) and external interface(s). So the tests outlined below can be performed on the local server or from a different host by replacing `localhost` with the server's external name or address.

Basic testing via web browser or curl

These initial tests can be performed via a web browser or the `curl` command line utility. By default, stand-alone instances of GMN use a non-trusted “snakeoil” self-signed certificate. The browser will warn about this and may require you to create a security exception. `curl` will need to be started with the `--insecure` switch. For example, `curl --insecure <url>`.

After the stand-alone GMN instance passes the tests, it can be joined to DataONE by performing the [Registering the new MN in a DataONE environment](#) section of the installation, in which the non-trusted certificate is replaced with a publicly trusted certificate from a 3rd party CA.

Node document

Open:

```
https://localhost/mn/v2
```

You should see an XML document such as this:

```
<?xml version="1.0" ?>
<ns1:node replicate="false" state="up" synchronize="true" type="mn" xmlns:ns1="http://
↪ns.dataone.org/service/types/v2.0">
  <identifier>urn:node:MyMemberNode</identifier>
  <name>My Member Node</name>
  <description>Test Member Node</description>
  <baseURL>https://localhost/mn</baseURL>
  <services>
    <service available="true" name="MNCore" version="v1"/>
    <service available="true" name="MNRead" version="v1"/>
    <service available="true" name="MNAuthorization" version="v1"/>
    <service available="true" name="MNStorage" version="v1"/>
    <service available="true" name="MNReplication" version="v1"/>
    <service available="true" name="MNCore" version="v2"/>
  </services>
</ns1:node>
```

(continues on next page)

(continued from previous page)

```

<service available="true" name="MNRead" version="v2"/>
<service available="true" name="MNAuthorization" version="v2"/>
<service available="true" name="MNStorage" version="v2"/>
<service available="true" name="MNReplication" version="v2"/>
</services>
<synchronization>
  <schedule hour="*" mday="*" min="42" mon="*" sec="0" wday="?" year="*" />
</synchronization>
<subject>CN=urn:node:MyMemberNode,DC=dataone,DC=org</subject>
<contactSubject>CN=My Name,O=Google,C=US,DC=cilogon,DC=org</contactSubject>
</ns1:node>

```

This is your Node document. It exposes information about your Node to the DataONE infrastructure. It currently contains only default values. The *Registering the new MN in a DataONE environment* section of the installation includes information on how to customize this document for your node.

Home page

GMN also has a home page with some basic statistics, available at:

```
https://localhost/mn/home
```

Note that the `home` endpoint is not part of DataONE's API definition for Member Nodes, and so does not include the DataONE API version designator (`/v1/`) in the URL.

Continue with the next installation section if the node is to be registered with DataONE.

Registering the new MN in a DataONE environment

This section assumes that the local setup for *Ubuntu* or *CentOS* has been successfully completed.

Completing this section will enable the stand-alone instance of GMN to be joined to the DataONE infrastructure.

Contents:

Obtain and install the server side certificate

GMN authenticates to incoming connections from *DataONE clients* and other parts of the DataONE infrastructure, such as *CNs* by providing a *server side certificate* during the SSL/TLS handshake.

All nodes that are registered with DataONE must have a valid server side certificate, issued by a publicly trusted *CA* such as VeriSign or Thawte.

The trusted certificate is purchased through the same procedure as for any secure web site. Organizations typically have established procedures for obtaining these certificates or may be using wildcard certificates. The procedure below assumes that a valid certificate has already been obtained.

Setup the server side certificate and private key

Delete the previously installed non-trusted “snakeoil” certificate:

```
$ rm /var/local/dataone/certs/server/{server_cert.pem,server_key_nopassword.
↪pem}
```

Move the trusted certificate and key to the `/var/local/dataone/certs/server` directory and rename them to `server_cert.pem` and `server_key.pem`.

If the key is password protected, Apache will prompt for the password each time it's started. As an optional step, the password can be removed:

```
$ cd /var/local/dataone/certs/server
$ sudo openssl rsa -in server_key.pem -out server_key_nopassword.pem
$ sudo chown root:root server_key.pem server_key_nopassword.pem
$ sudo chmod 400 server_key.pem server_key_nopassword.pem
```

If you wish to retain the password in the key, modify the `SSLCertificateKeyFile` setting in the `/etc/apache2/sites-available/gmn-ssl.conf` Virtual Host file to the path of the password protected key.

Other names and/or locations may also be used. If so, update the `SSLCertificateFile` and `SSLCertificateKeyFile` settings in the `gmn-ssl.conf` Virtual Host file to match.

If the server certificate is signed by intermediate certificate(s), the issuing CA will have provided the intermediate certificate chain in addition to the server side certificate. If so, move the intermediate certificate chain file to the `/var/local/dataone/certs/server` directory and uncomment the `SSLCertificateChainFile` setting for GMN in the `gmn-ssl.conf` Virtual Host file. As with the server side certificate and key, the path in `gmn-ssl.conf` can be adjusted if necessary.

Install the DataONE client side certificate

In addition to acting as servers in the DataONE infrastructure, Member Nodes also act as clients, initiating connections to other Nodes. When connecting to other Nodes, Member Nodes authenticate themselves in a process called *client side authentication*, in which a *client side certificate* is provided to the server.

Obtain the client side certificate

Client side certificates for MNs are issued by the DataONE CA. MNs go through a testing phase before being registered in the DataONE production environment used by the public, so DataONE will first issue a test certificate to your node. The test certificate is valid only in DataONE's test environments. When the MN is ready to join the production environment, DataONE will issue a production certificate for your node. The certificates are valid for several years and are linked to your MN via their *DNs*.

To obtain a client side certificate for testing:

1. Work with DataONE to determine a Node ID on the form, `urn:node:NODEID`, for your node.
2. Create an account on the [DataONE Registration page](#),
3. Notify DataONE by sending an email to support@dataone.org. In the email, state that you are requesting a client side certificate for a new MN and include the agreed upon Node ID, on the form `urn:node:NODEID`.
4. DataONE will create the certificate for you and notify you of its creation with a reply to your email.
5. Follow the link provided in the email, and sign in using the account created or used in the first step, above.

Warning: Anyone who has the private key can act as your Node in the DataONE infrastructure. Keep the private key safe. Set it to be readable only by root and follow best practices for security to keep root from being compromised. If your private key becomes compromised, please inform DataONE so that the certificate can be revoked and a new one generated.

Install the client side certificate

When the signed client side certificate has been received from DataONE, move it and its private key to the `/var/local/dataone/certs/client` folder.

Rename the files to `client_cert.pem` and `client_key.pem`.

Remove the password from the key:

```
$ cd /var/local/dataone/certs/client
$ sudo openssl rsa -in client_key.pem -out client_key_nopassword.pem
$ sudo chown root:root client_key.pem client_key_nopassword.pem
$ sudo chmod 400 client_key.pem client_key_nopassword.pem
```

Other names and/or directories may be used. If so, update `CLIENT_CERT_PATH` and `CLIENT_CERT_PRIVATE_KEY_PATH` in the `GMN settings.py` file to the new paths.

Install CILogon and DataONE root CA certificates

For a client side certificate to be considered valid by GMN, GMN must trust the CA that signed the client side certificate. This step sets up the CAs to be trusted.

Two basic types of client side certificates are used in the DataONE infrastructure. The first type is issued by the *CILogon* CA and is used for authenticating users. The second type is issued by the DataONE CA and is used for authenticating Nodes.

CILogon is the identity provider for DataONE. CILogon provides three *LOAs*. These instructions set GMN up to accept all three.

DataONE issues certificates that let Nodes communicate securely in the DataONE infrastructure. The DataONE CA root certificates must be trusted by all Nodes.

The OS typically comes with a complete set of commonly trusted CA root certificates. However, DataONE Nodes should not accept certificates signed by these, so a separate CA store is used for the CILogon and DataONE root CAs.

Two separate certificate chains are available. One is used for the DataONE production environment and one is used for all the testing and development environments. Only the DataONE CA differs between the chains.

Create a folder for the CA certificates:

```
$ sudo mkdir -p /var/local/dataone/certs/ca
```

Run **one** of the commands below, depending on which environment the MN is being registered into.

Registering in a testing environment (Staging, Sandbox, Development):

```
# Only run this command when registering the MN in a testing environment
$ sudo curl -o /var/local/dataone/certs/ca/DataONECACHain.crt \
https://repository.dataone.org/software/tools/trunk/ca/DataONETestCACHain.
↪ crt; \
c_rehash /var/local/dataone/certs/ca/
```

Registering in production:

```
# Only run this command when registering the MN in production
$ sudo curl -o /var/local/dataone/certs/ca/DataONECACHain.crt \
https://repository.dataone.org/software/tools/trunk/ca/DataONECACHain.crt; \
c_rehash /var/local/dataone/certs/ca/
```

Register the new Member Node with DataONE

A Member Node (MN) integrates itself into DataONE through a process called Node Registration. Registering the MN allows the Coordinating Nodes (CNs) to synchronize content, index the metadata and resource maps, and replicate its content to other MNs.

MNs go through a testing phase before being registered in the DataONE production environment used by the public. This document describes how to register the new MN in a test environment. When the MN is ready to be registered in the production environment, the same procedure is used.

Registering the MN in a testing environment involves the following steps:

1. Creating a DataONE identity in the environment.
2. Submitting a Node document. The Node document describes the MN and the level at which it will participate in the DataONE infrastructure.
3. DataONE evaluates the submission. Upon approval, the registration is complete, and the Node is part of the DataONE infrastructure.

Perform the steps in order, as each step depends on earlier steps.

Create a DataONE identity

This step must be performed by the person who will be the contact for the new MN. The contact person is often also the administrator for the MN.

Each DataONE environment has a web-based Identity Manager where DataONE identities are created and maintained. To create a DataONE identity, you will use the Identity Manager to authenticate with a *CILogon*-recognized identity, and then attach your name and contact email. At this point, DataONE will validate the information manually.

To register the administrative contact's DataONE identity in the target environment, perform the following steps:

1. Navigate to the Identity Manager of the target environment:

Environment	Identity Manager URL
Production	https://cn.dataone.org/portal
Staging	https://cn-stage.test.dataone.org/portal
Sandbox	https://cn-sandbox.test.dataone.org/portal
Development	https://cn-dev.test.dataone.org/portal

2. Follow the prompts to authenticate against your *Identity Provider*. If your institution is not listed, you can use a Google or ProtectNetwork account.
3. Once authenticated and back at the DataONE portal, supply your name and email, and then press **Register**
4. Record (copy to clipboard) the identity string shown in the 'Logged in as' field. This value is taken from the CILogon certificate issued when you authenticated against your chosen *Identity Provider*, and is also a DataONE subject.
5. Paste this value into the contactSubject field of the Node document you plan to submit in the next step.
6. DataONE requires that DataONE subjects that are to be used as contacts for MNs be verified. To verify the account, send an email to support@dataone.org. In the email, include the identity string obtained in the step above and request that the account be verified. You do not need to wait for a reply to continue to the next step.

Configure the Member Node information

Most of the values that are set up in this section are described in the [Node document section in the architecture documentation](#).

The Node document is a set of values that describe a MN or CN, its internet location, and the services it supports.

Modify the following settings:

- **NODE_IDENTIFIER:** A unique identifier for the node of the form `urn:node:NODEID` where `NODEID` is the node specific identifier. This value **MUST NOT** change for future implementations of the same node, whereas the `baseURL` may change in the future.

`NODEID` is typically a short name or acronym. As the identifier must be unique, coordinate with your DataONE developer contact to establish your test and production identifiers. The conventions for these are `urn:node:mnTestNODEID` for the development, sandbox and staging environments and `urn:node:NODEID` for the production environment. For reference, see the [list of current DataONE Nodes](#).

E.g.: `urn:node:USGSCSAS` (for production) and `urn:node:TestUSGSCSAS` (for testing).

- **NODE_NAME:** A human readable name of the Node. This name can be used as a label in many systems to represent the node, and thus should be short, but understandable.

E.g.: USGS Core Sciences Clearinghouse

- **NODE_DESCRIPTION:** Description of a Node, explaining the community it serves and other relevant information about the node, such as what content is maintained by this node and any other free style notes.

E.g.: US Geological Survey Core Science Metadata Clearinghouse archives metadata records describing datasets largely focused on wildlife biology, ecology, environmental science, temperature, geospatial data layers documenting land cover and stewardship (ownership and management), and more.

- **NODE_BASEURL:** The base URL of the node, indicating the protocol, fully qualified domain name, and path to the implementing service, excluding the version of the API.

E.g.: `https://server.example.edu/app/d1/mn`

- **NODE_SUBJECT:** Specify the subject for this Node (retrieved from the client certificate provided by DataONE)
- **NODE_CONTACT_SUBJECT:** The appropriate person or group to contact regarding the disposition, management, and status of this Member Node. The `contactSubject` is an X.509 Distinguished Name for a person or group that can be used to look up current contact details (e.g., name, email address) for the contact in the DataONE Identity service. DataONE uses the `contactSubject` to provide notices of interest to DataONE nodes, including information such as policy changes, maintenance updates, node outage notifications, among other information useful for administering a node. Each node that is registered with DataONE must provide at least one `contactSubject` that has been verified with DataONE.

The `contactSubject` must be the subject of the DataONE identity that was created in the [previous step](#).

E.g.: `CN=My Name,O=Google,C=US,DC=cilogon,DC=org`

- **NODE_REPLICATE:** Set to true if the node is willing to be a *replication target*, otherwise false.
- **DATAONE_ROOT:** Select the environment that matches the one that was selected in [Create a DataONE identity](#).

E.g.: `https://cn-stage.dataone.org/cn`

Submit Member Node information to DataONE

The Member Node information is submitted to DataONE in a Node document. GMN automatically generates the Node document based on the settings configured in the previous step.

After editing `settings.py`, check if the Node document is successfully generated:

```
$ su gmn
$ python /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/dl_gmn/
↪manage.py node view
```

If the Node document is successfully generated, an XML document will be displayed. For more information about this document, refer to <https://releases.dataone.org/online/api-documentation-v2.0.1/apis/Types.html#Types.Node>

When the Node document is successfully generated and displayed, register the MN by submitting the Node document to DataONE. The Node document is automatically submitted to DataONE over a TLS/SSL connection that has been authenticated with the client side certificate configured in *Install the DataONE client side certificate*.

```
$ python lib/python3.6/site-packages/dl_gmn/manage.py node register
```

- Check for a message saying that the registration was successful.

After running the script or running an automated registration, the Member Node should email support@dataone.org to notify of the registration request.

DataONE evaluates the submission

DataONE evaluates the submitted Node document and contacts the person listed as *contactSubject* in the Node document by email with the outcome of the approval process. After the node has been approved, the MN is part of the infrastructure environment in which it has been registered, and the CNs in that environment will start processing the information on the node.

Select the DataONE Tier

DataONE has defined several tiers, each of which designates a certain level of functionality exposed by the Member Node. The tiers enable Member Nodes to implement only the functionality for the level at which they wish to participate in the DataONE infrastructure.

The tiers are as follows:

Tier 1	Read, public objects	Tier 2	Access controlled objects (authentication and authorization)
Tier 3	Write (create, update and delete objects)		
Tier 4	Replication target		

Each tier implicitly includes all lower numbered tiers. For instance, a Tier 3 Node must implement tiers 1, 2 and 3.

GMN supports all tiers. To select the tier for your Member Node, take the following into account:

- A Tier 1 Node is typically used for exposing existing data to DataONE. As there is no support for manipulating the data through DataONE interfaces in this tier, GMN cannot be populated with objects while in this tier. Therefore, GMN should not initially be set to this tier. Instead, set GMN to Tier 3, populate the Node with objects, and then set the Tier to 1.
- A Tier 2 Node allows the access to objects to be controlled via access control lists (ACLs). Using this Tier implies the same strategy as for Tier 1.
- A Tier 3 Node allows using DataONE interfaces to set up the objects on the Member Node, for instance by using the DataONE Command Line Interface or by creating Python scripts or Java programs that are based on the libraries provided by DataONE. The objects can be set up with storage managed either by GMN itself or

by another, independent server that makes the object data available on the web. Access to the write functions is restricted by a whitelist.

- A Tier 4 member Node can act as a replication target, which allows the Member Node operator to provide storage space to DataONE (for storing object replicas).

When you have determined which tier to use, edit `settings.py`:

```
$ sudo nano /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/
↪ settings.py
```

- Set TIER to 1, 2, 3 or 4.

Final configuration and startup

Turn off stand-alone mode

```
$ sudo nano /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/
↪ settings.py
```

- Set `STAND_ALONE` to `False`.

Resources

View documentation for Apache2 configuration under Debian GNU/Linux:

```
$ zless /usr/share/doc/apache2.2-common/README.Debian.gz
```

Viewing the files involved in the SSL handshake:

```
openssl rsa -noout -text -in server.key
openssl req -noout -text -in server.csr
openssl rsa -noout -text -in ca.key
openssl x509 -noout -text -in ca.crt
```

Overview of the SSL handshake:

[SSL Handshake](#)

Add DataONE test certificate to system wide trusted CA store

```
$ sudo -s
$ sudo cp /var/local/dataone/certs/local_ca/ca.crt /usr/share/ca-certificates/dataone-
↪ gmn-test-ca.crt
$ sudo dpkg-reconfigure ca-certificates
$ sudo update-ca-certificates
```

In the `dpkg-reconfigure` GUI, enable the `dataone-gmn-test-ca.crt`.

Integration testing using certificates

Create two test certificates signed by the local CA. We simulate valid and invalid sessions by using “valid” and “invalid” strings in the Common Names.

```
$ cd /var/local/dataone/certs/local_ca
$ sudo openssl genrsa -des3 -out test_valid.key 4096
$ sudo openssl genrsa -des3 -out test_invalid.key 4096
```

Create *CSRs*:

When prompted for Common Name (CN), type “test_valid” for the certificate signed with the test_valid key and “test_invalid” for the certificate signed with the test_invalid key.

```
$ sudo openssl req -new -key test_valid.key -out test_valid.csr
$ sudo openssl req -new -key test_invalid.key -out test_invalid.csr
```

Sign the *CSR* with the *CA signing key*:

```
$ sudo openssl x509 -req -days 36500 -in test_valid.csr -CA ca.crt -CAkey ca.key -set_
↪serial 01 -out test_valid.crt
$ sudo openssl x509 -req -days 36500 -in test_invalid.csr -CA ca.crt -CAkey ca.key -
↪set_serial 01 -out test_invalid.crt
```

Remove passwords from the private keys:

```
$ sudo openssl rsa -in test_valid.key -out test_valid.nopassword.key
$ sudo openssl rsa -in test_invalid.key -out test_invalid.nopassword.key
```

Copy the keys to the integration tests:

```
$ cp test_valid.nopassword.key /var/local/dataone/gmn_venv_py3/src/tests
$ cp test_invalid.nopassword.key /var/local/dataone/gmn_venv_py3/src/tests
```

Migrating Existing Member Node to GMN

This section describes how to migrate an existing, operational MN to GMN.

If you are working on a fresh install, start at [GMN setup overview](#).

Because of changes in how later versions of GMN store System Metadata and Science Objects, there is no direct *pip* based upgrade path from 1.x. Instead, 3.x is installed side by side with 1.x and an automatic process migrates settings and contents from v1 to 3.x and switches Apache over to the new version.

The automated migration assumes that GMN v1 was installed with the default settings for filesystem locations and database settings. If this is not the case, constants in the migration scripts must be updated before the procedure will work correctly. Contact us for assistance.

The existing v1 instance is not modified by this procedure, so it is possible to roll back to v1 if there are any issues with the migration or 3.x.

Install GMN 3.x and migrate settings and contents

Prepare *pip* from PyPI:

```
$ sudo apt install --yes python-pip; \
sudo pip install --upgrade pip; \
sudo apt remove --yes python-pip;
```

Prepare dependencies:

```
$ sudo pip install --upgrade pip virtualenv
$ sudo apt install --yes libffi-dev
```

Create virtual environment for GMN 3.x:

```
$ sudo -u gmn virtualenv /var/local/dataone/gmn_venv_py3
```

Install GMN 3.x from PyPI:

```
$ sudo -u gmn --set-home /var/local/dataone/gmn_venv_py3/bin/pip install dataone.gmn
```

Configure GMN 3.x instance and migrate settings from GMN v1:

```
$ sudo /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/deployment/
↳migrate_v1_to_v2.sh
```

Migrate contents from GMN v1:

```
$ sudo -u gmn /var/local/dataone/gmn_venv_py3/bin/python \
/var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/d1_gmn/manage.py \
migrate_v1_to_v2
```

Verify successful upgrade:

- Seen from the user side, the main improvement in GMN v2 is that it adds support for v2 of the DataONE API. For any clients that continue to access GMN via the v1 API, there should be no apparent difference between v1 and v2. Clients that access GMN via the v2 API gain access to the new v2 functionality, such as Serial IDs.
- A quick way to check if the node is now running GMN 3.x is to open the v2 Node document in a browser, at <https://your.node.edu/mn/v2>. An XML document which announces both v1 and v2 services should be displayed.

Roll back to GMN v1

If there are any issues with GMN v2 or the migration procedure, please contact us for assistance. While the issues are being resolved, the following procedure will roll back to v1.

This procedure should not be performed after any new objects have been added to v2, as they will become unavailable in v1.

Switch the GMN version served by Apache to v1:

```
$ sudo a2dissite gmn3-ssl
$ sudo a2ensite gmn-ssl
```

Disable v2 services for this MN in the CN Node registry:

```
$ sudo -u gmn /var/local/dataone/gmn/bin/python \
/var/local/dataone/gmn/lib/python3.6/site-packages/gmn/manage.py \
register_node_with_dataone --update
```

Hosting multiple Member Nodes from the same GMN instance

A single GMN instance can host multiple separate MNs, referred to as “multi-hosting”. The number of MNs hosted within a GMN instance is limited only by the available hardware.

In a multi-hosting setup, each MN is functionally equivalent to individual MNs hosted on separate servers. They are individually configured and have separate management commands. Each MN has its own database and Science Object storage area in the filesystem.

At the same time, by sharing a server and a Python virtual environment, any upgrades or other system maintenance automatically applies to all the MNs.

Overall, multi-hosting can significantly lower the time required to maintain the services, lower hardware costs, and lower the complexity of deployments.

A multi-hosting setup works by configuring Apache to alias separate MN BaseURLs to separate WSGI configuration files. Each WSGI file then invokes GMN using a separate settings file. In turn, each settings file specifies a different database, Science Object storage area, and log file.

As each MN has its own settings file, the MNs can be configured individually for such things as branding and replication policies.

In order to be able to specify which MN a management command applies to, separate management commands are set up as well.

As with MNs hosted on separate servers, each MN needs to have a unique BaseURL. If the BaseURLs use the same domain name, they can share a single server side certificate and DNS record. E.g.:

```
https://xyz.node.edu/mn-one
https://xyz.node.edu/mn-two
```

Each MN can also have completely unrelated BaseURLs, as long as all the domain names resolve to the same server. In such as setup, the server can be set up to issue separate server side certificates for each MN, or both MNs can issue a shared certificate that covers both domain names. E.g.:

```
https://mn-one.some.domain.edu/mn/
https://mn-two.another.domain.org/some/path/
```

Example

What follows is a complete example on how to add a second MN to a GMN instance that has already been set up as described in the standard setup procedure, and is currently exposing a single working MN. After adding the second MN, there will be no difference to the original MN as seen from the user side.

In this example, we'll just call the original MN, "a", and the new MN, "b". In a real setup, these names would be selected to reflect the actual MN names.

If the new MN is intended to be joined to a DataONE environment, start by obtaining a client side certificate for the MN from DataONE. If the MN will be used for local testing only, a self signed certificate can be generated as described in [Local Certificate Authority \(CA\)](#). Make sure to modify the names of the output files if previously generated files are still in use. Then install the certificates as described in [Install non-trusted client side certificate](#).

```
$ cdgmn
$ sudo service apache2 stop

$ mv wsgi.py wsgi_a.py
$ mv settings.py settings_a.py
$ mv manage.py manage_a.py

$ cp wsgi_a.py wsgi_b.py
$ cp settings_a.py settings_b.py
$ cp manage_a.py manage_b.py
```

(continues on next page)

(continued from previous page)

```
$ editor wsgi_a.py
* Edit: dl_gmn.settings -> dl_gmn.settings_a

$ editor manage.py
* Edit: dl_gmn.settings -> dl_gmn.settings_a

$ editor wsgi_b.py
* Edit: dl_gmn.settings -> dl_gmn.settings_b

$ editor manage.py
* Edit: dl_gmn.settings -> dl_gmn.settings_b

$ editor settings_b.py
* Edit the settings as if setting up a regular new MN on a separate server
* In addition:
  * Change the following settings so that they're different from the values
  * used by the original MN:
  * NODE_BASEURL, DATABASE.NAME, LOG_PATH, OBJECT_STORE_PATH
  * For this example, we'll assume that we just added "_b" to the values
```

Create and initialize a database for the new MN:

```
$ su postgres -c 'createdb -E UTF8 gmn2_b'
$ ./manage_b.py migrate --run-syncdb
```

Configure Apache:

```
$ sudo -e /etc/apache2/sites-enabled/gmn3-ssl.conf
```

Duplicate and modify *WSGIScriptAlias* and *WSGIDaemonProcess* as follows. This pattern is used when the MNs use the same domain main in the BaseURL. It leaves the original MN available under the same BaseURL as before, and exposes the new MN under /mn_b/.

```
WSGIScriptAlias      /mn      ${gmn_root}/wsgi_a.py
WSGIScriptAlias      /mn_b    ${gmn_root}/wsgi_b.py
WSGIDaemonProcess    gmn_a    user=gmn processes=2 threads=25
WSGIDaemonProcess    gmn_b    user=gmn processes=2 threads=25
```

Add a new section to apply separate process groups to each MN (without this, both MNs will randomly be served from both BaseURLs):

```
<Location /mn>
  WSGIProcessGroup gmn_a
  SSLOptions +ExportCertData
</Location>
<Location /mn_b>
  WSGIProcessGroup gmn_b
  SSLOptions +ExportCertData
</Location>
```

Create crontab entries for the async jobs for the new MN:

```
$ crontab -e
```

Duplicate the two crontab entries, then change the first two from *manage.py* to *manage_a.py* and the last two to *manage_b.py*. Similarly append *_a* and *_b* to the log filenames.

Then all that remains is to start Apache again to make the new MN available for use.

```
$ sudo service apache2 start
```

Management commands for the original MN are now launched via *manage_a.py*, and via *manage_b.py* for the new MN. E.g., to register the new MN in a DataONE environment, use *manage_b.py node register*.

Depending on how backups are performed on the server, the new database and the Science Object storage area for the new MN may have to be added to the procedures.

Other administrative procedures, such as OS, GMN and DataONE Python stack upgrades, likely remain unchanged.

Troubleshooting

Psycopg2 gives “can’t adapt” errors

The version of Psycopg2 that was installed by default was not compatible.

Try uninstalling the default version and installing a specific version. The version referenced below had been found to work well with Postgres 8.4.10.

```
$ sudo apt-get remove python-psycopg2
$ mkdir ~/install
$ cd ~/install
$ wget http://initd.org/psycopg/tarballs/PSYCOPG-2-4/psycopg2-2.4.2.tar.gz
$ tar xzf psycopg2-2.4.2.tar.gz
$ cd psycopg2-2.4.2
$ sudo python setup.py install
```

Another option is to try to install Psycopg2 via *easy_install*:

```
$ sudo easy_install -m psycopg2
```

To remove a version of Psycopg2 that was installed with *easy_install*:

```
$ sudo rm -rf /usr/lib/python3.6/dist-packages/psycopg2
```

`SSLError(SSLError(1, '[SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown ca (_ssl.c:2273)'),`

SSL/TLS Troubleshooting

Commonly seen OpenSSL SSL/TLS connection errors and X.509 client or server side certificate configuration errors with possible causes and solutions.

Error Code 1

```
SSLError(SSLError(1, '[SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown
ca (_ssl.c:2273)'), ssl.SSLError: [Errno 1] _ssl.c:510: error:14094418:SSL rou-
tines:SSL3_READ_BYTES:tlsv1 alert unknown ca SSLError(SSLError("bad handshake: Error([('SSL
routines', 'ssl3_read_bytes', 'tlsv1 alert unknown ca')]),),),)
```

Cause:

- Client connected using a cert that was signed by a CA unknown to the server

- Apache was unable to find the root CA cert that was used for signing the client side cert in its local store of trusted root CA certs
- Apache was also unable to find intermediate certs that could be used for creating a chain from the client side cert to one of the root CA certs

Check:

- Check that the CA used for signing the client side cert is set up to be trusted by Apache
- The signing CA should either be in a folder pointed to by *SSLCACertificatePath* or in a cert bundle file pointed to by *SSLCACertificateFile*
 - Typical location of these settings is *gm3-ssl.conf*
- For client side certs signed by DataONE, point *SSLCACertificateFile* to local copy of cert bundle:
 - Test environments: <https://repository.dataone.org/software/tools/trunk/ca/DataONETestCAChain.crt>
 - Production: <https://repository.dataone.org/software/tools/trunk/ca/DataONECAChain.crt>
- For self signed client side certs, copy the signing CA cert to the directory pointed to by *SSLCACertificatePath* then run the *c_rehash* command in that directory.
- If the signing CA cert is in the right location, but seems to be ignored, check that the symbolic links containing hash values for the CA certs are present. If necessary, create them with the *c_rehash* command
- If any intermediate certs are required in order to connect the client side cert with a root CA cert, check that they are present. They should be installed just like root CA certs
- If client is connecting to a DataONE environment, check that the connection is to the env for which the client side cert was issued
 - DataONE uses *urn_node_<NAME>* for production certs and *urn_node_mnTest<NAME>* for test certs. E.g., in *settings.py*:
 - `DATAONE_ROOT = d1_common.const.URL_DATAONE_ROOT -> CLIENT_CERT_PATH = 'urn_node_<NAME>'`
 - `DATAONE_ROOT = 'https://cn-stage.test.dataone.org/cn' -> CLIENT_CERT_PATH = 'urn_node_mnTest<NAME>'`
- Check that the root CA certs are valid and PEM encoded
 - View the cert files with `openssl x509 -in <cert_file.pem> -text -noout`

Certificate verify failed

```
SSLError: ("bad handshake: Error([('SSL routines', 'ssl3_get_server_certificate', 'certificate verify failed')],)", (SSLError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)'),))
```

Cause:

- Client received a certificate that the client was unable to validate

Check:

- Check that the server side cert was signed by a CA known to the client and has not expired
- Check that the system clock on the client system is correct

Operation timed out

```
ssl.SSLError: ('The write operation timed out',) ssl.SSLError: ('The read operation timed out',) SSLEr-  
ror: _ssl.c:495: The handshake operation timed out
```

Cause:

- Client timed out while waiting for response from server

Check:

- Increase the timeout on the client
- Consider using streaming HTTP requests and responses

GMN Setup Background Information

PyPI

GMN is distributed via [PyPI](#), the Python Package Index.

bashrc

The *gm*n distribution includes a *.bashrc* file that contains various convenient settings and aliases for the *gm*n user. Installing the *.bashrc* file is optional but highly recommended, as it provides a standardized environment for administrators.

A brief message outlining the available settings and aliases will be displayed on each login.

Apache

The *mod_ssl* module handles TLS/SSL connections for GMN and validates client side certificates. It is included in the *apache2-common* package.

The *mod_wsgi* module enables Apache to communicate with *Django* and GMN.

Postgres

GMN uses Postgres Peer authentication, which is default in Ubuntu.

With Peer authentication, a process belonging to a given user can access Postgres as long as a corresponding username has been set up in Postgres with the `createuser` command.

As part of the GMN install, an empty database is created in Postgres with the `createdb` command. This database is owned by the `postgres` user and Peer authenticated users have permission to create tables in it.

GMN is configured to use this database via the `DATABASES/NAME` setting in `settings.py`. Before starting GMN, a Django management command that creates the tables that are required by GMN is run as the *gm*n user, which causes the *gm*n user to become the owner of the tables.

Asynchronous processing

CNs may send various messages to MNs. These include replication requests and System Metadata update notifications. Such requests are queued by GMN and handled asynchronously.

The asynchronous processes are implemented as Django management commands that are launched at regular intervals by *cron*. The management commands examine the queues and process the requests.

The asynchronous processes connect to CNs and other MNs on behalf of your GMN instance. These connections are made over TLS/SSL and use the client side certificate stored in `/var/local/dataone/certs/client`.

Authentication and authorization

Authentication and authorization in DataONE is based on *X.509* (SSL) certificates.

GMN authenticates to incoming connections from *DataONE clients* and other parts of the DataONE infrastructure, such as *CNs* by providing a *server side certificate* during the SSL/TLS handshake.

By default, a stand-alone instance of GMN uses a non-trusted, self-signed, “snakeoil” server side certificate. This defers the purchase of a publicly trusted certificate from a 3rd party *CA* such as VeriSign or Thawte until the stand-alone instance is registered with DataONE.

A stand-alone instance that is not going to be registered with DataONE can use the non-trusted certificate indefinitely. Such a certificate is as secure as a publicly trusted certificate when used locally.

In addition to acting as servers in the DataONE infrastructure, Member Nodes also act as clients, initiating connections to other Nodes. When connecting to other Nodes, Member Nodes authenticate themselves in a process called *client side authentication*, in which a client side certificate is provided over an LTS/SSL connection.

Nodes that are registered with DataONE will only trust Member Node connections where a client side certificate issued by the DataONE *CA* is provided. However, a stand-alone instance of GMN will not connect to registered Member Nodes, so a non-trusted client side certificate can be used instead.

Misc

GMN translates incoming date-times to UTC and provides outgoing date-times in UTC. Because of this, it is convenient to run the server in UTC, so that server related timestamps, such as the ones in logs, match up with timestamps stored in the GMN database and provided in DataONE REST API responses.

4.4.2 Authentication and authorization

DataONE specifies a security model for Member Nodes. The model covers most aspects of how clients authenticate and which content they are authorized for. Some aspects are left open for Member Nodes to implement as best fits their requirements.

This section outlines the main aspects of how authentication and authorization is implemented in GMN and how to configure GMN and clients. In-depth coverage of these topics is provided in the [DataONE architecture documentation](#).

Authentication

In DataONE, authentication is the process of confirming the identity claimed by a person or system that connects to a node in order to call the node’s DataONE REST API methods.

A person or system can connect to a node without claiming an identity. This is done by connecting via HTTPS (or HTTP for Tier 1 nodes) without providing a [X.509](#) (SSL) *client side certificate*. In this case, the connection is granted access only to publicly available APIs and objects.

To claim an identity, the person or system connects with a client side certificate. The certificate must be issued by a [CA](#) that is trusted by the node. A DataONE compliant serialization of the certificate *DN* becomes the primary DataONE subject. The certificate can also contain an X.509 v3 extension that hold additional DataONE subjects in the form of equivalent identities and group memberships.

When a node first receives an incoming connection with a client side certificate, it does basic validation of the certificate itself. This includes checking that the certificate was issued by a trusted CA, that it has not expired, has not been revoked and has not been tampered with. After the certificate has passed these tests, the node extracts the primary subject and any other subjects from the certificate. These become the authenticated subjects for the connection and authentication is complete.

GMN uses Apache for performing the basic validation of the certificate. If a certificate is provided but is invalid, Apache will return an error to the client, indicating why the certificate failed validation and will then terminate the connection.

Authorization

In DataONE, authorization is the process of confirming that an authenticated subject has access to a DataONE REST API method or object. Authorization happens each time a REST API call is made. When the call is made, the node will look at the list of authenticated subjects that is associated with the connection through which the call was made. If the list of authenticated subjects does not include a subject to which access to the REST API method has been granted, authorization is denied and GMN returns a 401 NotAuthorized exception to the client.

Permissions for create, update and delete

DataONE does not specify how Member Nodes should control access to the APIs that allow users to create, update and delete contents on the node. GMN controls the access to these APIs with a whitelist. If a subject that is not in the whitelist attempts to call, for instance, `MNStorage.create()`, GMN will return a DataONE exception such as this (formatted for readability):

```
Exception: NotAuthorized
errorCode: 401
detailCode: 0
description:
  Access allowed only for subjects with Create/Update/Delete permission.
  Session subjects:
    authenticatedUser (equivalent),
    public (equivalent),
    CN=First Last,O=Google,C=US,DC=cilogon,DC=org (primary)
```

This means that the connection was made with a certificate in which the subject was `CN=First Last,O=Google,C=US,DC=cilogon,DC=org` and that this subject was not in GMN's whitelist for create, update and delete.

To create a whitelist with this subject, first create a file, for instance, `whitelist.txt`. The most convenient location for this file is in the `gmh` folder:

```
sudo -Hu gmh bash -c '
  cd /var/local/dataone/gmh_venv_py3/lib/python3.6/site-packages/gmh
  nano whitelist.txt
'
```

In this file, add a line with an exact copy of the subject string marked as `primary` in the `NotAuthorized` exception (`CN=First Last,O=Google,C=US,DC=cilogon,DC=org` in this case).

Blank lines and lines starting with “#” are ignored in the whitelist file, allowing comments. The remaining lines must each contain a separate subject.

Then, add the entries in the whitelist text file to GMN’s database with the following command:

```
$ python manage.py set_whitelist whitelist.txt
```

Any existing subjects in the database are cleared before adding the subjects from the whitelist file. So subjects can be added or removed from the whitelist by adding or removing them in the file and then synchronizing with the database by running the command above.

Creating authenticated connections to your Node

To create an authenticated connection to your Node, you must connect over HTTPS and provide a client side certificate. For a stand-alone node, only the local CA is trusted by default. So only certificates issued by this CA can be used. If the GMN instance is joined to DataONE, it is set up to also trust certificates issued by CILogon and DataONE.

In addition, the certificate must be for a subject that has the rights required for performing the operation(s) the client intends to perform after connecting. For instance, GMN requires that the subject used in connections that create content on the Node validate against an internal *whitelist*.

For automated tasks, certificates issued by the local CA are preferred. DataONE does not issue certificates for clients, so cannot be used for this purpose and certificates issued by CILogon are secured by having a time limit of 18 hours, making them unsuitable for automated tasks.

When running as a regular user, the local CA must be used for a stand-alone instance. The local CA can also be used for a public instance but CILogon is a more secure choice due to the 18 hour expiration time.

Authenticating without a certificate

In a stand-alone testing environment, where network access to the GMN instance is strictly limited, it is possible to simply add `public` to the *whitelist for create, update and delete*. Because the public subject is assigned to all connections, this allows access to create, update and delete objects on the node without any authentication.

Thus, this mode allows modifying node contents when connecting entirely without a certificate. It also lets GMN be set up for access over regular HTTP.

Authenticating with any trusted certificate

Connections that are made with any certificate that is trusted by GMN are assigned the `authenticatedUser` subject. So, adding this subject to the *whitelist for create, update and delete* enables anyone that connects with a trusted certificate to alter content on the Node. This is highly insecure if the Node is set up to trust CILogon, as anyone can obtain a CILogon certificate through OpenID. However, it may be useful if the node exposes only public objects and so, does not need to trust CILogon.

4.4.3 Using GMN

After *GMN* has been set up according the setup instructions, it exposes the complete REST interface that DataONE has defined for Member Nodes. Currently, the easiest way to interact with GMN is to use the DataONE Command Line Client (CLI). The CLI is installed automatically with GMN and can be started by typing “dataone”. The CLI can also be scripted to perform tasks such as bulk object creations to populate an instance of GMN with science data.

See `/gmn/setup/ubuntu/gmn/testing`, `/gmn/setup/centos/12-testing` and the [DataONE Command Line Interface \(CLI\) documentation](#) for more information about how to use the CLI.

If more comprehensive access to the Node is required, DataONE provides libraries in Java and Python that simplify the process of interacting with DataONE Member Nodes and Coordinating Nodes. The libraries can be used as foundations for custom applications that use the DataONE infrastructure.

Contents:

General

Populating your new Node

The DataONE Client Library for Python includes an example on how to iterate over a set of files, create data packages (resource maps) for them, and upload them to a Member Node. DataONE provides similar libraries for Java.

The CLI can also be scripted to perform tasks such as bulk object creations to populate a MN with Science Data.

Vendor specific extensions

GMN implements a set of extensions that enhance the functionality of GMN. Most of these are designed to help with debugging and profiling and they are described in another section.

Remote URL

The Remote URL vendor specific extension enables GMN to be used for exposing science data that is already available through another web based service without having to create another copy of that data.

In the regular `MNStorage.create()` and `MNStorage.update()` REST calls, the bytes of the science objects are provided, and the MN manages the storage of the objects. When using the Remote URL extension, the bytes of the objects are not provided and instead, a HTTP or HTTPS URL to the original location of the data is provided. GMN then manages all aspects of exposing the science data except for the actual storage of the bytes of the exposed object.

When the object is downloaded from GMN, GMN streams the object from its original location in the background.

This extension is activated by adding an HTTP header to the REST call for `MNStorage.create()` and `MNStorage.update()`. The name of the header is `VENDOR_GMN_REMOTE_URL` and the value is the HTTP or HTTPS URL that references the object in the remote location. When this header is added, the section of the POST body that contains the object bytes is ignored, but it must still be included to form a valid REST call. It is typically set to contain a zero byte object.

4.4.4 Bulk Import

Copy from a running MN:

- Science objects
- Permissions
- Subjects
- Event logs

This function can be used for setting up a new instance of GMN to take over for an existing MN. The import has been tested with other versions of GMN but should also work with other node stacks.

This command can be run before the new GMN instance has been set up to run as a web service, so the procedure does not require two web servers to run at the same time.

The new GMN instance can be installed on the same server as the source MN or on a different server.

When replacing an older GMN instance by installing a new instance on the same server, the general procedure is:

- Install the new GMN instance using the regular install procedure, with the following exceptions:
 - Install the new GMN instance to a different virtualenv by using a different virtualenv directory name for the new instance.
 - Skip all Apache related steps.
 - Skip all certificate related steps.
 - Use a separate database for the new instance by modifying the database name in settings.py and using the new name when initializing the database.
- Manually copy individual settings from settings.py / settings_site.py of the old instance to settings.py of the new instance. The new instance will be using the same settings as the old one, including client side certificate paths and science object storage root.
- To make sure that all the settings were correctly copied from the old instance, Generate a Node document in the new instance and compare it with the version registered in the DataONE environment for the old instance.


```
$ manage.py node view
```
- If a certificate is specified with the `-cert-pub` and (optionally) `-cert-key` command line switches, GMN will connect to the source MN using that certificate. Else, GMN will connect using its client side certificate, if one has been set up via `CLIENT_CERT_PATH` and `CLIENT_CERT_PRIVATE_KEY_PATH` in settings.py. Else, GMN connects to the source MN without using a certificate.

The `-public` switch causes GMN to ignore any available certificates and connect as the public user. This is useful if the source MN has only public objects and a certificate that would be accepted by the source MN is not available.

After the certificate provided by GMN is accepted by the source MN, GMN is authenticated on the source MN for the subject(s) contained in the certificate. If no certificate was provided, only objects and APIs that are available to the public user are accessible

The importer depends on the source MN `listObjects()` API being accessible to one or more of the authenticated subjects, or to the public subject if no certificate was provided. Also, for MNs that filter results from `listObjects()`, only objects that are both returned by `listObjects()` and are readable by one or more of the authenticated subjects(s) can be imported.

If the source MN is a GMN instance, `PUBLIC_OBJECT_LIST` in its settings.py controls access to `listObjects()`. For regular authenticated subjects, results returned by `listObjects()` are filtered to include only objects for which one or more of the subjects have read or access or better. Subjects that are whitelisted for create, update and delete access in GMN, and subjects authenticated as Coordinating Nodes, have unfiltered access to `listObjects()`. See settings.py for more information.

Member Nodes keep an event log, where operations on objects, such as reads, are stored together with associated details. After completed object import, the importer will attempt to import the events for all successfully imported objects. For event logs, `getLogRecords()` provides functionality equivalent to what `listObjects` provides for objects, with the same access control related restrictions.

If the source MN is a GMN instance, `PUBLIC_LOG_RECORDS` in settings.py controls access to `getLogRecords()` and is equivalent to `PUBLIC_OBJECT_LIST`.

- Start the import. Since the new instance has been set up to use the same object storage location as the old instance, the importer will automatically detect that the object bytes are already present on disk and skip the `get()` calls for the objects.

```
$ manage.py import
```

- Temporarily start the new MN with connect to it and check that all data is showing as expected.

```
$ manage.py runserver
```

- Stop the source MN by stopping Apache.
- Modify the VirtualHost file for the source MN, e.g., `/etc/apache2/sites-available/gmn2-ssl.conf`, to point to the new instance, e.g., by changing `gmn_venv` to the new virtualenv location.
- Start the new instance by starting Apache.
- From the point of view of the CNs and other nodes in the environment, the node will not have changed, as it will be serving the same objects as before, so no further processing or synchronization is required.

If the new instance is set up on a different server, extra steps likely to be required include:

- Modify the BaseURL in `settings.py`
- Update the Node registration

```
$ manage.py node update
```

Notes:

- Any replica requests that have been accepted but not yet processed by the source MN will not be completed. However, requests expire and are automatically reissued by the CN after a certain amount of time, so this should be handled gracefully by the system.
- Any changes on the source MN that occur during the import may or may not be included in the import. To avoid issues such as lost objects, events and system metadata updates, it may be necessary to restrict access to the source MN during the transition.

4.4.5 Maintenance

Notes on maintaining a GMN instance.

Upgrading and updating GMN

As we are often improving stability and performance, as well as adding features to GMN and the DataONE software stack, we recommend that GMN nodes are regularly updated to the latest release. Updating GMN causes the underlying software stack to be updated as well.

GMN is currently in its 3rd major revision, designated by 3.x.x version numbers. Within 3.x.x versions, automated updates are provided, allowing the MN administrator to update to the latest release by running a few simple commands.

Nodes on the earlier GMN 1.x.x and 2.x.x versions require a full upgrade. Upgrades are more complex than updates, and are performed manually by a DataONE developer.

Finding your GMN version

To check which version you are running, enter GMN's Home page. The Home page is located at `BaseURL/home`. For instance, if your BaseURL is `https://my.node.org/mn`, your home page is at `https://my.node.org/mn/home`.

Based on your version number, see the applicable section below.

Upgrading GMN 1.x.x and 2.x.x to latest release

Note: This method is applicable only for nodes running the earlier GMN 1.x.x and 2.x.x versions. For nodes running GMN 3.x.x, see *Updating GMN 3.x.x to the latest release*.

Due to the complexity of upgrading from earlier GMN 1.x.x and 2.x.x versions, one of our developers, Roger Dahl, is available to perform the upgrade via an ssh connection directly on the GMN server. In order to accomplish this, it is preferable if an account can be set up on the GMN server with public key based authentication. The public key is available at:

<https://repository.dataone.org/documents/Management/Users/dahl/sshpublickey/>

- The account will need “sudo” access
- The account name can be selected according to the organization’s policies. If no specific policies are in place, “dahl” can be used

Opening temporary ssh access to the GMN server

Often, ssh access to the GMN server is not available from external networks. For use in such cases, DataONE provides a simple service that allows the MN administrator to open temporary ssh access directly to the GMN server by running the following command from a shell on the GMN server:

```
$ sudo ssh -p 46579 -vNTR 46578:localhost:22 dlr@73.228.47.109
```

- Password: data!one#
- Press Ctrl-C to terminate access

This opens a temporary secure reverse tunnel that allows access from a single IP address, belonging to the developer. Access remains available until the command is stopped by pressing Ctrl-C. This also immediately terminates any active ssh connections.

Typically, no modifications, such as opening firewalls, are required in order to establish the reverse tunnel. However, depending on the organization’s security policies, the MN administrator may require approval from IT staff.

Note that we are able to publish the password here, as connecting to the service by itself only allows a second reverse connection to be established. The second connection is restricted by IP address, encrypted and secured by an RSA key.

Updating GMN 3.x.x to the latest release

Note: This update method is applicable only for nodes already running earlier versions of GMN 3.x.x. For nodes running earlier versions of GMN, see *Upgrading GMN 1.x.x and 2.x.x to latest release*.

Log into the GMN server and perform the following commands:

```
sudo -Hu gmn bash -c '  
  pip install --upgrade dataone.gmn  
  manage.py migrate  
'
```

```
sudo -H bash -c '  
  sudo service apache2 restart  
'
```

Updating the Node document

Note: If these paths are not correct for the version of GMN currently running on your node, please upgrade to the latest release first.

The Node document contains information specific to a Node, such as the Member Node description and contact information.

Make the desired updates to the Node information by modifying the GMN `settings.py` file.

Publish the updated Node document:

```
sudo -Hu gmn bash -c '  
  manage.py node update  
'
```

4.4.6 Optimizing GMN performance

Postgres database

Increasing the memory available to Postgres for such things as sorting can dramatically include performance, as Postgres will use the disk as temporary storage if there is not enough memory. In Ubuntu 18.04 with Postgres 10, the default is 4MB. To increase the value, edit `work_mem` in `postgresql.conf`. E.g.,:

```
sudo editor /etc/postgresql/10/main/postgresql.conf
```

- Increase `work_mem` from 4MB to 32MB.

The `MNRead.listObjects()` and `MNCore.getLogRecords()` API issue ordered, sliced and filtered select statements. The base tables for these operations should be clustered (physically ordered) by the default sort order. Unfortunately, Django does not do this automatically. To cluster the base table for `MNRead.listObjects()`:

Find the names of the indexes, by running the GMN `manage.py` as the `gmn` user:

```
$ sudo -Hu gmn  
$ ./manage.py dbshell  
> \d app_scienceobject
```

Search for the combined index name (`modified_timestamp, id`)

Specify clustering on the index:

```
> cluster app_scienceobject using <index name, e.g., app_science_modifie_76ef91_idx>;
```

To cluster the base table for `MNCore.getLogRecords()`, repeat the procedure with `app_eventlog` and `(timestamp, id)`.

When successfully completed, `\d app_scienceobject / app_eventlog` will display the `CLUSTERED` keyword next to the clustered indexes.

Notes:

- Clustering on a large database can take a long time, and queries are not accepted during the process.
- Postgres will not automatically keep the table clustered. Instead, the table must be clustered whenever sufficient changes have been accumulated.
- To keep the table clustered for longer, adjust the fill factor.
- Use cron to schedule automatic clustering. Note that the tables are locked while the clustering operation runs.
- Search the web for Postgres “analyze” and “vacuum” for more information.

Profiling

When GMN is in debug mode (DEBUG is set to True in the GMN settings.py file), the following profiling functionality is available.

SQL query profiling

All REST calls accept a *vendor specific extensions* called `VENDOR_PROFILE_SQL`. When this parameter is provided, the normal output from the call is suppressed and a text document containing SQL query profiling information is returned instead. The document lists all the SQL queries that were used for filling the request together with execution times.

Note: If a REST call returns an exception, the exception is also suppressed.

Python profiling

All REST calls accept a *vendor specific extensions* called `VENDOR_PROFILE_PYTHON`. When this parameter is provided, the normal output from the call is suppressed and a text document containing Python script profiling information is returned instead. The document includes information such as the name and location, number of calls and cumulative execution times for the longest running functions.

Note: Only the view functions are covered. In particular, `response_handler`, where the SQL queries are executed, is not covered.

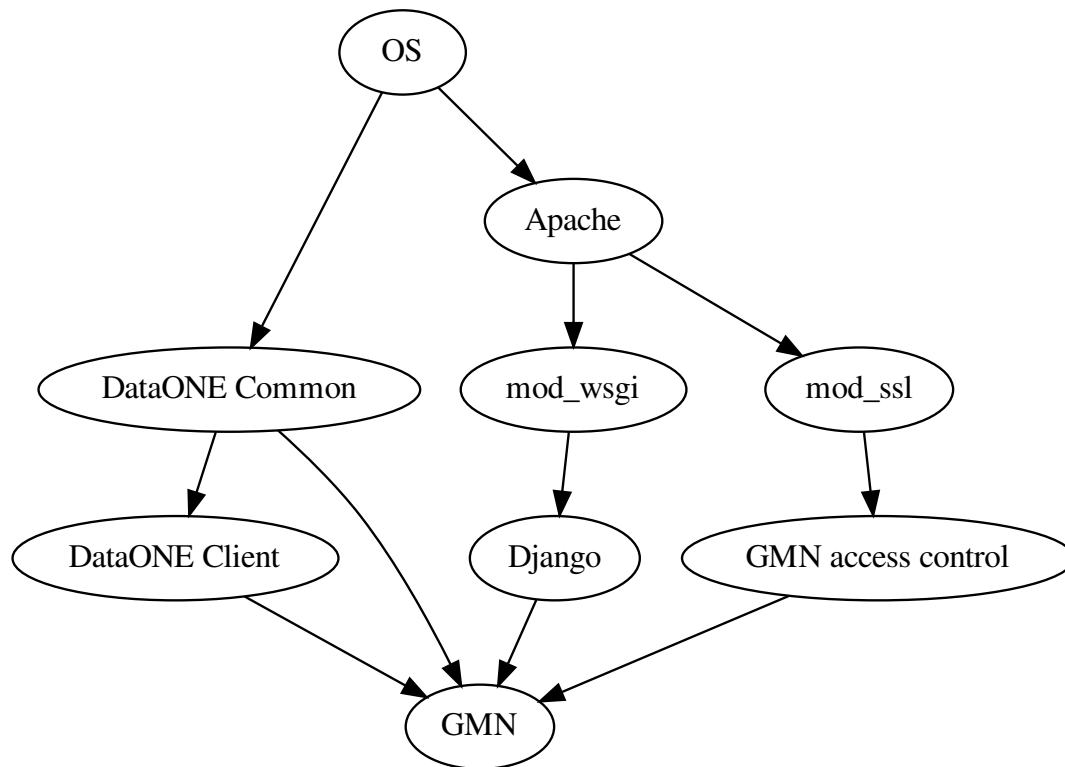
4.4.7 Implementation

Implementation notes intended for developers.

Contents:

Implementation

DataONE GMN is a web app implemented in *Python* based on the *Django* web app framework. Django is a *WSGI* compliant application. It is served by *Apache* via *mod_wsgi*. The DataONE infrastructure uses *SSL* and *X.509 certificates* for security and certificate validation is handled for GMN by *mod_ssl*.



Locking and concurrency

Locking and concurrency in GMN is based on Django's implementation of implicit database transactions, enabled by setting `ATOMIC_REQUESTS` to `True` in the database connection setup.

Django wraps each HTTP request in an implicit transaction. The transaction is rolled back if the request does not complete successfully. Upon a successfully completed request, the transaction is committed, thus making all modifications that the request made to the database visible simultaneously, bringing the database directly from one valid state to the next.

Transactions are also used in read-only requests as they hide any transitions between valid states that may happen during the processing of multiple database transactions during a single request.

Testing and debugging

In production, GMN is always served over SSL with an optional *client side certificate*. For testing and debugging, GMN must be served over HTTP because the Django development server does not support HTTPS. In that scenario,

it is not possible for the client to provide a certificate.

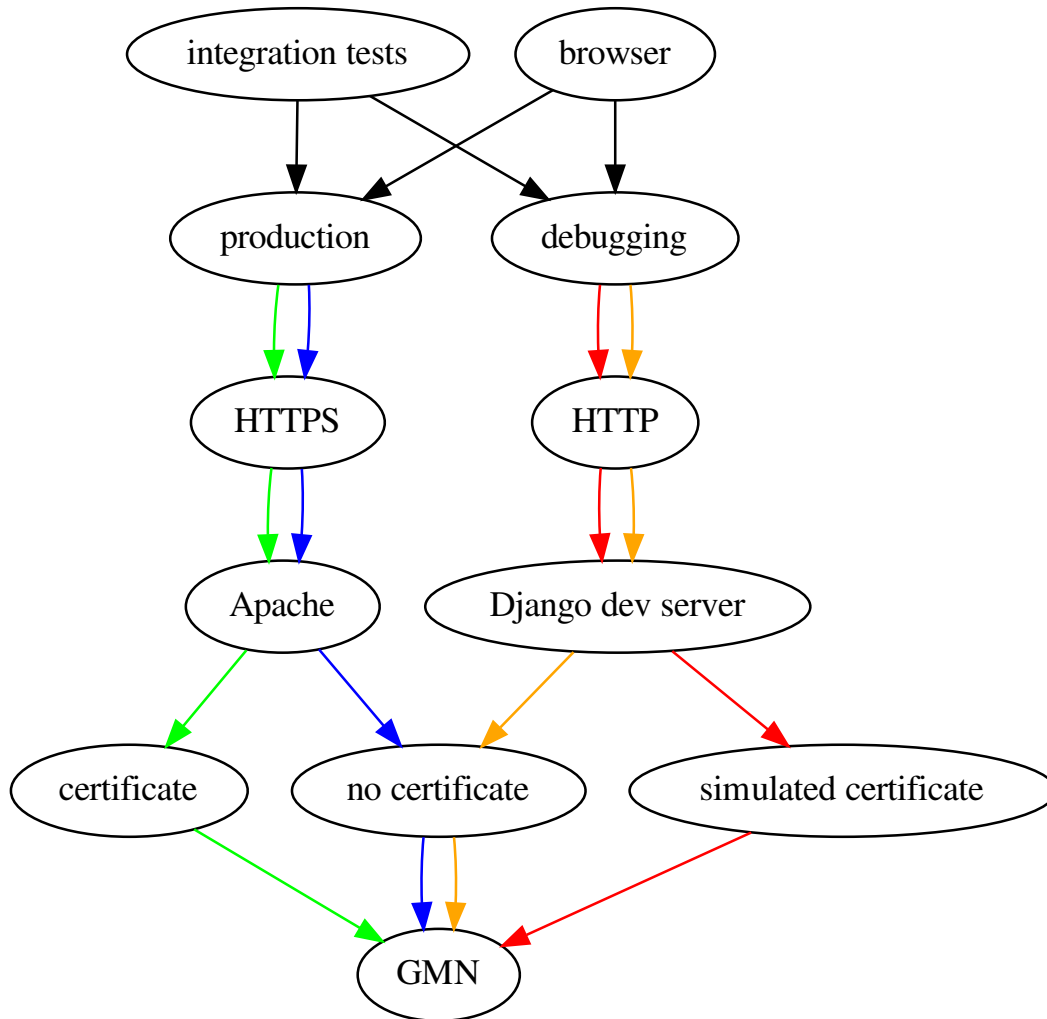


Figure: The various scenarios that GMN can be served under.

- Green: Production with client side certificate. Apache will reject the connection if the certificate is not valid, and GMN will not see the connection attempt. The certificate must be signed by *CILogon*.
- Blue: Production without a client side certificate. Apache accepts the connection. GMN falls back to the default Public session.
- Red: Testing and debugging with simulated certificate. This path is used by the integration tests. Debugging is supported. Because HTTP is used, no certificate can be provided. Instead, a valid certificate is simulated by using a Vendor Specific Extension to pass in a session.

Because Apache rejects connections with invalid certificates in production, there is no need to simulate a scenario where an invalid certificate is passed to GMN.

This path is only available when GMN is running in debug mode.

- Orange: Testing and debugging without a certificate. Same as the testing path with simulated certificate except that it simulates a connection without a session by not providing a session in the Vendor Specific Extension. This requires GMN to fall back to the default Public session.
- From the point of view of GMN, there are 3 types of connections:
 1. Connection with valid certificate
 2. Connection without certificate (accepted, fall back to Public)
 3. Connection with simulated certificate (accepted only in debug mode)

Integration tests against production instance

<TODO: Add instructions on how to run the integration tests with a valid certificate signed by CILogon>

Integration tests against debug instance

The integration tests are by default set up to assume that the GMN instance they connect to is in debug mode and they should all pass without any additional configuration.

Browser testing against production instance

In some cases, it's convenient to test GMN via a browser though only the GET based REST calls are conveniently reproducible from a browser. These instructions focus on [Firefox](#).

GMN will authenticate with a *server side certificate* signed by CILogon. Set the browser up to accept this certificate by adding the CILogon CA certificates to the browser's trusted CA store:

- Open the Certificate Manager (Edit | Preferences | Advanced | Encryption | View Certificates)
- Import new CA (Authorities | Import)
- Browse to `/var/local/dataone/ca/cilogon-basic.pem`
- Select "Trust this CA to identify web sites."

Repeat with the `cilogon-openid.pem` and `cilogon-silver.pem` certificates.

The functionality accessible by the Public principal through GET based REST calls can now be tested.

To test functionality accessible only to authenticated users, the browser must be set up to provide a valid certificate signed by CILogon.

<TODO: Add instructions on how to obtain a certificate from CILogon and install it in Firefox>

Browser testing against debug instance

In debug mode, GMN supports providing a simulated certificate via *vendor specific extensions*. In this mode, the session object that a certificate would normally contain is passed to GMN via a custom HTTP header. To enable Firefox to provide the header, install a Firefox extension such as [Modify Headers](#).

<TODO: Add instructions on how to use the Modify Headers extension to add a simulated certificate>

Uploading test objects

The `create()` call accept a *vendor specific extensions* called `VENDOR_TEST_OBJECT`. When this parameter is provided, the system metadata for the object is accepted without any information being added or overwritten by the MN.

Testing the replication processing

The DataONE Test Utilities for Python includes the Replication Tester (RepTest), a Python app that performs basic testing of the replication functionality of a Tier 4 MN. This describes how to set GMN up for testing with RepTest.

RepTest takes on the roles of the CN and another MN. So, for the test to be successful, GMN must be set up to accept RepTest both as a CN and another MN during the transfer of the object being replicated. GMN must also be set up to call back to RepTest during replication instead of to the root CN.

Without certificates

The simplest way to test the replication functionality is to turn off access control for objects and the replication API methods in GMN. Of course, this means that the access control is not tested.

<TODO: Describe how to set this up>

Changing root CN

RepTest needs to be set up as the root CN for the GMN instance being tested. This is done by modifying `DATAONE_ROOT` in `settings.py` to point to RepTest. E.g., if RepTest is running on the same machine as GMN:

```
DATAONE_ROOT = 'http://localhost:8181'
```

The port and network interface on which RepTest listens is configurable.

Background

The first time that GMN handles a request after startup, it will call `CNCore.listNodes()` on the root CN in the environment in which it is set up to find information about the other nodes in the environment. GMN will perform this call at even intervals to refresh its cache of the information.

When RepTest is set to be the root CN, RepTest receives this initial call. RepTest responds with a customized list of nodes holding only a CN and a MN. These nodes both point back to RepTest, thus setting the GMN instance up to accept calls from RepTest as if they originate from a CN. In addition, the replication related calls that GMN makes to the CN and MN replication counterpart become directed to RepTest, which uses them for orchestrating the replication process and checking that the MN is performing the replication correctly.

If GMN is not set up to use RepTest as a root CN, RepTest will abort testing with a authentication related exception. For instance, if RepTest calls `MNRead.getReplica()`, the exception may look like the following:

```
d1_common.types.exceptions.NotAuthorized: name: NotAuthorized
errorCode: 401
detailCode: 0
description:
  A CN has not authorized the target MN, "public" to create a replica of "anterior1.
  ↪ jpg".
```

(continues on next page)

(continued from previous page)

```
Exception received from the CN:
name: NotAuthorized
errorCode: 401
detailCode: 4871
description: There is no Member Node registered with a node subject matching public
nodeId: urn:node:mnDevGMN
```

This somewhat confusing error message is a NotAuthorized exception from GMN with a description field that contains the exception that was received from the CN which, in this case, is also a NotAuthorized exception.

The exception is raised because GMN called a real CN to get authorization for a call to `MNRead.getReplica()`. Since the replication was initiated by RepTest and not the real CN, the real CN rejects the request.

4.4.8 API

d1_gmn package

Subpackages

d1_gmn.app package

Subpackages

d1_gmn.app.management package

Subpackages

d1_gmn.app.management.commands package

GMN management commands.

Subpackages

d1_gmn.app.management.commands.util package

Utilities for GMN management commands.

Submodules

d1_gmn.app.management.commands.util.standard_args module

```
d1_gmn.app.management.commands.util.standard_args.add_arguments(parser,
                                                                    doc_str,
                                                                    add_base_url=True)
```

Add standard arguments for DataONE utilities to a command line parser.

d1_gmn.app.management.commands.util.util module**Submodules****d1_gmn.app.management.commands.audit-sync module****d1_gmn.app.management.commands.cert module****d1_gmn.app.management.commands.diag-resource-map module****d1_gmn.app.management.commands.diag-revision module****d1_gmn.app.management.commands.diag module****d1_gmn.app.management.commands.event module**

View and manipulate object event records.

```
class d1_gmn.app.management.commands.event.Command(*args, **kwargs)
```

Bases: `django.core.management.base.BaseCommand`

```
add_arguments(parser)
```

Entry point for subclassed commands to add custom arguments.

```
handle(*args, **opt)
```

The actual logic of the command. Subclasses must implement this method.

d1_gmn.app.management.commands.get-log module**d1_gmn.app.management.commands.get-object module****d1_gmn.app.management.commands.import-multiprocess module****d1_gmn.app.management.commands.import module****d1_gmn.app.management.commands.jwt module****d1_gmn.app.management.commands.logininfo_async module**

Async ObjectList Iterator.

Fast retrieval of logEntry from a DataONE Node.

```
class d1_gmn.app.management.commands.logininfo_async.EventLogIteratorAsync(async_client,
```

```
    page_size=1000,
```

```
    get_log_records_args_dict=None,
```

```
    max_concurrent_d1_rest_requests=10)
```

Bases: `object`

```
itr()
```

d1_gmn.app.management.commands.iter-check-redirect module

d1_gmn.app.management.commands.iter-fix module

d1_gmn.app.management.commands.node module

d1_gmn.app.management.commands.objectlist_async module

Async ObjectList Iterator.

Fast retrieval of ObjectInfo from a DataONE Node.

```
class d1_gmn.app.management.commands.objectlist_async.ObjectListIteratorAsync (async_client,  
                                     page_size=1000,  
                                     list_objects_args_  
                                     max_concurrent_a  
  
    Bases: object  
    itr()
```

d1_gmn.app.management.commands.process_refresh_queue module

d1_gmn.app.management.commands.process_replication_queue module

d1_gmn.app.management.commands.sciobj module

d1_gmn.app.management.commands.sync module

d1_gmn.app.management.commands.whitelist module

d1_gmn.app.middleware package

Submodules

d1_gmn.app.middleware.detail_codes module

Get Detail Code for DataONE exception based on REST call path.

```
class d1_gmn.app.middleware.detail_codes.DataoneExceptionToDetailCode  
    Bases: object  
    detail_code (request, exception)
```

d1_gmn.app.middleware.exception_handler module

Exception handler middleware.

Catch, log and serialize exceptions that are raised when processing a request.

Implements the system for returning information about exceptional conditions (errors) as described in Raised by MN and CN APIs <http://mule1.dataone.org/ArchitectureDocs/html>

An MN is required to always return a DataONE exception on errors. When running in production mode (settings.DEBUG = False and settings.DEBUG_GMN = False), GMN complies with this by wrapping any unhandled internal exception in a DataONE exception.

When running in Django debug mode (settings.DEBUG = True), non-DataONE exceptions are returned as Django HTML exception pages.

Responses to HEAD requests can not contain a body, so the exception is serialized to a set of HTTP headers for HEAD requests.

```
class d1_gmn.app.middleware.exception_handler.ExceptionHandler (next_in_chain_func)
    Bases: object

    process_exception (request, e)
```

d1_gmn.app.middleware.profiling_handler module

d1_gmn.app.middleware.request_handler module

d1_gmn.app.middleware.response_handler module

d1_gmn.app.middleware.session_cert module

Extract subjects from a DataONE X.509 v3 certificate.

If a certificate was provided, it has been validated by Apache before being passed to GMN. So it is known to signed by a trusted CA and to be unexpired.

A user can connect without providing a certificate (and so, without providing a session). This limits the user's access to data that is publicly available.

A user can connect with a certificate that does not contain a list of equivalent identities and group memberships (no SubjectInfo). This limits the user's access to data that is publicly available and that is available directly to that user (as designated in the Subject DN).

```
d1_gmn.app.middleware.session_cert.get_subjects (request)
    Get all subjects in the certificate.
```

- Returns: primary_str (primary subject), equivalent_set (equivalent identities, groups and group memberships)
- The primary subject is the certificate subject DN, serialized to a DataONE compliant subject string.

```
d1_gmn.app.middleware.session_cert.get_authenticated_subjects (cert_pem)
    Return primary subject and set of equivalents authenticated by certificate.
```

- cert_pem can be str or bytes

d1_gmn.app.middleware.session_jwt module

Validate Java Web Token (JWT) and extract subject.

```
d1_gmn.app.middleware.session_jwt.validate_jwt_and_get_subject_list (request)
```

d1_gmn.app.middleware.view_handler module

d1_gmn.app.migrations package

Submodules

d1_gmn.app.migrations.0001_initial module

```
class d1_gmn.app.migrations.0001_initial.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    initial = True

    dependencies = []

    operations = [<CreateModel name='BlockedMemberNode', fields=[('id', <django.db.models.
```

d1_gmn.app.migrations.0002_scienceobject_filename module

```
class d1_gmn.app.migrations.0002_scienceobject_filename.Migration(name,
                                                                    app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0001_initial')]

    operations = [<AddField model_name='scienceobject', name='filename', field=<django.db.
```

d1_gmn.app.migrations.0003_mediatype_mediatypeproperty module

```
class d1_gmn.app.migrations.0003_mediatype_mediatypeproperty.Migration(name,
                                                                    app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0002_scienceobject_filename')]

    operations = [<CreateModel name='MediaType', fields=[('id', <django.db.models.fields.A
```

d1_gmn.app.migrations.0004_auto_20170523_0137 module

```
class d1_gmn.app.migrations.0004_auto_20170523_0137.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0003_mediatype_mediatypeproperty')]

    operations = [<CreateModel name='SeriesIdToHeadPersistentId', fields=[('id', <django.db
```

d1_gmn.app.migrations.0005_auto_20170527_1554 module

```
class d1_gmn.app.migrations.0005_auto_20170527_1554.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0004_auto_20170523_0137')]

    operations = [<AlterField model_name='localreplica', name='pid', field=<django.db.mode
```

d1_gmn.app.migrations.0006_chainidtopersistentid module

```
class d1_gmn.app.migrations.0006_chainidtopersistentid.Migration(name,
                                                                app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0005_auto_20170527_1554')]
    operations = [<CreateModel name='ChainIdToPersistentId', fields=[('author_id', <django.db.models.fields.AutoField>)]>]
```

d1_gmn.app.migrations.0007_delete_chainidtopersistentid module

```
class d1_gmn.app.migrations.0007_delete_chainidtopersistentid.Migration(name,
                                                                app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0006_chainidtopersistentid')]
    operations = [<DeleteModel name='ChainIdToPersistentId'>]
```

d1_gmn.app.migrations.0008_chainidtopersistentid module

```
class d1_gmn.app.migrations.0008_chainidtopersistentid.Migration(name,
                                                                app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0007_delete_chainidtopersistentid')]
    operations = [<CreateModel name='ChainIdToPersistentId', fields=[('id', <django.db.models.fields.AutoField>)]>]
```

d1_gmn.app.migrations.0009_auto_20170603_0546 module

```
class d1_gmn.app.migrations.0009_auto_20170603_0546.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0008_chainidtopersistentid')]
    operations = [<CreateModel name='ChainIdToSeriesID', fields=[('id', <django.db.models.fields.AutoField>)]>]
```

d1_gmn.app.migrations.0010_auto_20170805_0107 module

```
class d1_gmn.app.migrations.0010_auto_20170805_0107.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0009_auto_20170603_0546')]
    operations = [<CreateModel name='Chain', fields=[('id', <django.db.models.fields.AutoField>)]>]
```

d1_gmn.app.migrations.0011_resourcemap_resourcemapmember module

```
class d1_gmn.app.migrations.0011_resourcemap_resourcemapmember.Migration(name,
                                                                app_label)
    Bases: django.db.migrations.migration.Migration
    dependencies = [('app', '0010_auto_20170805_0107')]
```

```
operations = [<CreateModel name='ResourceMap', fields=[('id', <django.db.models.fields
```

d1_gmn.app.migrations.0012_auto_20170821_2129 module

```
class d1_gmn.app.migrations.0012_auto_20170821_2129.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0011_resourcemap_resourcemapmember')]
    operations = [<AlterModelOptions name='eventlog', options={'ordering': ['-timestamp',
```

d1_gmn.app.migrations.0013_auto_20171017_0724 module

```
class d1_gmn.app.migrations.0013_auto_20171017_0724.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0012_auto_20170821_2129')]
    operations = [<AlterModelOptions name='scienceobject', options={'ordering': ['-modifi
```

d1_gmn.app.migrations.0014_auto_20171017_0725 module

```
class d1_gmn.app.migrations.0014_auto_20171017_0725.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0013_auto_20171017_0724')]
    operations = [<AlterField model_name='chain', name='sid', field=<django.db.models.fiel
```

d1_gmn.app.migrations.0015_auto_20171029_0246 module

```
class d1_gmn.app.migrations.0015_auto_20171029_0246.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0014_auto_20171017_0725')]
    operations = [<RenameField model_name='resourcemapmember', old_name='ResourceMap', new
```

d1_gmn.app.migrations.0016_auto_20180518_1539 module

```
class d1_gmn.app.migrations.0016_auto_20180518_1539.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration

    dependencies = [('app', '0015_auto_20171029_0246')]
    operations = [<AlterModelOptions name='eventlog', options={'ordering': ['timestamp',
```

d1_gmn.app.migrations.0017_auto_20180518_1540 module

d1_gmn.app.migrations.0018_auto_20180901_0115 module

```
class d1_gmn.app.migrations.0018_auto_20180901_0115.Migration(name, app_label)
    Bases: django.db.migrations.migration.Migration
```

```
dependencies = [('app', '0017_auto_20180518_1540')]
```

```
operations = [<AddIndex model_name='eventlog', index=<Index: fields='timestamp, id'>>
```

d1_gmn.app.migrations.0019_auto_20190418_1512 module

```
class d1_gmn.app.migrations.0019_auto_20190418_1512.Migration(name, app_label)
```

```
Bases: django.db.migrations.migration.Migration
```

```
dependencies = [('app', '0018_auto_20180901_0115')]
```

```
operations = [<AlterModelOptions name='eventlog', options={}>, <AlterModelOptions name=
```

d1_gmn.app.views package

Submodules

d1_gmn.app.views.assert_db module

Asserts used in views.

These directly return a DataONE Exception to the client if a test condition is not true.

```
d1_gmn.app.views.assert_db.is_valid_pid_for_create(did)
```

Assert that *did* can be used as a PID for creating a new object with MNStorage.create() or MNStorage.update().

```
d1_gmn.app.views.assert_db.is_valid_pid_to_be_updated(did)
```

Assert that *did* is the PID of an object that can be updated (obsoleted) with MNStorage.update()

```
d1_gmn.app.views.assert_db.is_did(did)
```

```
d1_gmn.app.views.assert_db.is_existing_object(did)
```

```
d1_gmn.app.views.assert_db.is_sid(did)
```

```
d1_gmn.app.views.assert_db.is_bool_param(param_name, bool_val)
```

```
d1_gmn.app.views.assert_db.is_in_revision_chain(pid)
```

```
d1_gmn.app.views.assert_db.is_not_obsoleted(pid)
```

```
d1_gmn.app.views.assert_db.post_has_mime_parts(request, parts)
```

Validate that a MMP POST contains all required sections.

Parameters

- **request** – Django Request
- **parts** – [(part_type, part_name), ...]

Returns None or raises exception.

Where information is stored in the request: part_type header: request.META['HTTP_<UPPER CASE NAME>'] part_type file: request.FILES['<name>'] part_type field: request.POST['<name>']

```
d1_gmn.app.views.assert_db.date_is_utc(date_time)
```

```
d1_gmn.app.views.assert_db.url_is_http_or_https(url)
```

```
d1_gmn.app.views.assert_db.url_is_retrievable(url)
```

```
d1_gmn.app.views.assert_db.is_not_replica(pid)
```

`d1_gmn.app.views.assert_db.is_not_archived(pid)`

d1_gmn.app.views.assert_sysmeta module

Asserts used in views.

These directly return a DataONE Exception to the client if a test condition is not true.

`d1_gmn.app.views.assert_sysmeta.sanity(request, sysmeta_pyxb)`

Check that sysmeta_pyxb is suitable for creating a new object and matches the uploaded sciobj bytes.

`d1_gmn.app.views.assert_sysmeta.matches_url_pid(sysmeta_pyxb, url_pid)`

`d1_gmn.app.views.assert_sysmeta.has_matching_modified_timestamp(new_sysmeta_pyxb)`

`d1_gmn.app.views.assert_sysmeta.obsoletes_not_specified(sysmeta_pyxb)`

`d1_gmn.app.views.assert_sysmeta.obsoletes_matches_pid_if_specified(sysmeta_pyxb,
old_pid)`

`d1_gmn.app.views.assert_sysmeta.is_valid_sid_for_new_standalone(sysmeta_pyxb)`

Assert that any SID in sysmeta_pyxb can be assigned to a new standalone object.

`d1_gmn.app.views.assert_sysmeta.is_valid_sid_for_chain(pid, sid)`

Assert that sid can be assigned to the single object pid or to the chain to which pid belongs.

- If the chain does not have a SID, the new SID must be previously unused.
- If the chain already has a SID, the new SID must match the existing SID.

`d1_gmn.app.views.assert_sysmeta.calculate_checksum(request, checksum_calculator)`

d1_gmn.app.views.create module

d1_gmn.app.views.decorators module

d1_gmn.app.views.external module

d1_gmn.app.views.get_package module

d1_gmn.app.views.gmn module

d1_gmn.app.views.headers module

d1_gmn.app.views.internal module

d1_gmn.app.views.slice module

Handle slicing / paging of multi-page result set.

`d1_gmn.app.views.slice.add_slice_filter(request, query, total_int)`

`d1_gmn.app.views.slice.cache_add_last_in_slice(request, query, start_int, total_int,
sort_field_list)`

d1_gmn.app.views.util module

Submodules

d1_gmn.app.auth module

d1_gmn.app.context_processors module

Django template context processors.

Before rendering a template, Django calls context processors as set up in `settings_default.TEMPLATE_CONTEXT_PROCESSORS`. The context processors are functions that are expected to return a dict which will be merged into the environment available to the template.

`d1_gmn.app.context_processors.global_settings(request)`
Expose some values from settings.py to templates.

d1_gmn.app.db_filter module

d1_gmn.app.delete module

d1_gmn.app.did module

Utilities for working with SIDs and PIDs.

`d1_gmn.app.did.is_valid_pid_for_create(did)`

Return True if `did` is the PID of an object that can be created with `MNStorage.create()` or `MNStorage.update()`.

To be valid for `create()` and `update()`, the DID:

- Must not be the PID of an object that exists on this MN
- Must not be a known SID known to this MN
- Must not have been accepted for replication by this MN.
- Must not be referenced as `obsoletes` or `obsoletedBy` in an object that exists on this MN

In addition, if the DID exists in a resource map:

- If `RESOURCE_MAP_CREATE = 'reserve'`:
 - The DataONE subject that is making the call must have `write` or `changePermission` on the resource map.

`d1_gmn.app.did.is_valid_sid_for_new_standalone(did)`

Return True if `did` can be assigned to a new standalone object.

`d1_gmn.app.did.is_valid_pid_to_be_updated(did)`

Return True if `did` is the PID of an object that can be updated (obsoleted) with `MNStorage.update()`

`d1_gmn.app.did.is_valid_sid_for_chain(pid, sid)`

Return True if `sid` can be assigned to the single object `pid` or to the chain to which `pid` belongs.

- If the chain does not have a SID, the new SID must be previously unused.
- If the chain already has a SID, the new SID must match the existing SID.

All known PIDs are associated with a chain.

Preconditions: - `pid` is verified to exist. E.g., with

`d1_gmn.app.views.asserts.is_existing_object()`.

- `sid` is None or verified to be a SID

`d1_gmn.app.did.get_did_by_foreign_key(did_foreign_key)`

Return the DID referenced by a ForeignKey or OneToOneField to IdNamespace.

Return None if ForeignKey or OneToOneField is NULL.

This is used instead of “`did_foreign_key.*.did`” on ForeignKeys and OneToOneFields that allow NULL (`null=True` in the model).

`d1_gmn.app.did.is_existing_object(did)`

Return True if PID is for an object for which science bytes are stored locally.

This excludes SIDs and PIDs for unprocessed replica requests, remote or non-existing revisions of local replicas and objects aggregated in Resource Maps.

`d1_gmn.app.did.is_sid(did)`

`d1_gmn.app.did.is_obsoleted(did)`

Return True if `did` is the PID of an object that has been obsoleted.

`d1_gmn.app.did.is_resource_map_db(pid)`

`d1_gmn.app.did.is_resource_map_member(pid)`

`d1_gmn.app.did.classify_identifier(did)`

Return a text fragment classifying the `did`

Return <UNKNOWN> if the DID could not be classified. This should not normally happen and may indicate that the DID was orphaned in the database.

`d1_gmn.app.did.get_or_create_did(id_str)`

`d1_gmn.app.did.is_in_revision_chain(sciobj_model)`

`d1_gmn.app.did.is_archived(pid)`

`d1_gmn.app.did.is_local_replica(pid)`

Includes unprocessed replication requests.

`d1_gmn.app.did.is_unprocessed_local_replica(pid)`

Is local replica with status “queued”.

`d1_gmn.app.did.is_revision_chain_placeholder(pid)`

For replicas, the PIDs referenced in revision chains are reserved for use by other replicas.

d1_gmn.app.event_log module

d1_gmn.app.gmn module

App performing filesystem setup and basic sanity checks on configuration values in `settings.py` before GMN starts servicing requests.

Django loads apps into the Application Registry in the order specified in `settings.INSTALLED_APPS`. This app must be set to load before the main GMN app by listing it above the main app in `settings.INSTALLED_APPS`.

```

class d1_gmn.app.gmn.Startup(app_name, app_module)
    Bases: django.apps.config AppConfig

    name = 'd1_gmn.app'

    ready()
        Called once per Django process instance.

        If the filesystem setup fails or if an error is found in settings.py,
        django.core.exceptions.ImproperlyConfigured is raised, causing Django not to launch the main GMN app.

    raise_config_error(setting_name, cur_val, exp_type, valid_str=None, is_none_allowed=False)

```

d1_gmn.app.local_replica module

d1_gmn.app.model_util module

Database model utilities.

These are in a separate module because module classes can only be referenced in an active Django context. More general utilities can be used without an active context.

Importing this module outside of Django context raises `django.core.exceptions.AppRegistryNotReady`: Apps aren't loaded yet.

```

d1_gmn.app.model_util.get_sci_model(pid)
d1_gmn.app.model_util.get_pids_for_all_locally_stored_objects()
d1_gmn.app.model_util.delete_unused_subjects()
    Delete any unused subjects from the database.

```

This is not strictly required as any unused subjects will automatically be reused if needed in the future.

d1_gmn.app.models module

d1_gmn.app.node module

Generate Node document based on the current settings for GMN.

```

d1_gmn.app.node.get_pretty_xml(api_major_int=2)
d1_gmn.app.node.get_xml(api_major_int)
d1_gmn.app.node.get_pyxb(api_major_int=2)

```

d1_gmn.app.node_registry module

Node Registry cache.

- Retrieve, hold and update a cache of the Node Registry for the DataONE environment in which this MN is registered.
- Query the Node Registry.

```

d1_gmn.app.node_registry.get_cn_subjects()
d1_gmn.app.node_registry.set_empty_cn_subjects_cache()

```

```
d1_gmn.app.node_registry.set_cn_subjects_for_environment()  
d1_gmn.app.node_registry.get_cn_subjects_string()  
d1_gmn.app.node_registry.get_cn_subjects_from_dataone_root()  
d1_gmn.app.node_registry.download_node_registry()  
d1_gmn.app.node_registry.create_root_cn_client()
```

d1_gmn.app.object_format_cache module

d1_gmn.app.proxy module

Proxy mode.

```
d1_gmn.app.proxy.get_sciobj_iter_remote(url)  
d1_gmn.app.proxy.is_proxy_url(url)
```

d1_gmn.app.psycpg_adapter module

Psycpg Postgres adapter for Python.

Registers custom adapters with Psycpg, which simplify reading and writing custom DataONE PyXB types to/from database models.

```
d1_gmn.app.psycpg_adapter.adapt_pyxb_binding(client)
```

d1_gmn.app.resource_map module

Utilities for manipulating resource maps.

```
d1_gmn.app.resource_map.assert_map_is_valid_for_create(resource_map)  
d1_gmn.app.resource_map.is_sciobj_valid_for_create()  
    When RESOURCE_MAP_CREATE == 'reserve', objects that are created and that are also aggregated in one  
    or more resource maps can only be created by a DataONE subject that has write or changePermission on the  
    resource map.  
d1_gmn.app.resource_map.create_or_update_db(sysmeta_pyxb)  
d1_gmn.app.resource_map.get_resource_map_from_sciobj(pid)  
d1_gmn.app.resource_map.create_or_update(map_pid, resource_map)  
d1_gmn.app.resource_map.get_resource_map_members(pid)  
    pid is the PID of a Resource Map or the PID of a member of a Resource Map.  
d1_gmn.app.resource_map.get_resource_map_members_by_map(map_pid)  
d1_gmn.app.resource_map.get_resource_map_members_by_member(member_pid)  
d1_gmn.app.resource_map.is_resource_map_sysmeta_pyxb(sysmeta_pyxb)  
d1_gmn.app.resource_map.parse_resource_map_from_str(resource_map_xml)
```

d1_gmn.app.revision module

Utilities for manipulating revision chains in the database.

`d1_gmn.app.revision.create_or_update_chain(pid, sid, obsoletes_pid, obsoleted_by_pid)`

`d1_gmn.app.revision.delete_chain(pid)`

`d1_gmn.app.revision.cut_from_chain(sciobj_model)`

Remove an object from a revision chain.

The object can be at any location in the chain, including the head or tail.

Preconditions: - The object with the pid is verified to exist and to be a member of an revision chain. E.g., with:

`d1_gmn.app.views.asserts.is_existing_object(pid)` `d1_gmn.app.views.asserts.is_in_revision_chain(pid)`

Postconditions: - The given object is a standalone object with empty `obsoletes`, `obsoletedBy` and `seriesId` fields.

- The previously adjacent objects in the chain are adjusted to close any gap that was created or remove dangling reference at the head or tail.
- If the object was the last object in the chain and the chain has a SID, the SID reference is shifted over to the new last object in the chain.

`d1_gmn.app.revision.get_all_pid_by_sid(sid)`

`d1_gmn.app.revision.resolve_sid(sid)`

Get the PID to which the `sid` currently maps.

Preconditions: - `sid` is verified to exist. E.g., with `d1_gmn.app.views.asserts.is_sid()`.

`d1_gmn.app.revision.get_sid_by_pid(pid)`

Given the `pid` of the object in a chain, return the SID for the chain.

Return None if there is no SID for the chain. This operation is also valid for standalone objects which may or may not have a SID.

This is the reverse of `resolve`.

All known PIDs are associated with a chain.

Preconditions: - `pid` is verified to exist. E.g., with

`d1_gmn.app.views.asserts.is_existing_object()`.

`d1_gmn.app.revision.set_revision_links(sciobj_model, obsoletes_pid=None, obsoleted_by_pid=None)`

`d1_gmn.app.revision.is_obsoletes_pid(pid)`

Return True if `pid` is referenced in the `obsoletes` field of any object.

This will return True even if the PID is in the `obsoletes` field of an object that does not exist on the local MN, such as replica that is in an incomplete chain.

`d1_gmn.app.revision.is_obsoleted_by_pid(pid)`

Return True if `pid` is referenced in the `obsoletedBy` field of any object.

This will return True even if the PID is in the `obsoletes` field of an object that does not exist on the local MN, such as replica that is in an incomplete chain.

`d1_gmn.app.revision.is_revision(pid)`

Return True if `pid` is referenced in the `obsoletes` or `obsoletedBy` field of any object.

This will return True even if the PID is in the `obsoletes` field of an object that does not exist on the local MN, such as replica that is in an incomplete chain.

d1_gmn.app.scimeta module

Utilities for Science Metadata.

`d1_gmn.app.scimeta.assert_valid(sysmeta_pyxb, pid)`

Validate file at `{sciobj_path}` against schema selected via `formatId` and raise `InvalidRequest` if invalid.

Validation is only performed when:

- SciMeta validation is enabled
- and Object size is below size limit for validation
- and `formatId` designates object as a Science Metadata object which is recognized and parsed by DataONE CNs
- and XML Schema (XSD) files for `formatId` are present on local system

d1_gmn.app.sciobj_store module

Manage the filesystem tree in which science object bytes are stored.

- Because it may be inefficient to store millions of files in a single folder and because such a folder is hard to deal with when performing backups and maintenance, GMN stores the objects in a folder hierarchy of 256 folders, each holding 256 folders, for a total of 65536 folders. The location in the hierarchy for a given object is based on its PID.
- Folders are created as required in the hierarchy.

`d1_gmn.app.sciobj_store.open_sciobj_file_by_pid_ctx(pid, write=False)`

Open the file containing the Science Object bytes of `pid` in the default location within the tree of the local SciObj store.

If `write` is True, the file is opened for writing and any missing directories are created. Return the file handle and `file_url` with the file location in a suitable form for storing in the DB.

If nothing was written to the file, it is deleted.

`d1_gmn.app.sciobj_store.open_sciobj_file_by_path_ctx(abs_path, write=False)`

Open the file containing the Science Object bytes at the custom location `abs_path` in the local filesystem.

If `write` is True, the file is opened for writing and any missing directories are created. Return the file handle and `file_url` with the file location in a suitable form for storing in the DB.

If nothing was written to the file, delete it.

`d1_gmn.app.sciobj_store.get_sciobj_iter_by_url(sciobj_url)`

`d1_gmn.app.sciobj_store.get_sciobj_byte_iterator_by_url(sciobj_url)`

`d1_gmn.app.sciobj_store.get_sciobj_iter_by_pid(pid)`

`d1_gmn.app.sciobj_store.open_sciobj_file_by_pid(pid, write=False)`

Open the file containing the Science Object bytes at the custom location `abs_path` in the local filesystem for read.

`dl_gmn.app.sciobj_store.open_sciobj_file_by_path(abs_path, write=False)`

Open a SciObj file for read or write. If opened for write, create any missing directories. For a SciObj stored in the default SciObj store, the path includes the PID hash based directory levels.

This is the only method in GMN that opens SciObj files, so can be modified to customize the SciObj storage locations and can be mocked for testing.

Note that when a SciObj is created by a client via `MNStorage.create()`, Django streams the SciObj bytes to a temporary file or memory location as set by `FILE_UPLOAD_TEMP_DIR` and related settings.

`dl_gmn.app.sciobj_store.get_rel_sciobj_file_path(pid)`

Get the relative local path to the file holding an object's bytes.

- The path is relative to `settings.OBJECT_STORE_PATH`
- There is a one-to-one mapping between pid and path
- The path is based on a SHA1 hash. It's now possible to craft SHA1 collisions, but it's so unlikely that we ignore it for now
- The path may or may not exist (yet).

`dl_gmn.app.sciobj_store.get_abs_sciobj_file_path_by_pid(pid)`

Get the absolute local path to the file holding an object's bytes.

- The path is to a location below `settings.OBJECT_STORE_PATH`
- There is a one-to-one mapping between pid and path
- The path is based on a SHA1 hash. It's now possible to craft SHA1 collisions, but it's so unlikely that we ignore it for now
- The path may or may not exist (yet).

`dl_gmn.app.sciobj_store.get_abs_sciobj_file_path_by_rel_path(rel_path)`

Get the absolute local path to the file holding an object's bytes.

- The path is to a location below `settings.OBJECT_STORE_PATH`
- There is a one-to-one mapping between pid and path
- The path is based on a SHA1 hash. It's now possible to craft SHA1 collisions, but it's so unlikely that we ignore it for now
- The path may or may not exist (yet).

`dl_gmn.app.sciobj_store.get_abs_sciobj_store_path()`

Get the absolute local path to the root of the default SciObj store.

- The path may or may not exist (yet).

`dl_gmn.app.sciobj_store.assert_sciobj_store_exists()`

`dl_gmn.app.sciobj_store.assert_sciobj_store_does_not_exist()`

`dl_gmn.app.sciobj_store.is_existing_store()`

`dl_gmn.app.sciobj_store.is_existing_sciobj_file(pid)`

`dl_gmn.app.sciobj_store.get_rel_sciobj_file_url_by_pid(pid)`

Get the URL that will be stored in the database for a SciObj that is saved in GMN's SciObj filesystem hierarchy below `settings.OBJECT_STORE_PATH`.

`dl_gmn.app.sciobj_store.get_abs_sciobj_file_url(abs_sciobj_file_path)`

Get the URL that will be stored in the database for a SciObj that is saved in a custom location outside of GMN's SciObj filesystem hierarchy.

`d1_gmn.app.sciobj_store.get_abs_sciobj_file_path_by_url(file_url)`

Get the absolute path to the file holding an object's bytes.

- `file_url` is an absolute or relative [file://](#) url as stored in the DB.

`d1_gmn.app.sciobj_store.get_gmn_version()`

`d1_gmn.app.sciobj_store.is_matching_version()`

`d1_gmn.app.sciobj_store.is_lower_version()`

`d1_gmn.app.sciobj_store.is_store_subdir(dir_path)`

`d1_gmn.app.sciobj_store.get_store_version()`

`d1_gmn.app.sciobj_store.save_store_version()`

`d1_gmn.app.sciobj_store.get_store_version_path()`

`d1_gmn.app.sciobj_store.create_store()`

`d1_gmn.app.sciobj_store.is_empty()`

`d1_gmn.app.sciobj_store.is_tmp()`

`d1_gmn.app.sciobj_store.assert_sciobj_store_version_match()`

`d1_gmn.app.sciobj_store.delete_sciobj(url_split, pid)`

d1_gmn.app.settings_default module

Default settings for GMN.

- These settings are only used as fallbacks in case the corresponding settings in `settings.py` are missing.
- This allows settings to be added without having to modify `settings.py` in existing deployments.
- Settings that are not described here are described in `settings_template.py`.

d1_gmn.app.sysmeta module

d1_gmn.app.sysmeta_extract module

d1_gmn.app.urls module

d1_gmn.app.util module

General utilities.

`d1_gmn.app.util.coerce_put_post(request)`

Django doesn't particularly understand REST. In case we send data over PUT, Django won't actually look at the data and load it. We need to twist its arm here.

The try/except abomination here is due to a bug in `mod_python`. This should fix it.

From `django-piston/piston/utils.py`

`d1_gmn.app.util.dump_stack()`

`d1_gmn.app.util.create_http_echo_response(request)`

`d1_gmn.app.util.get_static_path(rel_path)`

d1_gmn.tests package

Submodules

d1_gmn.tests.conftest module

```
d1_gmn.tests.conftest.gmn_client_v1(request)
d1_gmn.tests.conftest.gmn_client_v2(request)
d1_gmn.tests.conftest.gmn_client_v1_v2(request)
```

d1_gmn.tests.gmn_direct module

d1_gmn.tests.gmn_mock module

d1_gmn.tests.gmn_test_case module

d1_gmn.tests.gmn_test_client module

This module implements GMNTestClient, which extends d1_client.mnclient.MemberNodeClient with GMN specific test functionality.

The REST interfaces on GMN that provide this functionality are prefixed with “/diag/” and are only enabled when GMN runs in debug mode. The interfaces are not versioned, and so there is no version tag (such as “v1”) in the URL for these methods.

```
class d1_gmn.tests.gmn_test_client.GMNTestClient(*args, **kwargs)
    Bases: d1_client.mnclient.MemberNodeClient

    __init__(*args, **kwargs)
        Extend MemberNodeClient with GMN specific diagnostics wrappers.

        See d1baseclient.DataONEBaseClient for args.

    gm_n_vse_provide_subject(subject)
        GMN Vendor Specific Extension: Simulate subject.

    gm_n_vse_enable_sql_profiling()
        GMN Vendor Specific Extension: Enable SQL profiling.

    gm_n_vse_enable_python_profiling()
        GMN Vendor Specific Extension: Enable Python profiling.

    get_resource_path(path)
        Get path to test resources.

    get_replication_queue(headers=None)

    clear_replication_queue(headers=None)

    create_replication_queue(pid, sysmeta_pyxb, sourceNode, vendorSpecific=None)

    delete_all_access_policies(headers=None)

    get_access_policy(pid, headers=None)

    echo_session(headers=None)
```

whitelist_subject (*subject_str*, *headers=None*)
Add a subject to the whitelist.

create (*pid*, *obj*, *sysmeta_pyxb*, *vendorSpecific=None*)

slash (*arg1*, *arg2*, *arg3*, *headers=None*)

exception (*exception_type*, *headers=None*)

echo_request_object (*headers=None*)

echo_raw_post_data (*headers=None*)

delete_all_objects (*headers=None*)

get_setting (*setting*, *headers=None*)

delete_event_log (*headers=None*)
Delete event log for all objects.

inject_fictional_event_log (*event_log_csv*, *headers=None*)
Inject a fake event log.

concurrency_clear (*headers=None*)
Clear test key/vals.

concurrency_read_lock (*key*, *sleep_before*, *sleep_after*, *headers=None*)
PID read locking.

concurrency_write_lock (*key*, *val*, *sleep_before*, *sleep_after*, *headers=None*)
Test PID write locking.

concurrency_get_dictionary_id (*headers=None*)
Get dictionary ID.

d1_gmn.tests.gmn_wsgi module

class `d1_gmn.tests.gmn_wsgi.WSGIClient` (*django_settings_module=None*, *en-*
force_csrf_checks=True, **args*, ***kwargs*)
Bases: `django.core.handlers.base.BaseHandler`

Submit requests to GMN directly via its WSGI interface.

This client submits a raw WSGIRequest and returns the raw HttpResponse, both of which can contain streams.

The test client that is bundled with Django collapses all streams to `bytes`, both in requests and responses, so cannot be used for testing stream handling.

__init__ (*django_settings_module=None*, *enforce_csrf_checks=True*, **args*, ***kwargs*)
Initialize the client and corresponding Django service.

Parameters

- **django_settings_module** – str Override the DJANGO_SETTINGS_MODULE environment variable setting.

Requests will be sent to the Django service selected by the DJANGO_SETTINGS_MODULE environment variable. When running under pytest-django, the default value is set in “tox.ini”.

- **enforce_csrf_checks** – bool Disable Cross Site Request Forgery protection.

send_request (*wsgi_environ_dict*)
Send a WSGIRequest to a Django based service.

Parameters `wsgi_environ_dict` – dict WSGI environ dict from which to build a WSGIRequest.

Example for `MNStorage.create()`:

```
{ "REQUEST_METHOD": "POST", "PATH_INFO": "/v2/object", "CONTENT_TYPE": <content type of provided stream>, "CONTENT_LENGTH": <length of provided stream>, "REMOTE_ADDR": <IP or FQDN of client>, "wsgi.input": <
    MIME multipart stream providing the body of the request
>,
}
```

Returns `HttpResponse`

`d1_gmn.tests.mk_db_fixture` module

`d1_gmn.tests.mk_db_template` module

`d1_gmn.tests.test_api_is_authorized` module

`d1_gmn.tests.test_app_util` module

`d1_gmn.tests.test_archive_standalone` module

`d1_gmn.tests.test_authorization` module

`d1_gmn.tests.test_cert` module

`d1_gmn.tests.test_content_disposition` module

`d1_gmn.tests.test_cors` module

`d1_gmn.tests.test_create_and_get_standalone` module

`d1_gmn.tests.test_create_resource_map` module

`d1_gmn.tests.test_delete_standalone` module

`d1_gmn.tests.test_describe` module

`d1_gmn.tests.test_generate_identifier` module

`d1_gmn.tests.test_get_capabilities` module

`d1_gmn.tests.test_get_checksum` module

`d1_gmn.tests.test_get_log_records` module

d1_gmn.tests.test_get_log_records_auth module

d1_gmn.tests.test_get_package module

d1_gmn.tests.test_get_system_metadata module

d1_gmn.tests.test_list_objects module

d1_gmn.tests.test_mgmt_cert module

d1_gmn.tests.test_mgmt_diag_export_object_list module

d1_gmn.tests.test_mgmt_diag_repair_revision_chains module

d1_gmn.tests.test_mgmt_diag_update_sysmeta module

d1_gmn.tests.test_mgmt_import module

d1_gmn.tests.test_proxy_mode module

d1_gmn.tests.test_remote_replica module

d1_gmn.tests.test_replicate module

d1_gmn.tests.test_revision module

d1_gmn.tests.test_revision_archive module

d1_gmn.tests.test_revision_create_and_get module

d1_gmn.tests.test_revision_delete module

d1_gmn.tests.test_scimeta module

d1_gmn.tests.test_session_jwt module

d1_gmn.tests.test_settings module

d1_gmn.tests.test_slice module

d1_gmn.tests.test_streaming module

d1_gmn.tests.test_synchronization_failed module

d1_gmn.tests.test_sysmeta_extract module

d1_gmn.tests.test_system_metadata_changed module

d1_gmn.tests.test_timezone module

d1_gmn.tests.test_trusted_client_overrides module

d1_gmn.tests.test_update_system_metadata module

d1_gmn.tests.test_update_with_sid module

d1_gmn.tests.test_update_without_sid module

d1_gmn.tests.test_web_ui module

Submodules

d1_gmn.manage module

d1_gmn.serve-static-files module

d1_gmn.settings module

d1_gmn.settings_template module

Global settings for GMN.

d1_gmn.settings_test module

Test and debug settings for GMN.

- These settings are in effect when GMN is called through unit tests.

d1_gmn.version module

d1_gmn.wsgi module

4.5 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.6 DataONE Common Library for Python

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

The *DataONE Common Library for Python* is a component of the DataONE Investigator Toolkit (ITK). It forms the foundation on which higher level components in the DataONE Python stack are built. It provides functionality commonly needed by clients, servers and other applications that interact with the *DataONE* infrastructure, including:

- Serializing, deserializing, validating and type conversions for the DataONE XML types
- Parsing and generating X.509 v3 certificates with DataONE extension
- Parsing and generating OAI-ORE Resource Maps as used by DataONE
- Utilities for working with XML documents, URLs, date-times, etc, in the context of DataONE

Contents:

4.6.1 Installing DataONE Common Library for Python

DataONE Common Library for Python is distributed via PyPI, the Python Package Index.

Pip or another package manager such as apt may be used to install dependencies.

Note that versions available through package managers such as apt tend to lag significantly behind the latest versions, so it is recommended that Pip is used to manage dependencies. In order to avoid potential conflicts with system installed libraries, it is further recommended that a Virtual Environment or user installs of the dependencies are employed.

4.6.2 Windows

1. If you do not already have a working 32-bit Python 3.6 environment, download the latest 32-bit Python 3.6 Windows installer from <http://www.python.org/download/> and install it.
2. In Control Panel | Classic View | System | Advanced | Environment Variables, add ;C:\Python27;C:\Python27\Scripts to the end of the Path.
3. Install pip:

```
> python -c "import urllib2; exec(urllib2.urlopen('https://bitbucket.org/pypa/
↪setuptools/raw/bootstrap/ez_setup.py').read())"
> easy_install pip
```

4. Open a Command Prompt.
5. Install the DataONE Common Library for Python and dependencies:

```
> pip install dataone.common
```

4.6.3 Linux

1. Install pip (Python package installer):

```
$ sudo apt install --yes python-pip; sudo pip install pip --upgrade;
```

2. Install the DataONE Common Library for Python and dependencies:

```
$ sudo pip install dataone.common
```

4.6.4 Development

To set up a virtual environment:

```
pip install virtualenv
virtualenv dataone_python
source dataone_python/bin/activate
pip install -U iso8601
pip install -U pyxb
pip install -U requests
```

Or as a user specific installation:

```
pip install --user -U iso8601
pip install --user -U pyxb
pip install --user -U requests
```

4.6.5 Unit Tests

This library is shipped with unit tests that verify correct operation. It is recommended that these are executed after installation.

4.6.6 Updating the library

To update your copy of the library to the latest version available on PyPI, run `pip install` with the `--upgrade` option:

```
$ sudo pip install --upgrade dataone.common
```

It may also be necessary to regenerate the DataONE type bindings after an update. See [Types_](#) for more information.

4.6.7 API

d1_common package

DataONE Common Library.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_common.cert package

This package contains certificate related functionality, such as functions for extracting DataONE subjects from PEM (base64) encoded X.509 v3 certificates and Java Web Tokens (JWTs) as used in DataONE.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

d1_common.cert.jwt module

JSON Web Token (JWT) parsing and validation.

- <http://self-issued.info/docs/draft-jones-json-web-token-01.html>

JWT representations:

- `bu64`: A URL safe flavor of Base64 used by JWTs
- `jwt_bu64`: A complete JWT consisting of three dot separated bu64 encoded parts: (header_bu64, payload_bu64, signature_bu64)
- `jwt_tup`: A complete JWT consisting of a tuple of 3 decoded (raw) parts: (header_str, payload_str, signature_str)

`d1_common.cert.jwt.get_subject_with_local_validation(jwt_bu64, cert_obj)`

Validate the JWT and return the subject it contains.

- The JWT is validated by checking that it was signed with a CN certificate.
- The returned subject can be trusted for authz and authn operations.
- Possible validation errors include:
 - A trusted (TLS/SSL) connection could not be made to the CN holding the signing certificate.
 - The JWT could not be decoded.
 - The JWT signature signature was invalid.
 - The JWT claim set contains invalid “Not Before” or “Expiration Time” claims.

Parameters

- **jwt_bu64** – bytes The JWT encoded using a a URL safe flavor of Base64.
- **cert_obj** – cryptography.Certificate Public certificate used for signing the JWT (typically the CN cert).

Returns

- On successful validation, the subject contained in the JWT is returned.
- If validation fails for any reason, errors are logged and None is returned.

`d1_common.cert.jwt.get_subject_with_remote_validation(jwt_bu64, base_url)`

Same as `get_subject_with_local_validation()` except that the signing certificate is automatically downloaded from the CN.

- Additional possible validations errors:
 - The certificate could not be retrieved from the root CN.

`d1_common.cert.jwt.get_subject_with_file_validation(jwt_bu64, cert_path)`

Same as `get_subject_with_local_validation()` except that the signing certificate is read from a local PEM file.

`d1_common.cert.jwt.get_subject_without_validation(jwt_bu64)`

Extract subject from the JWT without validating the JWT.

- The extracted subject cannot be trusted for authn or authz.

Parameters `jwt_bu64` – bytes JWT, encoded using a a URL safe flavor of Base64.

Returns The subject contained in the JWT.

Return type `str`

`dl_common.cert.jwt.get_bu64_tup(jwt_bu64)`

Split the Base64 encoded JWT to its 3 component sections.

Parameters `jwt_bu64` – bytes JWT, encoded using a a URL safe flavor of Base64.

Returns Component sections of the JWT.

Return type 3-tup of Base64

`dl_common.cert.jwt.get_jwt_tup(jwt_bu64)`

Split and decode the Base64 encoded JWT to its 3 component sections.

- Reverse of `get_jwt_bu64()`.

Parameters `jwt_bu64` – bytes JWT, encoded using a a URL safe flavor of Base64.

Returns Raw component sections of the JWT.

Return type 3-tup of bytes

`dl_common.cert.jwt.get_jwt_bu64(jwt_tup)`

Join and Base64 encode raw JWT component sections.

- Reverse of `get_jwt_tup()`.

Parameters `jwt_tup` – 3-tup of bytes Raw component sections of the JWT.

Returns

bytes JWT, encoded using a a URL safe flavor of Base64.

Return type `jwt_bu64`

`dl_common.cert.jwt.get_jwt_dict(jwt_bu64)`

Parse Base64 encoded JWT and return as a dict.

- JWTs contain a set of values serialized to a JSON dict. This decodes the JWT and returns it as a dict containing Unicode strings.
- In addition, a SHA1 hash is added to the dict for convenience.

Parameters `jwt_bu64` – bytes JWT, encoded using a a URL safe flavor of Base64.

Returns Values embedded in and derived from the JWT.

Return type `dict`

`dl_common.cert.jwt.validate_and_decode(jwt_bu64, cert_obj)`

Validate the JWT and return as a dict.

- JWTs contain a set of values serialized to a JSON dict. This decodes the JWT and returns it as a dict.

Parameters

- `jwt_bu64` – bytes The JWT encoded using a a URL safe flavor of Base64.

- **cert_obj** – cryptography.Certificate Public certificate used for signing the JWT (typically the CN cert).

Raises *JwtException* – If validation fails.

Returns Values embedded in the JWT.

Return type dict

`d1_common.cert.jwt.log_jwt_dict_info(log, msg_str, jwt_dict)`
Dump JWT to log.

Parameters

- **log** – Logger Logger to which to write the message.
- **msg_str** – str A message to write to the log before the JWT values.
- **jwt_dict** – dict JWT containing values to log.

Returns None

`d1_common.cert.jwt.log_jwt_bu64_info(log, msg_str, jwt_bu64)`
Dump JWT to log.

Parameters

- **log** – Logger Logger to which to write the message.
- **msg_str** – str A message to write to the log before the JWT values.
- **jwt_bu64** – bytes JWT, encoded using a a URL safe flavor of Base64.

Returns None

`d1_common.cert.jwt.ts_to_str(jwt_dict)`
Convert timestamps in JWT to human readable dates.

Parameters **jwt_dict** – dict JWT with some keys containing timestamps.

Returns Copy of input dict where timestamps have been replaced with human readable dates.

Return type dict

`d1_common.cert.jwt.ts_to_dt(jwt_dict)`
Convert timestamps in JWT to datetime objects.

Parameters **jwt_dict** – dict JWT with some keys containing timestamps.

Returns Copy of input dict where timestamps have been replaced with `datetime.datetime()` objects.

Return type dict

`d1_common.cert.jwt.encode_bu64(b)`
Encode bytes to a URL safe flavor of Base64 used by JWTs.

- Reverse of `decode_bu64()`.

Parameters **b** – bytes Bytes to Base64 encode.

Returns URL safe Base64 encoded version of input.

Return type bytes

`d1_common.cert.jwt.decode_bu64(b)`
Encode bytes to a URL safe flavor of Base64 used by JWTs.

- Reverse of `encode_bu64()`.

Parameters `b` – bytes URL safe Base64 encoded bytes to encode.

Returns Decoded bytes.

Return type bytes

exception `d1_common.cert.jwt.JwtException`

Bases: `Exception`

Exceptions raised directly by this module.

`d1_common.cert.subject_info` module

Utilities for handling the DataONE SubjectInfo type.

Overview of Access Control in DataONE

Access control in DataONE works much like traditional Access Control Lists (ACLs). Each science object is associated with an ACL. The ACL contains a list of subjects and an access level for each subject. The access levels are *read*, *write* and *changePermission*. Each access level implicitly grants access to the lower levels, so only the highest access level for a given subject needs to be specified in the ACL.

This module handles the information that will be used for creating a list of authenticated subjects that can be compared against an ACL in order to determine if a given subject is allowed to access the object at the requested level.

DataONE supports a system where subjects can be linked to equivalent identities as well as managed in groups. E.g., a group of subjects can be created and all the subjects in the group can be given access to an object by only listing the single group subject in the object's ACL.

A given subject can describe an actual identity, an equivalent subject or a group subject. Any type of subject can be used in any capacity. E.g., each subject in a group can be any type of subject including another group.

Since ACLs can also contain any combination of subjects for actual identities, equivalent subjects and groups subjects, a list of subjects that includes all subjects that are associated with an authenticated subject is required in order to determine if access should be granted.

Arbitrarily nested subjects must be supported. E.g., If subj-1 has been successfully authenticated, and subj-1 has an equivalent subject called equiv-1, and equiv-1 is in a group with subject group-1, all of those subjects (subj-1, equiv-1, and group-1), must be included in the list of associated subjects. That way, access is granted to the object regardless of which of them are authenticated directly in the ACL.

Notes

- It's important to separate the roles of groups in the ACL and groups in the SubjectInfo. Including a group subject in an ACL grants access to all subjects in that group. However, including a subject that is in a group, in the ACL, does not give access to the other subjects of the group or to the group itself. In other words, groups add access for their members, not the other way around.
- In terms of generating a list of equivalent subjects based on SubjectInfo, the one way transfer of access from groups to their subjects means that, when a subject is found to belong to a group, only the group subject is included in the list (after which it may chain to more equivalent identities, etc). The group members are not included.

- For deriving a list of indirectly authenticated subjects, the SubjectInfo contains a set of statements that establish subject types and relationships between subjects. There are 4 kinds of statements:
 - Subject is a person
 - Subject is an equivalent of another subject
 - Subject is a group
 - Subject is member of a group
- An equivalent subject can only be the equivalent for a person. The equivalence relationship is the only one that causes each side to be granted all the rights of the other side, and so allows the two subjects to be used interchangeably. The other relationships cause one side to be granted the rights of the other side, but not the other way around. E.g.: Designating a subject as a member of a group causes the subject to be granted the rights of the group, but does not cause the group to be granted the rights of the subject.

Authorization examples

Given SubjectInfo:

```
A = person subject A, authenticated by certificate
B = person subject B

C = equivalent to A
D = equivalent to B
E = equivalent to D

F = group with members G, H, B
J = group with members K, L, F
M = group with members E, N
N = group with members C, D
```

Given ACL containing: D

- D is equivalent to B
- B is a Person, but it's unauthenticated

Authorization: Denied

Given ACL containing: N

- N is a group with members C and D
- D is equivalent to B, but B is not authenticated
- C is equivalent to A, and A is authenticated

Authorization: Granted

Given ACL containing: F

- F leads to G, H, B
- G -> unknown

- H -> unknown
- B -> person subject, but not authenticated

Authorization: Denied

`dl_common.cert.subject_info.extract_subjects(subject_info_xml, primary_str)`

Extract a set of authenticated subjects from a DataONE SubjectInfo.

- See `subject_info_tree` for details.

Parameters

- **subject_info_xml** – str A SubjectInfo XML document.
- **primary_str** – str A DataONE subject, typically a DataONE compliant serialization of the DN of the DataONE X.509 v3 certificate extension from which the SubjectInfo was extracted.

The primary subject can be viewed as the root of a tree. Any subject in the SubjectInfo that is directly or indirectly connected to the root subject is included in the returned set of authenticated subjects.

Returns

Set of authenticated subjects. Will always include the primary subject.

- All subjects in the returned set are equivalent to `primary_str` for the purpose of access control for private science objects.
- If SubjectInfo does not contain all relevant records, it is still considered to be valid, but the authenticated set will be incomplete.
- Only the subject strings and relationships in SubjectInfo are used by this function. Other information about subjects, such as name and email address, is ignored.
- No attempt should be made to infer type of subject from the content of a subject string. Subject strings should be handled as random Unicode sequences, each of which may designate an person subject, an equivalent subject, or a group subject.
- To determine if an action is authorized, the returned set is checked against the `authorized_set` for a given object. If one or more subjects exist in both sets, the action is authorized. The check can be performed with high performance using a set union operation in Python or an inner join in Postgres.
- Subject types are only known and relevant while processing the SubjectInfo type.
- The type of each subject in the `authenticated_subjects` and `allowed_subjects` lists are unknown and irrelevant.

Return type set

Notes

Procedure:

The set of authenticated subjects is generated from the SubjectInfo and primary subject using the following procedure:

- Start with empty set of subjects
- Add `authenticatedUser`
- If `subject` is not in set of subjects:

- Add `subject`
- Iterate over Person records
- If `Person.subject` is `subject`:
- If `Person.verified` is present and set:
- Add “verifiedUser”
- Iterate over `Person.equivalentIdentity`:
- Recursively add those subjects
- Iterate over `Person.isMemberOf`
- Recursively add those subjects, but ONLY check Group subjects
- Iterate over Group records
- If any `Group.hasMember` is `subject`:
- Recursively add `Group.subject` (not group members)

Handling of various invalid `SubjectInfo` and corner cases:

- `SubjectInfo` XML doc that is not well formed
- Return an exception that includes a useful error message with the line number of the issue
- `person.isMemberOf` and `group.hasMember` should always form pairs referencing each other.
- One side of the pair is missing
- Process the available side as normal
- `person.isMemberOf` subject references a person or equivalent instead of a group
- Only Group subjects are searched for `isMemberOf` references, so only the referenced Group subject is added to the list of authorized subjects
- Multiple Person or Group records conflict by using the same subject
- The records are handled as equivalents
- `person.isMemberOf` subject does not reference a known subject
- If the Person containing the dangling `isMemberOf` IS NOT connected with the authenticated subject, the whole record, including the `isMemberOf` subject is simply ignored
- If it IS connected with an authenticated subject, the `isMemberOf` subject is authenticated and recursive processing of the subject is skipped
- Circular references
- Handled by skipping recursive add for subjects that are already added
- See the unit tests for example `SubjectInfo` XML documents for each of these issues and the expected results.

`d1_common.cert.subject_info.deserialize_subject_info(subject_info_xml)`

Deserialize `SubjectInfo` XML doc to native object.

Parameters `subject_info_xml` – str `SubjectInfo` XML doc

Returns `SubjectInfo` PyXB object

`d1_common.cert.subject_info.gen_subject_info_tree(subject_info_pyxb, authn_subj, include_duplicates=False)`

Convert the flat, self referential lists in the `SubjectInfo` to a tree structure.

Parameters

- **subject_info_pyxb** – SubjectInfo PyXB object
- **authn_subj** – str The authenticated subject that becomes the root subject in the tree of subjects built from the SubjectInfo.

Only subjects that are authenticated by a direct or indirect connection to this subject are included in the tree.

- **include_duplicates** – Include branches of the tree that contain subjects that have already been included via other branches.

If the tree is intended for rendering, including the duplicates will provide a more complete view of the SubjectInfo.

Returns Tree of nodes holding information about subjects that are directly or indirectly connected to the authenticated subject in the root.

Return type *SubjectInfoNode*

```
class dl_common.cert.subject_info.SubjectInfoNode (label_str, type_str)
```

Bases: object

Tree representation of SubjectInfo.

In SubjectInfo, nested information is represented via self- referential lists. This class holds a recursive tree of nodes which simplifies processing of SubjectInfo for client apps.

```
SUBJECT_NODE_TAG = 'is_subject_node'
```

```
TYPE_NODE_TAG = 'is_type_node'
```

```
add_child (label_str, type_str)
```

Add a child node.

```
property node_gen
```

Generate all nodes for the tree rooted at this node.

Yields: SubjectInfoNode All nodes rooted at this node.

```
property leaf_node_gen
```

Generate all leaf nodes for the tree rooted at this node.

Yields: SubjectInfoNode All leaf nodes rooted at this node.

```
property parent_gen
```

Generate this node, then all parents from this node to the root.

Yields: SubjectInfoNode This node, then all parents from this node to the root.

```
get_path_str (sep='/', type_str=None)
```

Get path from root to this node.

Parameters

- **sep** – str One or more characters to insert between each element in the path. Defaults to “/” on Unix and “\” on Windows.
- **type_str** – SUBJECT_NODE_TAG, TYPE_NODE_TAG or None. If set, only include information from nodes of that type.

Returns String describing the path from the root to this node.

Return type str

get_leaf_node_path_list (*sep='/'*, *type_str=None*)

Get paths for all leaf nodes for the tree rooted at this node.

Parameters

- **sep** – str One or more characters to insert between each element in the path. Defaults to “/” on Unix and “\” on Windows.
- **type_str** – SUBJECT_NODE_TAG, TYPE_NODE_TAG or None. If set, only include information from nodes of that type.

Returns The paths to the leaf nodes for the tree rooted at this node.

Return type list of str

get_path_list (*type_str=None*)

Get list of the labels of the nodes leading up to this node from the root.

Parameters **type_str** – SUBJECT_NODE_TAG, TYPE_NODE_TAG or None. If set, only include information from nodes of that type.

Returns The labels of the nodes leading up to this node from the root.

Return type list of str

property is_leaf

Return True if this is a leaf node (has no children)

get_label_set (*type_str=None*)

Get a set of label_str for the tree rooted at this node.

Parameters **type_str** – SUBJECT_NODE_TAG, TYPE_NODE_TAG or None. If set, only include information from nodes of that type.

Returns The labels of the nodes leading up to this node from the root.

Return type set

get_subject_set ()

Get a set of subjects for the tree rooted at this node.

Returns: set: The subjects for the tree rooted at this node.

`d1_common.cert.subject_info.SubjectInfoTree`

alias of `d1_common.cert.subject_info.SubjectInfoNode`

d1_common.cert.subject_info_renderer module

d1_common.cert.subjects module

Extract subjects from a DataONE PEM (Base64) encoded X.509 v3 certificate.

The DataONE infrastructure uses X.509 v3 certificates to represent sessions. A session contains assertions about the identity of the caller. In particular, the session contains the primary identity, a list of equivalent identities and group memberships of the caller.

`d1_common.cert.subjects.extract_subjects` (*cert_pem*)

Extract subjects from a DataONE PEM (Base64) encoded X.509 v3 certificate.

Parameters **cert_pem** – str or bytes PEM (Base64) encoded X.509 v3 certificate

Returns

- The primary subject string, extracted from the certificate DN.

- A set of equivalent identities, group memberships and inferred symbolic subjects extracted from the SubjectInfo (if present.)
- All returned subjects are DataONE compliant serializations.
- A copy of the primary subject is always included in the set of equivalent identities.

Return type 2-tuple

d1_common.cert.view_subject_info module

d1_common.cert.x509 module

Utilities for processing X.509 v3 certificates.

`d1_common.cert.x509.extract_subjects(cert_pem)`

Extract primary subject and SubjectInfo from a DataONE PEM (Base64) encoded X.509 v3 certificate.

Parameters `cert_pem` – str or bytes PEM (Base64) encoded X.509 v3 certificate

Returns

- Primary subject (str) extracted from the certificate DN.
- SubjectInfo (XML str) if present (see the subject_info module for parsing)

Return type 2-tuple

`d1_common.cert.x509.extract_subject_from_dn(cert_obj)`

Serialize a DN to a DataONE subject string.

Parameters `cert_obj` – cryptography.Certificate

Returns Primary subject extracted from the certificate DN.

Return type str

The certificate DN (DistinguishedName) is a sequence of RDNs (RelativeDistinguishedName). Each RDN is a set of AVAs (AttributeValueAssertion / AttributeTypeAndValue). A DataONE subject is a plain string. As there is no single standard specifying how to create a string representation of a DN, DataONE selected one of the most common ways, which yield strings such as:

CN=Some Name A123,O=Some Organization,C=US,DC=Some Domain,DC=org

In particular, the sequence of RDNs is reversed. Attribute values are escaped, attribute type and value pairs are separated by “=”, and AVAs are joined together with “,”. If an RDN contains an unknown OID, the OID is serialized as a dotted string.

As all the information in the DN is preserved, it is not possible to create the same subject with two different DNs, and the DN can be recreated from the subject.

`d1_common.cert.x509.create_mn_dn(node_urn)`

Create a certificate DN suitable for use in Member Node client side certificates issued by DataONE, and thus in Certificate Signing Requests (CSR). The DN will be on the form:

DC=org, DC=dataone, CN=urn:node:<ID>

where <ID> typically is a short acronym for the name of the organization responsible for the Member Node.

The DN is formatted into a DataONE subject, which is used in authentication, authorization and event tracking.

Parameters `node_urn` – str Node URN. E.g.,

Production certificate: urn:node:XYZ. Test certificate
urn:node:mnTestXYZ.

Returns cryptography.x509.Name

`d1_common.cert.x509.create_simple_dn(common_name_str, domain_componet_list=None)`

Create a simple certificate DN suitable for use in testing and for generating self signed CA and other certificate.

DC=local, DC=dataone, CN=<common name>

Parameters

- **common_name_str** – The Common Name to use for the certificate. DataONE uses simple DNs without physical location information, so only the `common_name_str` (`CommonName`) needs to be specified.

For Member Node Client Side certificates or CSRs, `common_name_str` is the `node_id`, e.g., `urn:node:ABCD` for production, or `urn:node:mnTestABCD` for the test environments.

For a local CA, something like `localCA` may be used.

For a locally trusted client side certificate, something like `localClient` may be used.

- **domain_componet_list** – list Optionally set custom domain components.
- **fqdn_list** – list of str List of Fully Qualified Domain Names (FQDN) and/or IP addresses for which this certificate will provide authentication.

E.g.: `['my.membernode.org', '1.2.3.4']`

This is mainly useful for creating a self signed server side certificate or a CSR that will be submitted to a trusted CA, such as Verisign, for signing.

Returns cryptography.x509.Name

`d1_common.cert.x509.generate_csr(private_key_bytes, dn, fqdn_list=None)`

Generate a Certificate Signing Request (CSR).

Parameters

- **private_key_bytes** – bytes Private key with which the CSR will be signed.
- **dn** – cryptography.x509.Name The dn can be built by passing a list of cryptography.x509.NameAttribute to cryptography.x509.Name.

Simple DNs can be created with the `create_dn*` functions in this module.

- **fqdn_list** – list of str List of Fully Qualified Domain Names (FQDN) and/or IP addresses for which this certificate will provide authentication.

E.g.: `['my.membernode.org', '1.2.3.4']`

This is mainly useful for creating a self signed server side certificate or a CSR that will be submitted to a trusted CA, such as Verisign, for signing.

Returns cryptography.x509.CertificateSigningRequest

`d1_common.cert.x509.deserialize_pem(cert_pem)`

Deserialize PEM (Base64) encoded X.509 v3 certificate.

Parameters `cert_pem` – str or bytes PEM (Base64) encoded X.509 v3 certificate

Returns cryptography.Certificate

Return type `cert_obj`

`d1_common.cert.x509.deserialize_pem_file(cert_path)`

Deserialize PEM (Base64) encoded X.509 v3 certificate in file.

Parameters `cert_path` – str or bytes Path to PEM (Base64) encoded X.509 v3 certificate file

Returns `cryptography.Certificate`

Return type `cert_obj`

`d1_common.cert.x509.serialize_cert_to_pem(cert_obj)`

Serialize certificate to PEM.

The certificate can be also be a Certificate Signing Request (CSR).

Parameters `cert_obj` – `cryptography.Certificate`

Returns PEM encoded certificate

Return type `bytes`

`d1_common.cert.x509.extract_subject_info_extension(cert_obj)`

Extract DataONE SubjectInfo XML doc from certificate.

Certificates issued by DataONE may include an embedded XML doc containing additional information about the subject specified in the certificate DN. If present, the doc is stored as an extension with an OID specified by DataONE and formatted as specified in the DataONE SubjectInfo schema definition.

Parameters `cert_obj` – `cryptography.Certificate`

Returns SubjectInfo XML doc if present, else None

Return type `str`

`d1_common.cert.x509.download_as_der(base_url='https://cn.dataone.org/cn', out_sec=60.0)` *time-*

Download public certificate from a TLS/SSL web server as DER encoded `bytes`.

If the certificate is being downloaded in order to troubleshoot validation issues, the download itself may fail due to the validation issue that is being investigated. To work around such chicken-and-egg problems, temporarily wrap calls to the `download_*` functions with the `disable_cert_validation()` context manager (also in this module).

Parameters

- **base_url** – str A full URL to a DataONE service endpoint or a server hostname
- **timeout_sec** – int or float Timeout for the SSL socket operations

Returns The server's public certificate as DER encoded bytes.

Return type `bytes`

`d1_common.cert.x509.download_as_pem(base_url='https://cn.dataone.org/cn', out_sec=60.0)` *time-*

Download public certificate from a TLS/SSL web server as PEM encoded string.

Also see `download_as_der()`.

Parameters

- **base_url** – str A full URL to a DataONE service endpoint or a server hostname
- **timeout_sec** – int or float Timeout for the SSL socket operations

Returns The certificate as a PEM encoded string.

Return type `str`

`d1_common.cert.x509.download_as_obj` (*base_url='https://cn.dataone.org/cn',* *time-*
out_sec=60.0)

Download public certificate from a TLS/SSL web server as Certificate object.

Also see `download_as_der()`.

Parameters

- **base_url** – str A full URL to a DataONE service endpoint or a server hostname
- **timeout_sec** – int or float Timeout for the SSL socket operations

Returns `cryptography.Certificate`

`d1_common.cert.x509.decode_der` (*cert_der*)

Decode cert DER string to Certificate object.

Parameters **cert_der** – Certificate as a DER encoded string

Returns `cryptography.Certificate()`

`d1_common.cert.x509.disable_cert_validation` ()

Context manager to temporarily disable certificate validation in the standard SSL library.

Note: This should not be used in production code but is sometimes useful for troubleshooting certificate validation issues.

By design, the standard SSL library does not provide a way to disable verification of the server side certificate. However, a patch to disable validation is described by the library developers. This context manager allows applying the patch for specific sections of code.

`d1_common.cert.x509.extract_issuer_ca_cert_url` (*cert_obj*)

Extract issuer CA certificate URL from certificate.

Certificates may include a URL where the root certificate for the CA which was used for signing the certificate can be downloaded. This function returns the URL if present.

The primary use for this is to fix validation failure due to non-trusted issuer by downloading the root CA certificate from the URL and installing it in the local trust store.

Parameters **cert_obj** – `cryptography.Certificate`

Returns Issuer certificate URL if present, else None

Return type str

`d1_common.cert.x509.serialize_private_key_to_pem` (*private_key,*
passphrase_bytes=None)

Serialize private key to PEM.

Parameters

- **private_key**
- **passphrase_bytes**

Returns PEM encoded private key

Return type bytes

`d1_common.cert.x509.generate_private_key` (*key_size=2048*)

Generate a private key.

`d1_common.cert.x509.get_public_key_pem` (*cert_obj*)

Extract public key from certificate as PEM encoded PKCS#1.

Parameters **cert_obj** – `cryptography.Certificate`

Returns PEM encoded PKCS#1 public key.

Return type bytes

`d1_common.cert.x509.save_pem(pem_path, pem_bytes)`

Save PEM encoded bytes to file.

`d1_common.cert.x509.load_csr(pem_path)`

Load CSR from PEM encoded file.

`d1_common.cert.x509.load_private_key(pem_path, passphrase_bytes=None)`

Load private key from PEM encoded file.

`d1_common.cert.x509.generate_cert(ca_issuer, ca_key, csr_subject, csr_pub_key)`

`d1_common.cert.x509.serialize_cert_to_der(cert_obj)`

Serialize certificate to DER.

Parameters `cert_obj` – cryptography.Certificate

Returns DER encoded certificate

Return type bytes

`d1_common.cert.x509.generate_ca_cert(dn, private_key, fqdn_str=None, public_ip=None, private_ip=None, valid_days=3650)`

Parameters

- **dn** – cryptography.x509.Name A cryptography.x509.Name holding a sequence of cryptography.x509.NameAttribute objects.

See the create_dn* functions.

- **private_key** – RSAPrivateKey, etc
- **fqdn_str**
- **public_ip**
- **private_ip**
- **valid_days** – int Number of days from now until the certificate expires.

Returns:

`d1_common.cert.x509.input_key_passphrase(applicable_str='private key')`

`d1_common.cert.x509.check_cert_type(cert)`

`d1_common.cert.x509.rdn_escape(rdn_str)`

Escape string for use as an RDN (RelativeDistinguishedName)

The following chars must be escaped in RDNs: , = + < > # ; “

Parameters `rdn_str` – str

Returns Escaped string ready for use in an RDN (.)

Return type str

`d1_common.cert.x509.log_cert_info(logger, msg_str, cert_obj)`

Dump basic certificate values to the log.

Parameters

- **logger** – Logger Logger to which to write the certificate values.
- **msg_str** – str A message to write to the log before the certificate values.

- **cert_obj** – cryptography.Certificate Certificate containing values to log.

Returns None

`d1_common.cert.x509.get_cert_info_list(cert_obj)`

Get a list of certificate values.

Parameters **cert_obj** – cryptography.Certificate Certificate containing values to retrieve.

Returns Certificate value name, value

Return type list of tup

`d1_common.cert.x509.get_extension_by_name(cert_obj, extension_name)`

Get a standard certificate extension by attribute name.

Parameters

- **cert_obj** – cryptography.Certificate Certificate containing a standard extension.
- **extension_name** – str Extension name. E.g., 'SUBJECT_DIRECTORY_ATTRIBUTES'.

Returns Cryptography.Extension

d1_common.ext package

Submodules

d1_common.ext.mimeparser module

MIME-Type Parser.

This module provides basic functions for handling mime-types. It can handle matching mime-types against a list of media-ranges. See section 14.1 of the HTTP specification [RFC 2616] for a complete explanation.

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1>

Contents:

- `parse_mime_type()`: Parses a mime-type into its component parts.
- `parse_media_range()`: Media-ranges are mime-types with wild-cards and a 'q' quality parameter.
- `quality()`: Determines the quality ('q') of a mime-type when compared against a list of media-ranges.
- `quality_parsed()`: Just like `quality()` except the second parameter must be pre-parsed.
- `best_match()`: Choose the mime-type with the highest quality ('q') from a list of candidates.

`d1_common.ext.mimeparser.parse_mime_type(mime_type)`

Carves up a mime-type and returns a tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range 'application/xhtml;q=0.5' would get parsed into:

('application', 'xhtml', {'q', '0.5'})

`d1_common.ext.mimeparser.parse_media_range(range)`

Carves up a media range and returns a tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range 'application/*;q=0.5' would get parsed into:

('application', '*', {'q', '0.5'})

In addition this function also guarantees that there is a value for 'q' in the params dictionary, filling it in with a proper default if necessary.

`d1_common.ext.mimeparser.fitness_and_quality_parsed(mime_type, parsed_ranges)`

Find the best match for a given mime-type against a list of media_ranges that have already been parsed by `parse_media_range()`.

Returns a tuple of the fitness value and the value of the 'q' quality parameter of the best match, or (-1, 0) if no match was found. Just as for `quality_parsed()`, 'parsed_ranges' must be a list of parsed media ranges.

`d1_common.ext.mimeparser.quality_parsed(mime_type, parsed_ranges)`

Find the best match for a given mime-type against a list of media_ranges that have already been parsed by `parse_media_range()`.

Returns the 'q' quality parameter of the best match, 0 if no match was found. This function bahaves the same as `quality()` except that 'parsed_ranges' must be a list of parsed media ranges.

`d1_common.ext.mimeparser.quality(mime_type, ranges)`

Returns the quality 'q' of a mime-type when compared against the media- ranges in ranges. For example:

```
>>> quality('text/html', 'text/*;q=0.3, text/html;q=0.7, text/html;level=1,
text/html;level=2;q=0.4, */*;q=0.5')
0.7
```

`d1_common.ext.mimeparser.best_match(supported, header)`

Takes a list of supported mime-types and finds the best match for all the media- ranges listed in header. The value of header must be a string that conforms to the format of the HTTP Accept: header. The value of 'supported' is a list of mime-types.

```
>>> best_match(['application/xbel+xml', 'text/xml'], 'text/*;q=0.5,*/*; q=0.1')
'text/xml'
```

d1_common.iter package

This package contains iterators that provide a convenient way to retrieve and iterate over Node contents.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

d1_common.iter.bytes module

Generator that returns a bytes object in chunks.

class `d1_common.iter.bytes.BytesIterator(bytes_, chunk_size=1024)`

Bases: `object`

Generator that returns a bytes object in chunks.

property `size`

Returns:

int: The total number of bytes that will be returned by the iterator.

d1_common.iter.path module

Generator that resolves a list of file and dir paths and returns file paths with optional filtering and client feedback.

```
d1_common.iter.path.path_generator(path_list,          include_glob_list=None,          ex-  
                                clude_glob_list=None,      recursive=True,          ig-  
                                nore_invalid=False,          default_excludes=True,  
                                return_entered_dir_paths=False,          re-  
                                turn_skipped_dir_paths=False)  
  
# language=rst.
```

Parameters

- **path_list** – list of str

List of file- and dir paths. File paths are used directly and dirs are searched for files.

`path_list` does not accept glob patterns, as it's more convenient to let the shell expand glob patterns to directly specified files and dirs. E.g., to use a glob to select all .py files in a subdir, the command may be called with `sub/dir/*.py`, which the shell expands to a list of files, which are then passed to this function. The paths should be Unicode or utf-8 strings. Tilde (“~”) to home expansion is performed on the paths.

The shell can also expand glob patterns to dir paths or a mix of file and dir paths.

- **include_glob_list** – list of str

- **exclude_glob_list** – list of str

Patterns ending with “/” are matched only against dir names. All other patterns are matched only against file names.

If the include list contains any file patterns, files must match one or more of the patterns in order to be returned.

If the include list contains any dir patterns, dirs must match one or more of the patterns in order for the recursive search to descend into them.

The exclude list works in the same way except that matching files and dirs are excluded instead of included. If both include and exclude lists are specified, files and dirs must both match the include and not match the exclude patterns in order to be returned or descended into.

- **recursive** – bool

- **True** (default): Search subdirectories
- **False**: Do not search subdirectories

- **ignore_invalid** – bool

- **True**: Invalid paths in `path_list` are ignored.
- **False** (default): `EnvironmentError` is raised if any of the paths in `path_list` do not reference an existing file or dir.

- **default_excludes** – bool

- **True**: A list of glob patterns for files and dirs that should typically be ignored is added to any exclude patterns passed to the function. These include dirs such as `.git` and backup files, such as files appended with “~”.
- **False**: No files or dirs are excluded by default.

- **return_entered_dir_paths** – bool

- **False:** Only file paths are returned.
- **True:** Directory paths are also returned.
- **return_skipped_dir_paths** – bool
 - **False:** Paths of skipped dirs are not returned.
 - **True:** Paths of skipped dirs are returned.

The iterator never descends into excluded dirs, and by default, does not return the paths of excluded dirs. However, the client may need to get the paths of dirs that were excluded instead of dirs that were included. E.g., when looking for dirs to delete.

Returns File path iterator

Notes

During iteration, the iterator can be prevented from descending into a directory by sending a “skip” flag when the iterator yields the directory path. This allows the client to determine if directories should be iterated by, for instance, which files are present in the directory. This can be used in conjunction with the include and exclude glob lists. Note that, in order to receive directory paths that can be skipped, `return_entered_dir_paths` must be set to `True`.

The regular `for...in` syntax does not support sending the “skip” flag back to the iterator. Instead, use a pattern like:

```
itr = file_iterator.file_iter(..., return_entered_dir_paths=True)
try:
    path = itr.next()
    while True:
        skip_dir = determine_if_dir_should_be_skipped(path)
        file_path = itr.send(skip_dir)
except KeyboardInterrupt:
    raise StopIteration
except StopIteration:
    pass
```

Glob patterns are matched only against file and directory names, not the full paths.

Paths passed directly in `path_list` are not filtered.

The same file can be returned multiple times if `path_list` contains duplicated file paths or dir paths, or dir paths that implicitly include the same subdirs.

`include_glob_list` and `exclude_glob_list` are handy for filtering the files found in dir searches.

Remember to escape the include and exclude glob patterns on the command line so that they’re not expanded by the shell.

class `dl_common.iter.path.ArgParser` (*description_str=None*, *formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>*, ***val_dict*)

Bases: `object`

An `argparse.ArgumentParser` populated with a standard set of command line arguments for controlling the path generator from the command line.

The script that calls this function will typically add its own specific arguments by making additional `parser.add_argument()` calls.

When creating the `path_generator`, simply pass `parser.path_arg_dict` to `path_generator()`.

Example

```
import d1_common.iter.path

parser = d1_common.iter.path.ArgumentParser( __doc__,          # Set non-configurable values in-
                                             include_glob_list=['*.py'], return_entered_dir_paths=True,
) # Add command specific arguments parser.add_argument(...) # Create the path_generator and iterate over the
resulting paths for p in d1_common.iter.path.path_generator(parser.path_arg_dict):

    print(p)

ARG_DICT = {'default_excludes': ('--no-default-excludes', {'action': 'store_false',
__init__(description_str=None, formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>,
        **val_dict)
Create a ArgumentParser populated with a standard set of command line arguments for controlling the
path generator from the command line.
```

Parameters

- **description_str** – Description of the command The description is included in the automatically generated help message.
- **formatter_class** – Modify the help message format. See the *argparse* module for available Formatter classes.
- **fixed value overrides** – Passing any of these arguments causes provided value to be used when instantiating the path generator, and the corresponding command line argument to become hidden and unavailable.

fixed_path_list fixed_exclude_glob_list fixed_include_glob_list fixed_recursive
fixed_ignore_invalid fixed_default_excludes fixed_return_entered_dir_paths
fixed_return_skipped_dir_paths

- **default value overrides** – Passing any of these arguments causes the provided value to be used as the default. The corresponding command line argument is still available and can be used to override the default value

default_path_list default_exclude_glob_list default_include_glob_list
default_recursive default_ignore_invalid default_default_excludes de-
fault_return_entered_dir_paths default_return_skipped_dir_paths

add_argument (*arg_list, **arg_dict)
Add command specific arguments.

property args

Get complete command line arguments as Namespace object.

Returns

Complete command line arguments.

This is an exact representation of the parsed command line and does not include any fixed value substitutions from the val_dict passed to __init__().

Return type Namespace

property path_arg_dict

Get command line arguments as dict suitable for passing to a path_generator create call via argument unpacking.

Returns

Arguments valid for passing to path_generator() create call.

The dict will include any fixed value substitutions that were passed to `__init__()` via the `val_dict`.

Return type dict

d1_common.iter.stream module

class `d1_common.iter.stream.StreamIterator` (*stream, chunk_size=1024*)

Bases: `object`

Generator that returns a stream in chunks.

In this context, a stream is anything with a `read()` method and, if the client requires it, a way to determine the total number of elements that will be returned by the `read()` method at any point during iteration.

Typical sources for streams are files and HTML responses.

__init__ (*stream, chunk_size=1024*)

Args: *stream*: Object with `read()` method, such as an open file.

chunk_size: int Max number of elements to return in each chunk. The last chunk will normally be smaller. Other chunks may be smaller as well, but never empty.

property size

Returns:

int : The total number of bytes that will be returned by the iterator.

d1_common.iter.string module

Generator that returns the Unicode characters of a `str` in chunks.

class `d1_common.iter.string.StringIterator` (*string, chunk_size=1024*)

Bases: `object`

Generator that returns the Unicode characters of a `str` in chunks.

property size

Returns:

int : The total number of characters that will be returned by the iterator.

d1_common.tests package

Submodules

d1_common.tests.test_checksum module

d1_common.tests.test_date_time module

d1_common.tests.test_exceptions module

d1_common.tests.test_file_iterator module

`d1_common.tests.test_jwt` module

`d1_common.tests.test_logrecords` module

`d1_common.tests.test_multipart` module

`d1_common.tests.test_nodelist` module

`d1_common.tests.test_object_format_cache` module

`d1_common.tests.test_objectlist` module

`d1_common.tests.test_objectlocationlist` module

`d1_common.tests.test_pid` module

`d1_common.tests.test_replication_policy` module

`d1_common.tests.test_resource_map` module

`d1_common.tests.test_subject_info` module

`d1_common.tests.test_subject_info_renderer` module

`d1_common.tests.test_system_metadata` module

`d1_common.tests.test_type_conversions` module

`d1_common.tests.test_url` module

`d1_common.tests.test_util` module

`d1_common.tests.test_wrap_access_policy` module

`d1_common.tests.test_wrap_simple_xml` module

`d1_common.tests.test_x509` module

`d1_common.tests.test_xml` module

`d1_common.types` package

DataONE API types

DataONE services use XML messaging over HTTP as the primary means of communication between service nodes and clients. The XML messages are defined by XML Schema specifications and must be valid.

This package provides serialization, deserialization and validation of DataONE API XML types, allowing developers to handle the DataONE types as native objects, reducing development time.

Implicit validation is performed whenever objects are serialized and deserialized, so that developers can assume that information that was received from a DataONE node is complete and syntactically correct before attempting to process it. Also, attempts to submit incomplete or syntactically incorrect information to a DataONE node cause local errors that are easy to debug, rather than less specific errors returned from the target node to which the incorrect types were sent.

Notes

PyXB generated classes are specific to the version of the schema and the version of PyXB installed. Hence, even though PyXB generated classes are provided with the distribution of `d1_common_python`, it may be necessary to regenerate the classes depending on the particular version of PyXB installed.

To regenerate the binding classes, call the `genbind` script:

```
cd to the src folder of this distribution
$ export D1COMMON_ROOT="$(pwd) "
$ bash ${D1COMMON_ROOT}/d1_common/types/scripts/genbind
```

See also:

The DataONE API XML [Schemas](#).

Although this directory is not a package, this `__init__.py` file is required for `pytest` to be able to reach test directories below this directory.

Submodules

`d1_common.types.dataoneErrors` module

Import the PyXB binding required for handling the DataONE Exception types.

`d1_common.types.dataoneTypes` module

Combine the PyXB bindings required for handling all DataONE types.

`d1_common.types.dataoneTypes_v1` module

Import PyXB bindings required for handling v1.0 DataONE types.

`d1_common.types.dataoneTypes_v1_1` module

Combine all PyXB bindings required for handling DataONE types up to and including v1.1.

`d1_common.types.dataoneTypes_v1_2` module

Combine all PyXB bindings required for handling DataONE types up to and including v1.1.

d1_common.types.dataoneTypes_v2_0 module

Combine the PyXB bindings required for handling all DataONE types.

d1_common.types.exceptions module

Native objects for holding DataONE Exceptions.

- Wrap the PyXB client with Exception based classes
- PyXB based XML serialization and deserialization
- Add deserialize to string and HTTP headers

Notes

traceInformation:

traceInformation is an xs:anyType, meaning that is essentially the root of a new XML document of arbitrary complexity. Since the contents of the elements are unknown at the time when the PyXB binding are created, PyXB cannot automatically serialize and deserialize the traceInformation field together with the rest of the DataONEException XML type.

To make it easier to use the traceInformation element, we support a special case where it can be read and written as a single string of bytes, where the contents are application specific. Any other content must be generated and parsed as XML by the user.

Example of serialized DataONE Exception:

```
<error detailCode="1020" errorCode="404" name="NotFound" identifier="testpid">
<description>Attempted to perform operation on non-existing object</description>
<traceInformation>view_handler.py(128)
views.py(102)
auth.py(392)
auth.py(315)
</traceInformation>
</error>
```

`d1_common.types.exceptions.xml_is_dataone_exception(xml_str)`

Return True if XML doc is a valid DataONE Exception.

`d1_common.types.exceptions.pyxb_is_dataone_exception(obj_pyxb)`

Return True if PyXB object is a valid DataONE Exception.

`d1_common.types.exceptions.deserialize(dataone_exception_xml)`

Deserialize a DataONE Exception XML doc.

`d1_common.types.exceptions.deserialize_from_headers(headers)`

Deserialize a DataONE Exception that is stored in a map of HTTP headers (used in responses to HTTP HEAD requests).

`d1_common.types.exceptions.create_exception_by_name(name, detailCode='0', description="", traceInformation=None, identifier=None, nodeId=None)`

Create a DataONEException based object by name.

Parameters

- **name** – str The type name of a DataONE Exception. E.g. NotFound.

If an unknown type name is used, it is automatically set to ServiceFailure. As the XML Schema for DataONE Exceptions does not restrict the type names, this may occur when deserializing an exception not defined by DataONE.

- **detailCode** – int Optional index into a table of predefined error conditions.

See also:

For remaining args, see: `DataONEException()`

```
d1_common.types.exceptions.create_exception_by_error_code(errorCode, detail-
                                                            Code='0', descrip-
                                                            tion="", traceIn-
                                                            formation=None,
                                                            identifier=None,
                                                            nodeId=None)
```

Create a DataONE Exception object by errorCode.

See Also: For args, see: `DataONEException()`

```
exception d1_common.types.exceptions.DataONEException(errorCode, detailCode='0',
                                                         description="", traceInfor-
                                                         mation=None, identifier=None,
                                                         nodeId=None)
```

Bases: `Exception`

Base class for exceptions raised by DataONE.

```
__init__(errorCode, detailCode='0', description="", traceInformation=None, identifier=None,
         nodeId=None)
```

Args: **errorCode**: int HTTP Status code for the error. E.g., NotFound is 404.

detailCode: int Optional index into a table of predefined error conditions.

description: str Optional additional information about the error, intended for users. E.g., if the error is NotFound, this may be the resource that was not found.

traceInformation: str Optional additional information about the error, intended for developers. E.g., stack traces or source code references.

identifier: str Optional Persistent ID (PID) or Series ID (SID).

nodeId: str Optional Node Identifier URN. E.g., `urn:node:MyNode`

```
friendly_format()
```

Serialize to a format more suitable for displaying to end users.

```
serialize_to_transport(encoding='utf-8', xslt_url=None)
```

Serialize to XML bytes with prolog.

Parameters

- **encoding** – str Encoding to use for XML doc bytes
- **xslt_url** – str If specified, add a processing instruction to the XML doc that specifies the download location for an XSLT stylesheet.

Returns XML holding a DataONEError based type.

Return type bytes

```
serialize_to_display(xslt_url=None)
```

Serialize to a pretty printed Unicode str, suitable for display.

Args: `xslt_url`: url Optional link to an XSLT stylesheet. If provided, a processing instruction for the stylesheet is included in the XML prolog.

encode (*encoding*='utf-8')

Serialize to UTF-8 encoded XML bytes with prolog.

serialize_to_headers ()

Serialize to a dict of HTTP headers.

Used in responses to HTTP HEAD requests. As with regular HTTP GET requests, HEAD requests may return DataONE Exceptions. Since a response to a HEAD request cannot include a body, the error is returned as a set of HTTP headers instead of an XML document.

get_pyxb ()

Generate a DataONE Exception PyXB object.

The PyXB object supports directly reading and writing the individual values that may be included in a DataONE Exception.

property name

Returns:

str: Type name of object based on DataONEException. E.g.: AuthenticationTimeout.

exception `d1_common.types.exceptions.AuthenticationTimeout` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type AuthenticationTimeout.

See Also: `DataONEException()`

exception `d1_common.types.exceptions.IdentifierNotUnique` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type IdentifierNotUnique.

See Also: `DataONEException()`

exception `d1_common.types.exceptions.InsufficientResources` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type InsufficientResources.

See Also: `DataONEException()`

exception `d1_common.types.exceptions.InvalidCredentials` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type InvalidCredentials.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.InvalidRequest` (*detailCode*, *description=None*,
traceInformation=None, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `InvalidRequest`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.InvalidSystemMetadata` (*detailCode*, *description=None*, *traceInformation=None*,
identifier=None, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `InvalidSystemMetadata`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.InvalidToken` (*detailCode*, *description=None*,
traceInformation=None, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `InvalidToken`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.NotAuthorized` (*detailCode*, *description=None*,
traceInformation=None, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `NotAuthorized`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.NotFound` (*detailCode*, *description=None*, *traceInformation=None*,
identifier=None, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `NotFound`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.NotImplemented` (*detailCode*, *description=None*,
traceInformation=None, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `NotImplemented`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.ServiceFailure` (*detailCode*, *description=None*,
traceInformation=None, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `ServiceFailure`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.UnsupportedMetadataType` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `UnsupportedMetadataType`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.UnsupportedType` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `UnsupportedType`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.SynchronizationFailed` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `SynchronizationFailed`.

See Also: `DataONEException()`

exception `dl_common.types.exceptions.VersionMismatch` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `dl_common.types.exceptions.DataONEException`

DataONE Exception of type `VersionMismatch`.

See Also: `DataONEException()`

dl_common.utils package

DataONE Common Library.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

dl_common.utils.filesystem module

Utilities for filesystem paths and operations.

`dl_common.utils.filesystem.gen_safe_path(*path_list)`

Escape characters that are not allowed or often cause issues when used in file- or directory names, then join the arguments to a filesystem path.

Parameters **positional args** – str Strings to use as elements in a filesystem path, such as PID, SID or URL.

Returns A path safe for use as a file- or directory name.

Return type str

`dl_common.utils.filesystem.gen_safe_path_element(s)`

Escape characters that are not allowed or often cause issues when used in file- or directory names.

Leading and trailing “.” (period) are stripped out in order to prevent inadvertently creating hidden files.

Parameters *s* – str Any string, such as a PID, SID or URL

Returns A string safe for use as a file- or directory name.

Return type str

`dl_common.utils.filesystem.create_missing_directories_for_file(file_path)`

Create any directories in *dir_path* that do not yet exist.

Parameters *file_path* – str Relative or absolute path to a file that may or may not exist.

Must be a file path, as any directory element at the end of the path will not be created.

See also:

`create_missing_directories_for_dir()`

`dl_common.utils.filesystem.create_missing_directories_for_dir(dir_path)`

Create any directories in *dir_path* that do not yet exist.

Parameters *dir_path* – str Relative or absolute path to a directory that may or may not exist.

Must be a directory path, as any filename element at the end of the path will also be created as a directory.

See also:

`create_missing_directories_for_file()`

`dl_common.utils.filesystem.abs_path_from_base(base_path, rel_path)`

Join a base and a relative path and return an absolute path to the resulting location.

Parameters

- **base_path** – str Relative or absolute path to prepend to *rel_path*.
- **rel_path** – str Path relative to the location of the module file from which this function is called.

Returns Absolute path to the location specified by *rel_path*.

Return type str

`dl_common.utils.filesystem.abs_path(rel_path)`

Convert a path that is relative to the module from which this function is called, to an absolute path.

Parameters *rel_path* – str Path relative to the location of the module file from which this function is called.

Returns Absolute path to the location specified by *rel_path*.

Return type str

d1_common.utils.progress_logger module

One stop shop for providing progress information and event counts during time consuming operations performed in command line scripts and Django management commands.

The ProgressLogger keeps track of how many tasks have been processed by a script, how many are remaining, and how much time has been used. It then calculates and periodically displays a progress update containing an ETA and completed percentage.

The ProgressLogger can also be used for counting errors and other notable events that may occur during processing, and displays total count for each type of tracked event in the progress updates.

In the following example, progress information is added to a script that processes the tasks in a list of tasks. All the tasks require the same processing, so there's only one task type, and one loop in the script.

```
import logging
import d1_common.utils.progress_logger

def main():
    logging.basicConfig(level=logging.DEBUG)

    progress_logger = d1_common.utils.progress_logger.ProgressLogger()
    long_task_list = get_long_task_list()

    self.progress_logger.start_task_type("My time consuming task", len(long_task_list))

    for task in long_task_list:
        self.progress_logger.start_task("My time consuming task")
        do_time_consuming_work_on_task(task)
        if task.has_some_issue():
            progress_logger.event("Task has issue")

        if task.has_other_issue():
            progress_logger.event("Task has other issue")

    self.progress_logger.end_task_type("My time consuming task")
    self.progress_logger.completed()
```

Yields progress output such as:

```
My time consuming task: 64/1027 (6.23% 0d00h00m) My time consuming task: 123/1027 (11.98% 0d00h00m) My time consuming task: 180/1027 (17.53% 0d00h00m) Events:
```

```
Task has issue: 1
```

```
My time consuming task: 236/1027 (22.98% 0d00h00m) Events:
```

```
Task has issue: 2 Task has other issue: 1
```

```
My time consuming task: 436/1027 (32.98% 0d00h00m) Events:
```

```
Task has issue: 2 Task has other issue: 1
```

```
My time consuming task: 636/1027 (44.12% 0d00h00m) Events:
```

```
Task has issue: 2 Task has other issue: 1
```

```
Completed. runtime_sec=5.44 total_run_dhm="0d00h00m"
```

```
class d1_common.utils.progress_logger.ProgressLogger(logger=None, log_level=20,
                                                    log_interval_sec=1.0)
```

```
Bases: object
```

```
__init__(logger=None, log_level=20, log_interval_sec=1.0)
```

Create one object of this class at the start of the script and keep a reference to it while the script is running.

Parameters

- **logger** – Optional logger to which the progress log entries are written. A new logger is created if not provided.
- **log_level** – The level of severity to set for the progress log entries.
- **log_interval_sec** – Minimal time between writing log entries. Log entries may be written with less time between entries if the total processing time for a task type is less than the interval, or if processing multiple task types concurrently.

start_task_type (*task_type_str*, *total_task_count*)

Call when about to start processing a new type of task, typically just before entering a loop that processes many task of the given type.

Parameters

- **task_type_str** (*str*) – The name of the task, used as a dict key and printed in the progress updates.
- **total_task_count** (*int*) – The total number of the new type of task that will be processed.

This starts the timer that is used for providing an ETA for completing all tasks of the given type.

The task type is included in progress updates until `end_task_type()` is called.

end_task_type (*task_type_str*)

Call when processing of all tasks of the given type is completed, typically just after exiting a loop that processes many tasks of the given type.

Progress messages logged at intervals will typically not include the final entry which shows that processing is 100% complete, so a final progress message is logged here.

start_task (*task_type_str*, *current_task_index=None*)

Call when processing is about to start on a single task of the given task type, typically at the top inside of the loop that processes the tasks.

Parameters

- **task_type_str** (*str*) – The name of the task, used as a dict key and printed in the progress updates.
- **current_task_index** (*int*) – If the task processing loop may skip or repeat tasks, the index of the current task must be provided here. This parameter can normally be left unset.

event (*event_name*, *count_int=1*)

Register an event that occurred during processing of a task of the given type.

Args: *event_name*: *str* A name for a type of events. Events of the same type are displayed as a single entry and a total count of occurrences.

completed ()

Call when about to exit the script.

Logs total runtime for the script and issues a warning if there are still active task types. Active task types should be closed with `end_task_type()` when processing is completed for tasks of the given type in order for accurate progress messages to be displayed.

d1_common.wrap package

DataONE API Type wrappers.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

`d1_common.wrap.access_policy` module

Context manager for working with the DataONE `AccessPolicy` type.

Examples

Perform multiple operations on an `AccessPolicy`:

```
# Wrap a SystemMetadata PyXB object to modify its AccessPolicy section
with d1_common.wrap.access_policy.wrap(sysmeta_pyxb) as ap:

    # Print a list of subjects that have the changePermission access level
    print(ap.get_subjects('changePermission'))

    # Clear any existing rules in the access policy
    ap.clear()

    # Add a new rule
    ap.add_perm('subj1', 'read')

# Exit the context manager scope to write the changes that were made back to the
# wrapped SystemMetadata.
```

If only a single operation is to be performed, use one of the module level functions:

```
# Add public read permission to an AccessPolicy. This adds an allow rule with
# a "read" permission for the symbolic subject, "public". It is a no-op if any of
# the existing rules already provide "read" or better to "public".
add_public_read(access_pyxb)
```

Notes

Overview:

Each science object in DataONE has an associated `SystemMetadata` document in which there is an `AccessPolicy` element. The `AccessPolicy` contains rules assigning permissions to subjects. The supported permissions are `read`, `write` and `changePermission`.

`write` implicitly includes `read`, and `changePermission` implicitly includes `read` and `write`. So, only a single permission needs to be assigned to a subject in order to determine all permissions for the subject.

There can be multiple rules in a policy and each rule can contain multiple subjects and permissions. So the same subject can be specified multiple times in the same rules or in different rules, each time with a different set of permissions, while permissions also implicitly include lower permissions.

Due to this, the same permissions can be expressed in many different ways. This wrapper hides the variations, exposing a single canonical set of rules that can be read, modified and written. That is, the wrapper allows working with any set of permissions in terms of the simplest possible representation that covers the resulting effective permissions.

E.g., the following two access policies are equivalent. The latter represents the canonical representation of the former.

```

<accessPolicy>
  <allow>
    <subject>subj2</subject>
    <subject>subj1</subject>
    <perm>read</perm>
  </allow>
  <allow>
    <subject>subj4</subject>
    <perm>read</perm>
    <perm>changePermission</perm>
  </allow>
  <allow>
    <subject>subj2</subject>
    <subject>subj3</subject>
    <perm>read</perm>
    <perm>write</perm>
  </allow>
  <allow>
    <subject>subj5</subject>
    <perm>read</perm>
    <perm>write</perm>
  </allow>
</accessPolicy>

```

and

```

<accessPolicy>
  <allow>
    <subject>subj1</subject>
    <perm>read</perm>
  </allow>
  <allow>
    <subject>subj2</subject>
    <subject>subj3</subject>
    <subject>subj5</subject>
    <perm>write</perm>
  </allow>
  <allow>
    <subject>subj4</subject>
    <perm>changePermission</perm>
  </allow>
</accessPolicy>

```

Representations of rules, permissions and subjects:

subj_dict maps each subj to the perms the the subj has specifically been given. It holds perms just having been read for PyXB. Duplicates caused by the same subj being given the same perm in multiple ways are filtered out.

```

{
  'subj1': { 'read' },
  'subj2': { 'read', 'write' },
  'subj3': { 'read', 'write' },
  'subj4': { 'changePermission', 'read' },
  'subj5': { 'read', 'write' }
}

```

perm_dict maps each perm that a subj has specifically been given, to the subj. If the AccessPolicy contains multiple allow elements, and they each give different perms to a subj, those show up as additional mappings. Duplicates

caused by the same subj being given the same perm in multiple ways are filtered out. Calls such as `add_perm()` also cause extra mappings to be added here, as long as they're not exact duplicates. Whenever this dict is used for generating PyXB or making comparisons, it is first normalized to a `norm_perm_list`.

```
{
  'read': { 'subj1', 'subj2' },
  'write': { 'subj3' },
  'changePermission': { 'subj2' },
}
```

`subj_highest_dict` maps each subj to the highest perm the subj has. The dict has the same number of keys as there are subj.

```
{
  'subj1': 'write',
  'subj2': 'changePermission',
  'subj3': 'write',
}
```

`highest_perm_dict` maps the highest perm a subj has, to the subj. The dict can have at most 3 keys:

```
{
  'changePermission': { 'subj2', 'subj3', 'subj5', 'subj6' },
  'read': { 'public' },
  'write': { 'subj1', 'subj4' }
}
```

`norm_perm_list` is a minimal, ordered and hashable list of lists. The top level has up to 3 lists, one for each perm that is in use. Each of the lists then has a list of subj for which that perm is the highest perm. `norm_perm_list` is the shortest way that the required permissions can be expressed, and is used for comparing access policies and creating uniform PyXB objects:

```
[
  ['read', ['public']],
  ['write', ['subj1', 'subj4']],
  ['changePermission', ['subj2', 'subj3', 'subj5', 'subj6']]
]
```

`d1_common.wrap.access_policy.wrap` (*access_pyxb*, *pyxb_binding=None*, *read_only=False*)
Work with the AccessPolicy in a SystemMetadata PyXB object.

Parameters

- **access_pyxb** – AccessPolicy PyXB object The AccessPolicy to modify.
- **read_only** – bool Do not update the wrapped AccessPolicy.

When only a single AccessPolicy operation is needed, there's no need to use this context manager. Instead, use the generated context manager wrappers.

`d1_common.wrap.access_policy.wrap_sysmeta_pyxb` (*sysmeta_pyxb*, *pyxb_binding=None*, *read_only=False*)
Work with the AccessPolicy in a SystemMetadata PyXB object.

Parameters

- **sysmeta_pyxb** – SystemMetadata PyXB object SystemMetadata containing the AccessPolicy to modify.
- **read_only** – bool Do not update the wrapped AccessPolicy.

When only a single AccessPolicy operation is needed, there's no need to use this context manager. Instead, use the generated context manager wrappers.

There is no clean way in Python to make a context manager that allows client code to replace the object that is passed out of the manager. The AccessPolicy schema does not allow the AccessPolicy element to be empty. However, the SystemMetadata schema specifies the AccessPolicy as optional. By wrapping the SystemMetadata instead of the AccessPolicy when working with AccessPolicy that is within SystemMetadata, the wrapper can handle the situation of empty AccessPolicy by instead dropping the AccessPolicy from the SystemMetadata.

```
class dl_common.wrap.access_policy.AccessPolicyWrapper (access_pyxb,  
                                                         pyxb_binding=None)
```

Bases: object

Wrap an AccessPolicy and provide convenient methods to read, write and update it.

Parameters *access_pyxb* – AccessPolicy PyXB object The AccessPolicy to modify.

update ()

Update the wrapped AccessPolicy PyXB object with normalized and minimal rules representing current state.

get_normalized_pyxb ()

Returns:

AccessPolicy PyXB object : Current state of the wrapper as the minimal rules required for correctly representing the perms.

get_normalized_perm_list ()

Returns:

A minimal, ordered, hashable list of subjects and permissions that represents the current state of the wrapper.

get_highest_perm_str (*subj_str*)

Parameters *subj_str* – str Subject for which to retrieve the highest permission.

Returns The highest permission for subject or None if subject does not have any permissions.

get_effective_perm_list (*subj_str*)

Parameters *subj_str* – str Subject for which to retrieve the effective permissions.

Returns

List of permissions up to and including the highest permission for subject, ordered lower to higher, or empty list if subject does not have any permissions.

E.g.: If 'write' is highest permission for subject, return ['read', 'write'].

Return type list of str

get_subjects_with_equal_or_higher_perm (*perm_str*)

Parameters *perm_str* – str Permission, read, write or changePermission.

Returns

Subj that have perm equal or higher than *perm_str*.

Since the lowest permission a subject can have is read, passing read will return all subjects.

Return type set of str

dump ()

Dump the current state to debug level log.

is_public()

Returns:

bool: True if AccessPolicy allows public read.

is_private()

Returns:

bool: **True** if AccessPolicy does not grant access to any subjects.

is_empty()

Returns:

bool: True if AccessPolicy does not grant access to any subjects.

are_equivalent_pyxb(*access_pyxb*)

Parameters *access_pyxb* – AccessPolicy PyXB object with which to compare.

Returns

True if *access_pyxb* grants the exact same permissions as the wrapped AccessPolicy.

Differences in how the permissions are represented in the XML docs are handled by transforming to normalized lists before comparison.

Return type bool

are_equivalent_xml(*access_xml*)

Parameters *access_xml* – AccessPolicy XML doc with which to compare.

Returns

True if *access_xml* grants the exact same permissions as the wrapped AccessPolicy.

Differences in how the permissions are represented in the XML docs are handled by transforming to normalized lists before comparison.

Return type bool

subj_has_perm(*subj_str*, *perm_str*)

Returns:

bool: True if *subj_str* has perm equal to or higher than *perm_str*.

clear()

Remove AccessPolicy.

Only the rightsHolder set in the SystemMetadata will be able to access the object unless new perms are added after calling this method.

add_public_read()

Add public public read perm.

Add an allow rule with a read permission for the symbolic subject, *public*. It is a no no-op if any of the existing rules already provide read or higher to *public*.

add_authenticated_read()

Add read perm for all authenticated subj.

Public read is removed if present.

add_verified_read()

Add read perm for all verified subj.

Public read is removed if present.

add_perm (*subj_str*, *perm_str*)

Add a permission for a subject.

Parameters

- **subj_str** – str Subject for which to add permission(s)
- **perm_str** – str Permission to add. Implicitly adds all lower permissions. E.g., `write` will also add `read`.

remove_perm (*subj_str*, *perm_str*)

Remove permission from a subject.

Parameters

- **subj_str** – str Subject for which to remove permission(s)
- **perm_str** – str Permission to remove. Implicitly removes all higher permissions. E.g., `write` will also remove `changePermission` if previously granted.

remove_subj (*subj_str*)

Remove all permissions for subject.

Parameters **subj_str** – str Subject for which to remove all permissions. Since subjects can only be present in the AccessPolicy when they have one or more permissions, this removes the subject itself as well.

The subject may still have access to the obj. E.g.:

- The obj has public access.
- The subj has indirect access by being in a group which has access.
- The subj has an equivalent subj that has access.
- The subj is set as the rightsHolder for the object.

`d1_common.wrap.access_policy.update` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.get_normalized_pyxb` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.get_normalized_perm_list` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.get_highest_perm_str` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.get_effective_perm_list` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.get_subjects_with_equal_or_higher_perm` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.dump` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.is_public` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.is_private` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.is_empty` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.are_equivalent_pyxb` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.are_equivalent_xml` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.subj_has_perm` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.clear` (*access_pyxb*, *args, **kwargs)

`d1_common.wrap.access_policy.add_public_read` (*access_pyxb*, *args, **kwargs)

```
d1_common.wrap.access_policy.add_authenticated_read(access_pyxb, *args, **kwargs)
d1_common.wrap.access_policy.add_verified_read(access_pyxb, *args, **kwargs)
d1_common.wrap.access_policy.add_perm(access_pyxb, *args, **kwargs)
d1_common.wrap.access_policy.remove_perm(access_pyxb, *args, **kwargs)
d1_common.wrap.access_policy.remove_subj(access_pyxb, *args, **kwargs)
d1_common.wrap.access_policy.mk_func(func_name)
d1_common.wrap.access_policy.method_obj(self)
```

Update the wrapped AccessPolicy PyXB object with normalized and minimal rules representing current state.

d1_common.wrap.simple_xml module

Context manager for simple XML processing.

Example

```
with d1_common.wrap.simple_xml.wrap(my_xml_str) as xml_wrapper:
    # Read, modify and write the text in an XML element
    text_str = xml.get_element_text('my_el')
    xml.set_element_text('{} more text'.format(text_str))
    # Discard the wrapped XML and replace it with the modified XML. Calling get_xml()
    # is required because context managers cannot replace the object that was passed
    # to the manager, and strings are immutable. If the wrapped XML is needed later,
    # just store another reference to it.
    my_xml_str = xml_wrapper.get_xml()
```

Notes

Typically, the DataONE Python stack, and any apps based on the stack, process XML using the PyXB bindings for the DataONE XML types. However, in some rare cases, it is necessary to process XML without using PyXB, and this wrapper provides some basic methods for such processing.

Uses include:

- Process XML that is not DataONE types, and so does not have PyXB binding.
- Process XML that is invalid in such a way that PyXB cannot parse or generate it.
- Process XML without causing xs:dateTime fields to be normalized to the UTC time zone (PyXB is based on the XML DOM, which requires such normalization.)
- Generate intentionally invalid XML for DataONE types in order to test how MNs, CNs and other components of the DataONE architecture handle and recover from invalid input.
- Speed up simple processing, when the performance overhead of converting the documents to and from PyXB objects, with the schema validation and other processing that it entails, would be considered too high.

Usage:

- Methods that take `el_name` and `el_idx` operate on the element with index `el_idx` of elements with name `el_name`. If `el_idx` is higher than the number of elements with name `el_name`, SimpleXMLWrapperException is raised.

- Though this wrapper does not require XML to validate against the DataONE schemas, it does require that the wrapped XML is well formed and it will only generate well formed XML.
- If it's necessary to process XML that is not well formed, a library such as BeautifulSoup may be required.
- In some cases, it may be possible read or write XML that is not well formed by manipulating the XML directly as a string before wrapping or after generating.
- This wrapper is based on the ElementTree module.

```
dl_common.wrap.simple_xml.wrap(xml_str)
```

Simple processing of XML.

```
class dl_common.wrap.simple_xml.SimpleXMLWrapper(xml_str)
```

Bases: object

Wrap an XML document and provide convenient methods for performing simple processing on it.

Parameters `xml_str` – str XML document to read, write or modify.

```
parse_xml(xml_str)
```

```
get_xml(encoding='unicode')
```

Returns:

str : Current state of the wrapper as XML

```
get_pretty_xml(encoding='unicode')
```

Returns:

str : Current state of the wrapper as a pretty printed XML string.

```
get_xml_below_element(el_name, el_idx=0, encoding='unicode')
```

Parameters

- **el_name** – str Name of element that is the base of the branch to retrieve.
- **el_idx** – int Index of element to use as base in the event that there are multiple sibling elements with the same name.

Returns XML fragment rooted at `el`.

Return type str

```
get_element_list_by_name(el_name, namespaces=None)
```

Parameters `el_name` – str Name of element for which to search.

Returns

List of elements with name `el_name`.

If there are no matching elements, an empty list is returned.

Return type list

```
get_element_list_by_attr_key(attr_key, namespaces=None)
```

Parameters `attr_key` – str Name of attribute for which to search

Returns

List of elements containing an attribute key named `attr_key`.

If there are no matching elements, an empty list is returned.

Return type list

get_element_by_xpath (*xpath_str*, *namespaces=None*)

Parameters *xpath_str* – str XPath matching the elements for which to search.

Returns

List of elements matching *xpath_str*.

If there are no matching elements, an empty list is returned.

Return type list

get_element_by_name (*el_name*, *el_idx=0*)

Parameters

- **el_name** – str Name of element to get.
- **el_idx** – int Index of element to use as base in the event that there are multiple sibling elements with the same name.

Returns The selected element.

Return type element

get_element_by_attr_key (*attr_key*, *el_idx=0*)

Parameters

- **attr_key** – str Name of attribute for which to search
- **el_idx** – int Index of element to use as base in the event that there are multiple sibling elements with the same name.

Returns Element containing an attribute key named *attr_key*.

get_element_text (*el_name*, *el_idx=0*)

Parameters

- **el_name** – str Name of element to use.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

Returns Text of the selected element.

Return type str

set_element_text (*el_name*, *el_text*, *el_idx=0*)

Parameters

- **el_name** – str Name of element to update.
- **el_text** – str Text to set for element.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

get_element_text_by_attr_key (*attr_key*, *el_idx=0*)

Parameters

- **attr_key** – str Name of attribute for which to search
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

Returns Text of the selected element.

Return type str

set_element_text_by_attr_key (*attr_key, el_text, el_idx=0*)

Parameters

- **attr_key** – str Name of attribute for which to search
- **el_text** – str Text to set for element.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

get_attr_value (*attr_key, el_idx=0*)

Return the value of the selected attribute in the selected element.

Parameters

- **attr_key** – str Name of attribute for which to search
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

Returns Value of the selected attribute in the selected element.

Return type str

set_attr_text (*attr_key, attr_val, el_idx=0*)

Set the value of the selected attribute of the selected element.

Parameters

- **attr_key** – str Name of attribute for which to search
- **attr_val** – str Text to set for the attribute.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

get_element_dt (*el_name, tz=None, el_idx=0*)

Return the text of the selected element as a `datetime.datetime` object.

The element text must be a ISO8601 formatted datetime

Parameters

- **el_name** – str Name of element to use.
- **tz** – `datetime.tzinfo` Timezone in which to return the datetime.
 - Without a timezone, other contextual information is required in order to determine the exact represented time.
 - If dt has timezone: The `tz` parameter is ignored.
 - If dt is naive (without timezone): The timezone is set to `tz`.
 - `tz=None`: Prevent naive dt from being set to a timezone. Without a timezone, other contextual information is required in order to determine the exact represented time.
 - `tz=datetime.datetime.UTC()`: Set naive dt to UTC.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

Returns `datetime.datetime`

set_element_dt (*el_name*, *dt*, *tz=None*, *el_idx=0*)

Set the text of the selected element to an ISO8601 formatted datetime.

Parameters

- **el_name** – str Name of element to update.
- **dt** – datetime.datetime Date and time to set
- **tz** – datetime.tzinfo Timezone to set
 - Without a timezone, other contextual information is required in order to determine the exact represented time.
 - If dt has timezone: The `tz` parameter is ignored.
 - If dt is naive (without timezone): The timezone is set to `tz`.
 - `tz=None`: Prevent naive dt from being set to a timezone. Without a timezone, other contextual information is required in order to determine the exact represented time.
 - `tz=dl_common.date_time.UTC()`: Set naive dt to UTC.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

remove_children (*el_name*, *el_idx=0*)

Remove any child elements from element.

Parameters

- **el_name** – str Name of element to update.
- **el_idx** – int Index of element to use in the event that there are multiple sibling elements with the same name.

replace_by_etree (*root_el*, *el_idx=0*)

Replace element.

Select element that has the same name as `root_el`, then replace the selected element with `root_el`

`root_el` can be a single element or the root of an element tree.

Parameters **root_el** – element New element that will replace the existing element.

replace_by_xml (*xml_str*, *el_idx=0*)

Replace element.

Select element that has the same name as `xml_str`, then replace the selected element with `xml_str`

- `xml_str` must have a single element in the root.
- The root element in `xml_str` can have an arbitrary number of children.

Parameters **xml_str** – str New element that will replace the existing element.

exception `dl_common.wrap.simple_xml.SimpleXMLWrapperException`

Bases: `Exception`

Submodules

d1_common.bagit module

Create and validate BagIt Data Packages / zip file archives.

See also:

- <https://en.wikipedia.org/wiki/BagIt>
- <https://tools.ietf.org/html/draft-kunze-bagit-05>
- https://releases.dataone.org/online/api-documentation-v2.0.1/apis/MN_APIs.html#MNPackage.getPackage
- <https://releases.dataone.org/online/api-documentation-v2.0/design/DataPackage.html>

`d1_common.bagit.validate_bagit_file(bagit_path)`

Check if a BagIt file is valid.

Raises *ServiceFailure* – If the BagIt zip archive file fails any of the following checks:

- Is a valid zip file.
- The tag and manifest files are correctly formatted.
- Contains all the files listed in the manifests.
- The file checksums match the manifests.

`d1_common.bagit.create_bagit_stream(dir_name, payload_info_list)`

Create a stream containing a BagIt zip archive.

Parameters

- **dir_name** – str The name of the root directory in the zip file, under which all the files are placed (avoids “zip bombs”).
- **payload_info_list** – list List of payload_info_dict, each dict describing a file.
 - keys: pid, filename, iter, checksum, checksum_algorithm
 - If the filename is None, the pid is used for the filename.

d1_common.checksum module

Utilities for handling checksums.

Warning: The MD5 checksum algorithm is not cryptographically secure. It’s possible to craft a sequence of bytes that yields a predetermined checksum.

`d1_common.checksum.create_checksum_object_from_stream(f, algorithm='SHA-1')`

Calculate the checksum of a stream.

Parameters

- **f** – file-like object Only requirement is a `read()` method that returns bytes.
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns Populated Checksum PyXB object.

`d1_common.checksum.create_checksum_object_from_iterator(itr, algorithm='SHA-1')`

Calculate the checksum of an iterator.

Parameters

- **itr** – iterable Object which supports the iterator protocol.
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns Populated Checksum PyXB object.

`d1_common.checksum.create_checksum_object_from_bytes(b, algorithm='SHA-1')`
Calculate the checksum of bytes.

Warning: This method requires the entire object to be buffered in (virtual) memory, which should normally be avoided in production code.

Parameters

- **b** – bytes Raw bytes
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns Populated PyXB Checksum object.

`d1_common.checksum.calculate_checksum_on_stream(f, algorithm='SHA-1', chunk_size=1048576)`
Calculate the checksum of a stream.

Parameters

- **f** – file-like object Only requirement is a `read()` method that returns bytes.
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.
- **chunk_size** – int Number of bytes to read from the file and add to the checksum at a time.

Returns Checksum as a hexadecimal string, with length decided by the algorithm.

Return type str

`d1_common.checksum.calculate_checksum_on_iterator(itr, algorithm='SHA-1')`
Calculate the checksum of an iterator.

Parameters

- **itr** – iterable Object which supports the iterator protocol.
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns Checksum as a hexadecimal string, with length decided by the algorithm.

Return type str

`d1_common.checksum.calculate_checksum_on_bytes(b, algorithm='SHA-1')`
Calculate the checksum of bytes.

Warning: This method requires the entire object to be buffered in (virtual) memory, which should normally be avoided in production code.

Parameters

- **b** – bytes Raw bytes
- **algorithm** – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns Checksum as a hexadecimal string, with length decided by the algorithm.

Return type str

`dl_common.checksum.are_checksums_equal (checksum_a_pyxb, checksum_b_pyxb)`

Determine if checksums are equal.

Parameters `checksum_a_pyxb, checksum_b_pyxb` – PyXB Checksum objects to compare.

Returns

bool

- **True:** The checksums contain the same hexadecimal values calculated with the same algorithm. Identical checksums guarantee (for all practical purposes) that the checksums were calculated from the same sequence of bytes.
- **False:** The checksums were calculated with the same algorithm but the hexadecimal values are different.

Raises **ValueError** – The checksums were calculated with different algorithms, hence cannot be compared.

`dl_common.checksum.get_checksum_calculator_by_dataone_designator (dataone_algorithm_name)`

Get a checksum calculator.

Parameters `dataone_algorithm_name` – str Checksum algorithm, MD5 or SHA1 / SHA-1.

Returns

Checksum calculator from the hashlib library

Object that supports `update (arg)`, `digest ()`, `hexdigest ()` and `copy ()`.

`dl_common.checksum.get_default_checksum_algorithm ()`

Get the default checksum algorithm.

Returns

Checksum algorithm that is supported by DataONE, the DataONE Python stack and is in common use within the DataONE federation. Currently, SHA-1.

The returned string can be passed as the `algorithm_str` to the functions in this module.

Return type str

`dl_common.checksum.is_supported_algorithm (algorithm_str)`

Determine if string is the name of a supported checksum algorithm.

Parameters `algorithm_str` – str String that may or may not contain the name of a supported algorithm.

Returns

bool

- **True:** The string contains the name of a supported algorithm and can be passed as the `algorithm_str` to the functions in this module.
- **False:** The string is not a supported algorithm.

`dl_common.checksum.get_supported_algorithms ()`

Get a list of the checksum algorithms that are supported by the DataONE Python stack.

Returns List of algorithms that are supported by the DataONE Python stack and can be passed to as the `algorithm_str` to the functions in this module.

Return type list

`dl_common.checksum.format_checksum (checksum_pyxb)`

Create string representation of a PyXB Checksum object.

Parameters PyXB Checksum object

Returns Combined hexadecimal value and algorithm name.

Return type str

d1_common.const module

System wide constants for the Python DataONE stack.

d1_common.date_time module

Utilities for handling date-times in DataONE.

Timezones (tz):

- A datetime object can be **tz-naive** or **tz-aware**.
- **tz-naive**: The datetime does not include timezone information. As such, it does not by itself fully specify an absolute point in time. The exact point in time depends on in which timezone the time is specified, and the information may not be accessible to the end user. However, as timezones go from GMT-12 to GMT+14, and when including a possible daylight saving offset of 1 hour, a tz-naive datetime will always be within 14 hours of the real time.
- **tz-aware**: The datetime includes a timezone, specified as an abbreviation or as a hour and minute offset. It specifies an exact point in time.

class d1_common.date_time.UTC

Bases: datetime.tzinfo

datetime.tzinfo based class that represents the UTC timezone.

Date-times in DataONE should have timezone information that is fixed to UTC. A naive Python datetime can be fixed to UTC by attaching it to this datetime.tzinfo based class.

utcoffset (dt)

Returns:

UTC offset of zero

tzname (dt=None)

Returns:

str: "UTC"

dst (dt=None)

Args: dt: Ignored.

Returns: timedelta(0), meaning that daylight saving is never in effect.

class d1_common.date_time.FixedOffset (name, offset_hours=0, offset_minutes=0)

Bases: datetime.tzinfo

datetime.tzinfo derived class that represents any timezone as fixed offset in minutes east of UTC.

- Date-times in DataONE should have timezone information that is fixed to UTC. A naive Python datetime can be fixed to UTC by attaching it to this datetime.tzinfo based class.
- See the UTC class for representing timezone in UTC.

__init__ (name, offset_hours=0, offset_minutes=0)

Args: name: str Name of the timezone this offset represents.

offset_hours: Number of hours offset from UTC.

offset_minutes: Number of minutes offset from UTC.

utcoffset (*dt*)

Args: *dt*: Ignored.

Returns The time offset from UTC.

Return type `datetime.timedelta`

tzname (*dt*)

Args: *dt*: Ignored.

Returns: Name of the timezone this offset represents.

dst (*dt=None*)

Args: *dt*: Ignored.

Returns: `timedelta(0)`, meaning that daylight saving is never in effect.

`d1_common.date_time.is_valid_iso8601` (*iso8601_str*)

Determine if string is a valid ISO 8601 date, time, or datetime.

Parameters *iso8601_str* – str String to check.

Returns True if string is a valid ISO 8601 date, time, or datetime.

Return type bool

`d1_common.date_time.has_tz` (*dt*)

Determine if datetime has timezone (is not naive)

Parameters *dt* – datetime

Returns

bool

- **True:** datetime is tz-aware.
- **False:** datetime is tz-naive.

`d1_common.date_time.is_utc` (*dt*)

Determine if datetime has timezone and the timezone is in UTC.

Parameters *dt* – datetime

Returns True if datetime has timezone and the timezone is in UTC

Return type bool

`d1_common.date_time.are_equal` (*a_dt, b_dt, round_sec=1*)

Determine if two datetimes are equal with fuzz factor.

A naive datetime (no timezone information) is assumed to be in in UTC.

Parameters

- **a_dt** – datetime Timestamp to compare.
- **b_dt** – datetime Timestamp to compare.
- **round_sec** – int or float Round the timestamps to the closest second divisible by this value before comparing them.

E.g.:

– `n_round_sec = 0.1`: nearest 10th of a second.

- `n_round_sec = 1`: nearest second.
- `n_round_sec = 30`: nearest half minute.

Timestamps may lose resolution or otherwise change slightly as they go through various transformations and storage systems. This again may cause timestamps that have been processed in different systems to fail an exact equality compare even if they were initially the same timestamp. This rounding avoids such problems as long as the error introduced to the original timestamp is not higher than the rounding value. Of course, the rounding also causes a loss in resolution in the values compared, so should be kept as low as possible. The default value of 1 second should be a good tradeoff in most cases.

Returns

bool

- **True**: If the two datetimes are equal after being rounded by `round_sec`.

`d1_common.date_time.ts_from_dt(dt)`

Convert datetime to POSIX timestamp.

Parameters `dt` – datetime

- Timezone aware datetime: The `tz` is included and adjusted to UTC (since timestamp is always in UTC).
- Naive datetime (no timezone information): Assumed to be in UTC.

Returns

int or float

- The number of seconds since Midnight, January 1st, 1970, UTC.
- If `dt` contains sub-second values, the returned value will be a float with fraction.

See also:

`dt_from_ts()` for the reverse operation.

`d1_common.date_time.dt_from_ts(ts, tz=None)`

Convert POSIX timestamp to a timezone aware datetime.

Parameters

- **ts** – int or float, optionally with fraction The number of seconds since Midnight, January 1st, 1970, UTC.
- **tz** – `datetime.tzinfo` - If supplied: The `dt` is adjusted to that `tz` before being returned. It does not affect the `ts`, which is always in UTC.
- If not supplied: the `dt` is returned in UTC.

Returns

datetime Timezone aware datetime, in UTC.

See also:

`ts_from_dt()` for the reverse operation.

`d1_common.date_time.http_datetime_str_from_dt(dt)`

Format datetime to HTTP Full Date format.

Parameters `dt` – datetime

- `tz-aware`: Used in the formatted string.
- `tz-naive`: Assumed to be in UTC.

Returns

str The returned format is a fixed-length subset of that defined by RFC 1123 and is the preferred format for use in the HTTP Date header. E.g.:

```
Sat, 02 Jan 1999 03:04:05 GMT
```

See also:

- <http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3.1>

`dl_common.date_time.xsd_datetime_str_from_dt(dt)`

Format datetime to a xs:dateTime string.

Parameters `dt` – datetime

- `tz-aware`: Used in the formatted string.
- `tz-naive`: Assumed to be in UTC.

Returns

str The returned format can be used as the date in xs:dateTime XML elements. It will be on the form `YYYY-MM-DDTHH:MM:SS.mmm+00:00`.

`dl_common.date_time.dt_from_http_datetime_str(http_full_datetime)`

Parse HTTP Full Date formats and return as datetime.

Parameters `http_full_datetime` – str Each of the allowed formats are supported:

- Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
- Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
- Sun Nov 6 08:49:37 1994 ; ANSI C's `asctime()` format

HTTP Full Dates are always in UTC.

Returns

datetime The returned datetime is always timezone aware and in UTC.

See also:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3.1>

`dl_common.date_time.dt_from_iso8601_str(iso8601_str)`

Parse ISO8601 formatted datetime string.

Parameters `iso8601_str` – str ISO 8601 formatted datetime.

- `tz-aware`: Used in the formatted string.
- `tz-naive`: Assumed to be in UTC.
- Partial strings are accepted as long as they're on the general form. Everything from just 2014 to 2006-10-20T15:34:56.123+02:30 will work. The sections that are not present in the string are set to zero in the returned datetime.
- See `test_iso8601.py` in the `iso8601` package for examples.

Returns

datetime The returned datetime is always timezone aware and in UTC.

Raises `d1_common.date_time.iso8601.ParseError` – If `“iso8601_string”` is not on the general form of ISO 8601.

`d1_common.date_time.normalize_datetime_to_utc(dt)`

Adjust datetime to UTC.

Apply the timezone offset to the datetime and set the timezone to UTC.

This is a no-op if the datetime is already in UTC.

Parameters `dt` – datetime - tz-aware: Used in the formatted string. - tz-naive: Assumed to be in UTC.

Returns

datetime The returned datetime is always timezone aware and in UTC.

Notes

This forces a new object to be returned, which fixes an issue with serialization to XML in PyXB. PyXB uses a mixin together with datetime to handle the XML `xs:dateTime`. That type keeps track of timezone information included in the original XML doc, which conflicts if we return it here as part of a datetime mixin.

See also:

`cast_naive_datetime_to_tz()`

`d1_common.date_time.cast_naive_datetime_to_tz(dt, tz=UTC)`

If datetime is tz-naive, set it to `tz`. If datetime is tz-aware, return it unmodified.

Parameters

- `dt` – datetime tz-naive or tz-aware datetime.
- `tz` – datetime.tzinfo The timezone to which to adjust tz-naive datetime.

Returns

datetime tz-aware datetime.

Warning: This will change the actual moment in time that is represented if the datetime is naive and represents a date and time not in `tz`.

See also:

`normalize_datetime_to_utc()`

`d1_common.date_time.strip_timezone(dt)`

Make datetime tz-naive by stripping away any timezone information.

Parameters

- `dt` – datetime
- - `tz-aware` – Used in the formatted string.
- - `tz-naive` – Returned unchanged.

Returns

datetime tz-naive datetime.


```
dl_common.date_time.utc_now()
```

Returns: tz-aware datetime: The current local date and time adjusted to the UTC timezone.

Notes

- Local time is retrieved from the local machine clock.
- Relies on correctly set timezone on the local machine.
- Relies on current tables for Daylight Saving periods.
- Local machine timezone can be checked with: `$ date +%z %Z`.

```
dl_common.date_time.date_utc_now_iso()
```

Returns:

str [The current local date as an ISO 8601 string in the UTC timezone] Does not include the time.

```
dl_common.date_time.local_now()
```

Returns:

tz-aware datetime : The current local date and time in the local timezone

```
dl_common.date_time.local_now_iso()
```

Returns:

str : The current local date and time as an ISO 8601 string in the local timezone

```
dl_common.date_time.to_iso8601_utc(dt)
```

Args: dt: datetime.

Returns: str: ISO 8601 string in the UTC timezone

```
dl_common.date_time.create_utc_datetime(*datetime_parts)
```

Create a datetime with timezone set to UTC.

Parameters tuple of int – year, month, day, hour, minute, second, microsecond

Returns datetime

```
dl_common.date_time.round_to_nearest(dt, n_round_sec=1.0)
```

Round datetime up or down to nearest divisor.

Round datetime up or down to nearest number of seconds that divides evenly by the divisor.

Any timezone is preserved but ignored in the rounding.

Parameters

- **dt** – datetime
- **n_round_sec** – int or float Divisor for rounding

Examples

- `n_round_sec = 0.1`: nearest 10th of a second.
- `n_round_sec = 1`: nearest second.
- `n_round_sec = 30`: nearest half minute.

d1_common.env module

Utilities for handling DataONE environments.

`d1_common.env.get_d1_env_keys()`
Get the DataONE env dict keys in preferred order.

Returns DataONE env dict keys

Return type list

`d1_common.env.get_d1_env(env_key)`
Get the values required in order to connect to a DataONE environment.

Returns Values required in order to connect to a DataONE environment.

Return type dict

`d1_common.env.get_d1_env_by_base_url(cn_base_url)`
Given the BaseURL for a CN, return the DataONE environment dict for the CN's environment.

d1_common.logging_context module

Context manager that enables temporary changes in logging level.

Source: <https://docs.python.org/2/howto/logging-cookbook.html>

class `d1_common.logging_context.LoggingContext` (*logger, level=None, handler=None, close=True*)

Bases: object

Logging Context Manager.

__init__ (*logger, level=None, handler=None, close=True*)
Args: *logger*: logger Logger for which to change the logging level.

level: Temporary logging level.

handler: Optional logging handler to use. Supplying a new handler allows temporarily changing the logging format as well.

close: Automatically close handler (if supplied).

d1_common.multipart module

Utilities for handling MIME Multipart documents.

`d1_common.multipart.parse_response(response, encoding='utf-8')`
Parse a multipart `Requests.Response` into a tuple of `BodyPart` objects.

Parameters

- **response** – `Requests.Response`
- **encoding** – The parser will assume that any text in the HTML body is encoded with this encoding when decoding it for use in the `text` attribute.

Returns

tuple of BodyPart Members: `headers` (`CaseInsensitiveDict`), `content` (bytes), `text` (Unicode), `encoding` (str).

`d1_common.multipart.parse_str(mmp_bytes, content_type, encoding='utf-8')`

Parse multipart document bytes into a tuple of BodyPart objects.

Parameters

- **mmp_bytes** – bytes Multipart document.
- **content_type** – str Must be on the form, multipart/form-data; boundary=<BOUNDARY>, where <BOUNDARY> is the string that separates the parts of the multipart document in mmp_bytes. In HTTP requests and responses, it is passed in the Content-Type header.
- **encoding** – str The coding used for the text in the HTML body.

Returns

tuple of BodyPart Members: headers (CaseInsensitiveDict), content (bytes), text (Unicode), encoding (str).

`d1_common.multipart.normalize(body_part_tup)`

Normalize a tuple of BodyPart objects to a string.

Normalization is done by sorting the body_parts by the Content- Disposition headers, which is typically on the form, form-data; name="name_of_part".

`d1_common.multipart.is_multipart(header_dict)`

Parameters **header_dict** – CaseInsensitiveDict

Returns True if header_dict has a Content-Type key (case insensitive) with value that begins with 'multipart'.

Return type bool

d1_common.node module

Utilities for handling the DataONE Node and NodeList types.

`d1_common.node.pyxb_to_dict(node_list_pyxb)`

Returns Representation of node_list_pyxb, keyed on the Node identifier (urn:node: *).

Return type dict

Example:

```
{
  u'urn:node:ARCTIC': {
    'base_url': u'https://arcticdata.io/metacat/d1/mn',
    'description': u'The US National Science Foundation...',
    'name': u'Arctic Data Center',
    'ping': None,
    'replicate': 0,
    'state': u'up',
    'synchronize': 1,
    'type': u'mn'
  },
  u'urn:node:BCODMO': {
    'base_url': u'https://www.bco-dmo.org/d1/mn',
    'description': u'Biological and Chemical Oceanography Data...',
    'name': u'Biological and Chemical Oceanography Data...',
    'ping': None,
```

(continues on next page)

(continued from previous page)

```

    'replicate': 0,
    'state': u'up',
    'synchronize': 1,
    'type': u'mn'
  },
}

```

d1_common.object_format_cache module

Local cache of the DataONE ObjectFormatList for a given DataONE environment.

As part of the metadata for a science object, DataONE stores a type identifier called an ObjectFormatID. The ObjectFormatList allows mapping ObjectFormatIDs to filename extensions and content type.

The cache is stored in a file and is automatically updated periodically.

Simple methods for looking up elements of the ObjectFormatList are provided.

Examples

Section of an ObjectFormatList:

```

{
  '-//ecoinformatics.org/eml-access-2.0.0beta4//EN': { 'extension': 'xml', 'format_name': 'Ecological Metadata Language, Access module, version 2.0.0beta4', 'format_type': 'METADATA', 'media_type': {
    'name': 'text/xml', 'property_list': []
  }
}, '-//ecoinformatics.org/eml-access-2.0.0beta6//EN': {
  'extension': 'xml', 'format_name': 'Ecological Metadata Language, Access module, version 2.0.0beta6', 'format_type': 'METADATA', 'media_type': {
    'name': 'text/xml', 'property_list': []
  }
},
}

```

```

class d1_common.object_format_cache.Singleton
    Bases: object

```

```

class d1_common.object_format_cache.ObjectFormatListCache (cn_base_url='https://cn.dataone.org/cn',
    object_format_cache_path='/home/docs/checkouts/readthedocs.org/user_
python/checkouts/latest/lib_common/src/d1_common/object_format_cache.json',
    cache_refresh_period=datetime.timedelta(days=30), lock_file_path='/tmp/object_format_cache.lock')
    Bases: d1_common.object_format_cache.Singleton

```

```

__init__ (cn_base_url='https://cn.dataone.org/cn', object_format_cache_path='/home/docs/checkouts/readthedocs.org/user_
python/checkouts/latest/lib_common/src/d1_common/object_format_cache.json',
    cache_refresh_period=datetime.timedelta(days=30), lock_file_path='/tmp/object_format_cache.lock')

```

Parameters

- **cn_base_url** – str: BaseURL for a CN in the DataONE Environment being targeted.

This can usually be left at the production root, even if running in other environments.

- **object_format_cache_path** – str Path to a file in which the cached ObjectFormatList is or will be stored.

By default, the path is set to a cache file that is distributed together with this module.

The directories must exist. The file is created if it doesn't exist. The file is recreated whenever needed. Paths under “/tmp” will typically cause the file to have to be recreated after reboot while paths under “/var/tmp/” typically persist over reboot.

- **cache_refresh_period** – datetime.timedelta or None Period of time in which to use the cached ObjectFormatList before refreshing it by downloading a new copy from the CN. The ObjectFormatList does not change often, so a month is probably a sensible default.

Set to None to disable refresh. When refresh is disabled, `object_format_cache_path` must point to an existing file.

property object_format_dict

Direct access to a native Python dict representing cached ObjectFormatList.

get_content_type (*format_id*, *default=None*)

get_filename_extension (*format_id*, *default=None*)

refresh_cache ()

Force a refresh of the local cached version of the ObjectFormatList.

This is typically not required, as the cache is refreshed automatically after the configured `cache_expiration_period`.

is_valid_format_id (*format_id*)

d1_common.replication_policy module

Utilities for handling the DataONE ReplicationPolicy type.

The Replication Policy is an optional section of the System Metadata which may be used to enable or disable replication, set the desired number of replicas and specify remote MNs to either prefer or block as replication targets.

Examples:

```
ReplicationPolicy:

<replicationPolicy replicationAllowed="true" numberReplicas="3">
  <!--Zero or more repetitions:-->
  <preferredMemberNode>node1</preferredMemberNode>
  <preferredMemberNode>node2</preferredMemberNode>
  <preferredMemberNode>node3</preferredMemberNode>
  <!--Zero or more repetitions:-->
  <blockedMemberNode>node4</blockedMemberNode>
  <blockedMemberNode>node5</blockedMemberNode>
</replicationPolicy>
```

`d1_common.replication_policy.has_replication_policy` (*sysmeta_pyxb*)

Args: *sysmeta_pyxb*: SystemMetadata PyXB object.

Returns: bool: True if SystemMetadata includes the optional ReplicationPolicy section.

```
d1_common.replication_policy.sysmeta_add_preferred(sysmeta_pyxb, node_urn)
```

Add a remote Member Node to the list of preferred replication targets to this System Metadata object.

Also remove the target MN from the list of blocked Member Nodes if present.

If the target MN is already in the preferred list and not in the blocked list, this function is a no-op.

Parameters

- **sysmeta_pyxb** – SystemMetadata PyXB object. System Metadata in which to add the preferred replication target.

If the System Metadata does not already have a Replication Policy, a default replication policy which enables replication is added and populated with the preferred replication target.

- **node_urn** –

str Node URN of the remote MN that will be added. On the form

urn:node:MyMemberNode.

```
d1_common.replication_policy.sysmeta_add_blocked(sysmeta_pyxb, node_urn)
```

Add a remote Member Node to the list of blocked replication targets to this System Metadata object.

The blocked node will not be considered a possible replication target for the associated System Metadata.

Also remove the target MN from the list of preferred Member Nodes if present.

If the target MN is already in the blocked list and not in the preferred list, this function is a no-op.

Parameters

- **sysmeta_pyxb** – SystemMetadata PyXB object. System Metadata in which to add the blocked replication target.

If the System Metadata does not already have a Replication Policy, a default replication policy which enables replication is added and then populated with the blocked replication target.

- **node_urn** – **str** Node URN of the remote MN that will be added. On the form

urn:node:MyMemberNode.

```
d1_common.replication_policy.sysmeta_set_default_rp(sysmeta_pyxb)
```

Set a default, empty, Replication Policy.

This will clear any existing Replication Policy in the System Metadata.

The default Replication Policy disables replication and sets number of replicas to 0.

Parameters **sysmeta_pyxb** – SystemMetadata PyXB object. System Metadata in which to set a default Replication Policy.

```
d1_common.replication_policy.normalize(rp_pyxb)
```

Normalize a ReplicationPolicy PyXB type in place.

The preferred and blocked lists are sorted alphabetically. As blocked nodes override preferred nodes, and any node present in both lists is removed from the preferred list.

Parameters **rp_pyxb** – ReplicationPolicy PyXB object The object will be normalized in place.

```
d1_common.replication_policy.is_preferred(rp_pyxb, node_urn)
```

Parameters

- **rp_pyxb** – ReplicationPolicy PyXB object The object will be normalized in place.

- **node_urn** – str Node URN of the remote MN for which to check preference.

Returns

True if `node_urn` is a preferred replica target.

As blocked nodes override preferred nodes, return False if `node_urn` is in both lists.

Return type bool

```
d1_common.replication_policy.is_blocked(rp_pyxb, node_urn)
```

Parameters

- **rp_pyxb** – ReplicationPolicy PyXB object The object will be normalized in place.
- **node_urn** – str Node URN of the remote MN for which to check preference.

Returns

True if `node_urn` is a blocked replica target.

As blocked nodes override preferred nodes, return True if `node_urn` is in both lists.

Return type bool

```
d1_common.replication_policy.are_equivalent_pyxb(a_pyxb, b_pyxb)
```

Check if two ReplicationPolicy objects are semantically equivalent.

The ReplicationPolicy objects are normalized before comparison.

Parameters `a_pyxb, b_pyxb` – ReplicationPolicy PyXB objects to compare

Returns True if the resulting policies for the two objects are semantically equivalent.

Return type bool

```
d1_common.replication_policy.are_equivalent_xml(a_xml, b_xml)
```

Check if two ReplicationPolicy XML docs are semantically equivalent.

The ReplicationPolicy XML docs are normalized before comparison.

Parameters `a_xml, b_xml` – ReplicationPolicy XML docs to compare

Returns True if the resulting policies for the two objects are semantically equivalent.

Return type bool

```
d1_common.replication_policy.add_preferred(rp_pyxb, node_urn)
```

Add a remote Member Node to the list of preferred replication targets.

Also remove the target MN from the list of blocked Member Nodes if present.

If the target MN is already in the preferred list and not in the blocked list, this function is a no-op.

Parameters

- **rp_pyxb** – SystemMetadata PyXB object. Replication Policy in which to add the preferred replication target.
- **node_urn** – str Node URN of the remote MN that will be added. On the form `urn:node:MyMemberNode`.

```
d1_common.replication_policy.add_blocked(rp_pyxb, node_urn)
```

Add a remote Member Node to the list of blocked replication targets.

Also remove the target MN from the list of preferred Member Nodes if present.

If the target MN is already in the blocked list and not in the preferred list, this function is a no-op.

Parameters

- **rp_pyxb** – SystemMetadata PyXB object. Replication Policy in which to add the blocked replication target.
- **node_urn** – str Node URN of the remote MN that will be added. On the form `urn:node:MyMemberNode`.

`d1_common.replication_policy.pyxb_to_dict(rp_pyxb)`

Convert ReplicationPolicy PyXB object to a normalized dict.

Parameters **rp_pyxb** – ReplicationPolicy to convert.

Returns Replication Policy as normalized dict.

Return type dict

Example:

```
{
  'allowed': True,
  'num': 3,
  'blockedMemberNode': {'urn:node:NODE1', 'urn:node:NODE2', 'urn:node:NODE3'},
  'preferredMemberNode': {'urn:node:NODE4', 'urn:node:NODE5'},
}
```

`d1_common.replication_policy.dict_to_pyxb(rp_dict)`

Convert dict to ReplicationPolicy PyXB object.

Parameters **rp_dict** – Native Python structure representing a Replication Policy.

Example:

```
{
  'allowed': True,
  'num': 3,
  'blockedMemberNode': {'urn:node:NODE1', 'urn:node:NODE2', 'urn:node:NODE3'},
  'preferredMemberNode': {'urn:node:NODE4', 'urn:node:NODE5'},
}
```

Returns ReplicationPolicy PyXB object.

d1_common.resource_map module

Read and write DataONE OAI-ORE Resource Maps.

DataONE supports a system that allows relationships between Science Objects to be described. These relationships are stored in *OAI-ORE Resource Maps*.

This module provides functionality for the most common use cases when parsing and generating Resource Maps for use in DataONE.

For more information about how Resource Maps are used in DataONE, see:

<https://releases.dataone.org/online/api-documentation-v2.0.1/design/DataPackage.html>

Common RDF/XML namespaces:


```

dc: <http://purl.org/dc/elements/1.1/>
foaf: <http://xmlns.com/foaf/0.1/>
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns# >
rdfs1: <http://www.w3.org/2001/01/rdf-schema# >
ore: <http://www.openarchives.org/ore/terms/>
dcterms: <http://purl.org/dc/terms/>
cito: <http://purl.org/spar/cito/>

```

Note: In order for Resource Maps to be recognized and indexed by DataONE, they must be created with `formatId` set to `http://www.openarchives.org/ore/terms`.

`dl_common.resource_map.createSimpleResourceMap(ore_pid, scimeta_pid, sciobj_pid_list)`

Create a simple OAI-ORE Resource Map with one Science Metadata document and any number of Science Data objects.

This creates a document that establishes an association between a Science Metadata object and any number of Science Data objects. The Science Metadata object contains information that is indexed by DataONE, allowing both the Science Metadata and the Science Data objects to be discoverable in DataONE Search. In search results, the objects will appear together and can be downloaded as a single package.

Parameters

- **ore_pid** – str Persistent Identifier (PID) to use for the new Resource Map
- **scimeta_pid** – str PID for an object that will be listed as the Science Metadata that is describing the Science Data objects.
- **sciobj_pid_list** – list of str List of PIDs that will be listed as the Science Data objects that are being described by the Science Metadata.

Returns OAI-ORE Resource Map

Return type *ResourceMap*

`dl_common.resource_map.createResourceMapFromStream(in_stream, base_url='https://cn.dataone.org/cn')`

Create a simple OAI-ORE Resource Map with one Science Metadata document and any number of Science Data objects, using a stream of PIDs.

Parameters

- **in_stream** – The first non-blank line is the PID of the resource map itself. Second line is the science metadata PID and remaining lines are science data PIDs.

Example stream contents:

```

PID_ORE_value
sci_meta_pid_value
data_pid_1
data_pid_2
data_pid_3

```

- **base_url** – str Root of the DataONE environment in which the Resource Map will be used.

Returns OAI-ORE Resource Map

Return type *ResourceMap*

```
class dl_common.resource_map.ResourceMap (ore_pid=None,          scimeta_pid=None,
                                           scidata_pid_list=None,
                                           base_url='https://cn.dataone.org/cn',
                                           api_major=2,  ore_software_id='DataONE.org
Python ITK 3.4.5', *args, **kwargs)
```

Bases: `rdflib.graph.ConjunctiveGraph`

OAI-ORE Resource Map.

```
__init__ (ore_pid=None,          scimeta_pid=None,          scidata_pid_list=None,
          base_url='https://cn.dataone.org/cn',  api_major=2,  ore_software_id='DataONE.org
Python ITK 3.4.5', *args, **kwargs)
```

Create a OAI-ORE Resource Map.

Parameters

- **ore_pid** – str Persistent Identifier (PID) to use for the new Resource Map
- **scimeta_pid** – str PID for an object that will be listed as the Science Metadata that is describing the Science Data objects.
- **scidata_pid_list** – list of str List of PIDs that will be listed as the Science Data objects that are being described by the Science Metadata.
- **base_url** – str Root of the DataONE environment in which the Resource Map will be used.
- **api_major** – The DataONE API version to use for the the DataONE Resolve API. Clients call the Resolve API to get a list of download locations for the objects in the Resource Map.
- **ore_software_id** – str Optional string which identifies the software that was used for creating the Resource Map. If specified, should be on the form of a UserAgent string.
- **args and kwargs** – Optional arguments forwarded to `rdflib.ConjunctiveGraph.__init__()`.

```
initialize (pid, ore_software_id='DataONE.org Python ITK 3.4.5')
```

Create the basic ORE document structure.

```
serialize_to_transport (doc_format='xml', *args, **kwargs)
```

Serialize ResourceMap to UTF-8 encoded XML document.

Parameters

- **doc_format** – str One of: `xml`, `n3`, `turtle`, `nt`, `pretty-xml`, `trix`, `trig` and `nquads`.
- **args and kwargs** – Optional arguments forwarded to `rdflib.ConjunctiveGraph.serialize()`.

Returns UTF-8 encoded XML doc.

Return type bytes

Note: Only the default, “xml”, is automatically indexed by DataONE.

```
serialize_to_display (doc_format='pretty-xml', *args, **kwargs)
```

Serialize ResourceMap to an XML doc that is pretty printed for display.

Parameters

- **doc_format** – str One of: xml, n3, turtle, nt, pretty-xml, trix, trig and nquads.
- **args and kwargs** – Optional arguments forwarded to `rdflib.ConjunctiveGraph.serialize()`.

Returns Pretty printed Resource Map XML doc

Return type str

Note: Only the default, “xml”, is automatically indexed by DataONE.

deserialize (**args, **kwargs*)

Deserialize Resource Map XML doc.

The source is specified using one of source, location, file or data.

Parameters

- **source** – InputSource, file-like object, or string In the case of a string the string is the location of the source.
- **location** – str String indicating the relative or absolute URL of the source. Graph’s absolutize method is used if a relative location is specified.
- **file** – file-like object
- **data** – str The document to be parsed.
- **format** – str Used if format can not be determined from source. Defaults to `rdflib/xml`. Format support can be extended with plugins.
Built-in: `xml, n3, nt, trix, rdfs`
- **publicID** – str Logical URI to use as the document base. If None specified the document location is used (at least in the case where there is a document location).

Raises `xml.sax.SAXException` based exception – On parse error.

getAggregation ()

Returns:

str : URIRef of the Aggregation entity

getObjectByPid (*pid*)

Parameters *pid* – str

Returns URIRef of the entry identified by *pid*.

Return type str

addResource (*pid*)

Add a resource to the Resource Map.

Parameters *pid* – str

setDocuments (*documenting_pid, documented_pid*)

Add a CiTO, the Citation Typing Ontology, triple asserting that *documenting_pid* documents *documented_pid*.

Adds assertion: *documenting_pid* cito:documents *documented_pid*

Parameters

- **documenting_pid** – str PID of a Science Object that documents `documented_pid`.
- **documented_pid** – str PID of a Science Object that is documented by `documenting_pid`.

setDocumentedBy (*documented_pid, documenting_pid*)

Add a CiTO, the Citation Typing Ontology, triple asserting that `documented_pid` isDocumentedBy `documenting_pid`.

Adds assertion: `documented_pid cito:isDocumentedBy documenting_pid`

Parameters

- **documented_pid** – str PID of a Science Object that is documented by `documenting_pid`.
- **documenting_pid** – str PID of a Science Object that documents `documented_pid`.

addMetadataDocument (*pid*)

Add a Science Metadata document.

Parameters **pid** – str PID of a Science Metadata object.

addDataDocuments (*scidata_pid_list, scimeta_pid=None*)

Add Science Data object(s)

Parameters

- **scidata_pid_list** – list of str List of one or more PIDs of Science Data objects
- **scimeta_pid** – str PID of a Science Metadata object that documents the Science Data objects.

getResourceMapPid ()

Returns:

str : PID of the Resource Map itself.

getAllTriples ()

Returns:

list of tuples : Each tuple holds a subject, predicate, object triple

getAllPredicates ()

Returns: list of str: All unique predicates.

Notes

Equivalent SPARQL:

```
SELECT DISTINCT ?p
WHERE {
  ?s ?p ?o .
}
```

getSubjectObjectsByPredicate (*predicate*)

Parameters **predicate** – str Predicate for which to return subject, object tuples.

Returns All subject/objects with `predicate`.

Return type list of subject, object tuples

Notes

Equivalent SPARQL:

```
SELECT DISTINCT ?s ?o
WHERE {{
    ?s {0} ?o .
}}
```

getAggregatedPids()

Returns: list of str: All aggregated PIDs.

Notes

Equivalent SPARQL:

```
SELECT ?pid
WHERE {
    ?s ore:aggregates ?o .
    ?o dcterms:identifier ?pid .
}
```

getAggregatedScienceMetadataPids()

Returns: list of str: All Science Metadata PIDs.

Notes

Equivalent SPARQL:

```
SELECT DISTINCT ?pid
WHERE {
    ?s ore:aggregates ?o .
    ?o cito:documents ?o2 .
    ?o dcterms:identifier ?pid .
}
```

getAggregatedScienceDataPids()

Returns: list of str: All Science Data PIDs.

Notes

Equivalent SPARQL:

```
SELECT DISTINCT ?pid
WHERE {
    ?s ore:aggregates ?o .
    ?o cito:isDocumentedBy ?o2 .
    ?o dcterms:identifier ?pid .
}
```

asGraphvizDot(stream)

Serialize the graph to .DOT format for ingestion in Graphviz.

Args: stream: file-like object open for writing that will receive the resulting document.

parseDoc (*doc_str, format='xml'*)
Parse a OAI-ORE Resource Maps document.

See Also: `rdflib.ConjunctiveGraph.parse` for documentation on arguments.

d1_common.revision module

Utilities for working with revision / obsolescence chains.

`d1_common.revision.get_identifiers` (*sysmeta_pyxb*)
Get set of identifiers that provide revision context for SciObj.

Returns: tuple: PID, SID, OBSOLETES_PID, OBSOLETE_BY_PID

`d1_common.revision.topological_sort` (*unsorted_dict*)
Sort objects by dependency.

Sort a dict of obsoleting PID to obsoleted PID to a list of PIDs in order of obsolescence.

Parameters *unsorted_dict* – dict Dict that holds obsolescence information. Each key/value pair establishes that the PID in *key* identifies an object that obsoletes an object identifies by the PID in *value*.

Returns

sorted_list: A list of PIDs ordered so that all PIDs that obsolete an object are listed after the object they obsolete.

unconnected_dict: A dict of PID to obsoleted PID of any objects that could not be added to a revision chain. These items will have obsoletes PIDs that directly or indirectly reference a PID that could not be sorted.

Return type tuple of *sorted_list*, *unconnected_dict*

Notes

obsoletes_dict is modified by the sort and on return holds any items that could not be sorted.

The sort works by repeatedly iterating over an unsorted list of PIDs and moving PIDs to the sorted list as they become available. A PID is available to be moved to the sorted list if it does not obsolete a PID or if the PID it obsoletes is already in the sorted list.

`d1_common.revision.get_pids_in_revision_chain` (*client, did*)

Args: *client*: `d1_client.cnclient.CoordinatingNodeClient` or `d1_client.mnclient.MemberNodeClient`.

did [str] SID or a PID of any object in a revision chain.

Returns All PIDs in the chain. The returned list is in the same order as the chain. The initial PID is typically obtained by resolving a SID. If the given PID is not in a chain, a list containing the single object is returned.

Return type list of str

`d1_common.revision.revision_list_to_obsoletes_dict` (*revision_list*)

Args: *revision_list*: list of tuple tuple: PID, SID, OBSOLETES_PID, OBSOLETE_BY_PID.

Returns: dict: Dict of obsoleted PID to obsoleting PID.

`d1_common.revision.revision_list_to_obsoleted_by_dict` (*revision_list*)

Args: *revision_list*: list of tuple tuple: PID, SID, OBSOLETE_PID, OBSOLETE_BY_PID.

Returns: dict: Dict of obsoleting PID to obsoleted PID.

d1_common.system_metadata module

Utilities for handling the DataONE SystemMetadata type.

DataONE API methods such as *MNStorage.create()* require a Science Object and System Metadata pair.

Examples

Example v2 SystemMetadata XML document with all optional values included:

```
<v2:systemMetadata xmlns:v2="http://ns.dataone.org/service/types/v2.0">
  <!--Optional:-->
  <serialVersion>11</serialVersion>

  <identifier>string</identifier>
  <formatId>string</formatId>
  <size>11</size>
  <checksum algorithm="string">string</checksum>

  <!--Optional:-->
  <submitter>string</submitter>
  <rightsHolder>string</rightsHolder>

  <!--Optional:-->
  <accessPolicy>
    <!--1 or more repetitions:-->
    <allow>
      <!--1 or more repetitions:-->
      <subject>string</subject>
      <!--1 or more repetitions:-->
      <permission>read</permission>
    </allow>
  </accessPolicy>

  <!--Optional:-->
  <replicationPolicy replicationAllowed="true" numberReplicas="3">
    <!--Zero or more repetitions:-->
    <preferredMemberNode>string</preferredMemberNode>
    <!--Zero or more repetitions:-->
    <blockedMemberNode>string</blockedMemberNode>
  </replicationPolicy>

  <!--Optional:-->
  <obsoletes>string</obsoletes>
  <obsoletedBy>string</obsoletedBy>
  <archived>true</archived>
  <dateUploaded>2014-09-18T17:18:33</dateUploaded>
  <dateSysMetadataModified>2006-08-19T11:27:14-06:00</dateSysMetadataModified>
  <originMemberNode>string</originMemberNode>
  <authoritativeMemberNode>string</authoritativeMemberNode>
```

(continues on next page)

(continued from previous page)

```

<!--Zero or more repetitions:-->
<replica>
  <replicaMemberNode>string</replicaMemberNode>
  <replicationStatus>failed</replicationStatus>
  <replicaVerified>2013-05-21T19:02:49-06:00</replicaVerified>
</replica>

<!--Optional:-->
<seriesId>string</seriesId>

<!--Optional:-->
<mediaType name="string">
  <!--Zero or more repetitions:-->
  <property name="string">string</property>
</mediaType>

<!--Optional:-->
<fileName>string</fileName>
</v2:systemMetadata>

```

`dl_common.system_metadata.is_sysmeta_pyxb` (*sysmeta_pyxb*)

Args: *sysmeta_pyxb*: Object that may or may not be a SystemMetadata PyXB object.

Returns

- True if *sysmeta_pyxb* is a SystemMetadata PyXB object.
- False if *sysmeta_pyxb* is not a PyXB object or is a PyXB object of a type other than SystemMetadata.

Return type

bool

`dl_common.system_metadata.normalize_in_place` (*sysmeta_pyxb*, *reset_timestamps=False*, *reset_filename=False*)

Normalize SystemMetadata PyXB object in-place.

Parameters

- ***sysmeta_pyxb*** – SystemMetadata PyXB object to normalize.
- ***reset_timestamps*** – bool True: Timestamps in the SystemMetadata are set to a standard value so that objects that are compared after normalization register as equivalent if only their timestamps differ.

Notes

The SystemMetadata is normalized by removing any redundant information and ordering all sections where there are no semantics associated with the order. The normalized SystemMetadata is intended to be semantically equivalent to the un-normalized one.

`dl_common.system_metadata.are_equivalent_pyxb` (*a_pyxb*, *b_pyxb*, *ignore_timestamps=False*, *ignore_filename=False*)

Determine if SystemMetadata PyXB objects are semantically equivalent.

Normalize then compare SystemMetadata PyXB objects for equivalency.

Parameters

- ***a_pyxb*, *b_pyxb*** – SystemMetadata PyXB objects to compare

- **ignore_timestamps** – bool `True`: Timestamps are ignored during the comparison.
- **ignore_filename** – bool `True`: FileName elements are ignored during the comparison.

This is necessary in cases where GMN returns a generated filename because one was not provided in the SysMeta.

Returns `True` if SystemMetadata PyXB objects are semantically equivalent.

Return type bool

Notes

The SystemMetadata is normalized by removing any redundant information and ordering all sections where there are no semantics associated with the order. The normalized SystemMetadata is intended to be semantically equivalent to the un-normalized one.

`d1_common.system_metadata.are_equivalent_xml(a_xml, b_xml, ignore_timestamps=False)`

Determine if two SystemMetadata XML docs are semantically equivalent.

Normalize then compare SystemMetadata XML docs for equivalency.

Parameters

- **a_xml, b_xml** – bytes UTF-8 encoded SystemMetadata XML docs to compare
- **ignore_timestamps** – bool `True`: Timestamps in the SystemMetadata are ignored so that objects that are compared register as equivalent if only their timestamps differ.

Returns `True` if SystemMetadata XML docs are semantically equivalent.

Return type bool

Notes

The SystemMetadata is normalized by removing any redundant information and ordering all sections where there are no semantics associated with the order. The normalized SystemMetadata is intended to be semantically equivalent to the un-normalized one.

`d1_common.system_metadata.clear_elements(sysmeta_pyxb, clear_replica=True, clear_serial_version=True)`

{clear_replica} causes any replica information to be removed from the object.

{clear_replica} ignores any differences in replica information, as this information is often different between MN and CN.

`d1_common.system_metadata.update_elements(dst_pyxb, src_pyxb, el_list)`

Copy elements specified in `el_list` from `src_pyxb` to `dst_pyxb`

Only elements that are children of root are supported. See `SYSMETA_ROOT_CHILD_LIST`.

If an element in `el_list` does not exist in `src_pyxb`, it is removed from `dst_pyxb`.

```
d1_common.system_metadata.generate_system_metadata_pyxb(pid, format_id,
sciobj_stream, submitter_str, rights_holder_str,
authoritative_mn_urn,
sid=None, obsoletes_pid=None,
obsoleted_by_pid=None,
is_archived=False,
serial_version=1, uploaded_datetime=None,
modified_datetime=None,
file_name=None, origin_mn_urn=None,
is_private=False, access_list=None,
media_name=None, media_property_list=None,
is_replication_allowed=False,
preferred_mn_list=None,
blocked_mn_list=None,
pyxb_binding=None)
```

Generate a System Metadata PyXB object

Parameters

- **pid**
- **format_id**
- **sciobj_stream**
- **submitter_str**
- **rights_holder_str**
- **authoritative_mn_urn**
- **pyxb_binding**
- **sid**
- **obsoletes_pid**
- **obsoleted_by_pid**
- **is_archived**
- **serial_version**
- **uploaded_datetime**
- **modified_datetime**
- **file_name**
- **origin_mn_urn**
- **access_list**
- **is_private**
- **media_name**
- **media_property_list**

- **is_replication_allowed**
- **preferred_mn_list**
- **blocked_mn_list**

Returns systemMetadata PyXB object

```
d1_common.system_metadata.gen_checksum_and_size(sciobj_stream)
d1_common.system_metadata.gen_access_policy(pyxb_binding, sysmeta_pyxb, is_private, access_list)
d1_common.system_metadata.gen_replication_policy(pyxb_binding, preferred_mn_list=None, blocked_mn_list=None, is_replication_allowed=False)
d1_common.system_metadata.gen_media_type(pyxb_binding, media_name, media_property_list=None)
```

d1_common.type_conversions module

Utilities for handling the DataONE types.

- Handle conversions between XML representations used in the D1 Python stack.
- Handle conversions between v1 and v2 DataONE XML types.

The DataONE Python stack uses the following representations for the DataONE API XML docs:

- As native Unicode `str`, typically “pretty printed” with indentations, when formatted for display.
- As UTF-8 encoded `bytes` when send sending or receiving over the network, or loading or saving as files.
- Schema validation and manipulation in Python code as PyXB binding objects.
- General processing as ElementTrees.

In order to allow conversions between all representations without having to implement separate conversions for each combination of input and output representation, a “hub and spokes” model is used. Native Unicode `str` was selected as the “hub” representation due to:

- PyXB provides translation to/from string and DOM.
- ElementTree provides translation to/from string.

```
d1_common.type_conversions.get_version_tag_by_pyxb_binding(pyxb_binding)
```

Map PyXB binding to DataONE API version.

Given a PyXB binding, return the API major version number.

Parameters `pyxb_binding` – PyXB binding object

Returns DataONE API major version number, currently, v1, 1, v2 or 2.

```
d1_common.type_conversions.get_pyxb_binding_by_api_version(api_major, api_minor=0)
```

Map DataONE API version tag to PyXB binding.

Given a DataONE API major version number, return PyXB binding that can serialize and deserialize DataONE XML docs of that version.

Parameters `api_major`, `api_minor` – str or int DataONE API major and minor version numbers.

- If `api_major` is an integer, it is combined with `api_minor` to form an exact version.

- If `api_major` is a string of v1 or v2, `api_minor` is ignored and the latest PyXB binding available for the `api_major` version is returned.

Returns E.g., `dl_common.types.dataoneTypes_v1_1`.

Return type PyXB binding

`dl_common.type_conversions.get_version_tag(api_major)`

Args:

`api_major`: int DataONE API major version. Valid versions are currently 1 or 2. Returns: str: DataONE API version tag. Valid version tags are currently v1 or v2.

`dl_common.type_conversions.extract_version_tag_from_url(url)`

Extract a DataONE API version tag from a MN or CN service endpoint URL.

Parameters `url` – str Service endpoint URL. E.g.: `https://mn.example.org/path/v2/object/pid`.

Returns Valid version tags are currently v1 or v2.

Return type str

`dl_common.type_conversions.get_pyxb_namespaces()`

Returns:

list of str: XML namespaces currently known to PyXB

`dl_common.type_conversions.str_to_v1_str(xml_str)`

Convert a API v2 XML doc to v1 XML doc.

Removes elements that are only valid for v2 and changes namespace to v1.

If doc is already v1, it is returned unchanged.

Parameters `xml_str` – str API v2 XML doc. E.g.: `SystemMetadata v2`.

Returns API v1 XML doc. E.g.: `SystemMetadata v1`.

Return type str

`dl_common.type_conversions.pyxb_to_v1_str(pyxb_obj)`

Convert a API v2 PyXB object to v1 XML doc.

Removes elements that are only valid for v2 and changes namespace to v1.

Parameters `pyxb_obj` – PyXB object API v2 PyXB object. E.g.: `SystemMetadata v2_0`.

Returns API v1 XML doc. E.g.: `SystemMetadata v1`.

Return type str

`dl_common.type_conversions.str_to_v1_pyxb(xml_str)`

Convert a API v2 XML doc to v1 PyXB object.

Removes elements that are only valid for v2 and changes namespace to v1.

Parameters `xml_str` – str API v2 XML doc. E.g.: `SystemMetadata v2`.

Returns API v1 PyXB object. E.g.: `SystemMetadata v1_2`.

Return type PyXB object

`dl_common.type_conversions.str_to_v2_str(xml_str)`

Convert a API v1 XML doc to v2 XML doc.

All v1 elements are valid for v2, so only changes namespace.

Parameters `xml_str` – str API v1 XML doc. E.g.: `SystemMetadata v1`.

Returns API v2 XML doc. E.g.: `SystemMetadata v2`.

Return type str

`dl_common.type_conversions.pyxb_to_v2_str(pyxb_obj)`

Convert a API v1 PyXB object to v2 XML doc.

All v1 elements are valid for v2, so only changes namespace.

Parameters `pyxb_obj` – PyXB object API v1 PyXB object. E.g.: `SystemMetadata v1_0`.

Returns API v2 XML doc. E.g.: `SystemMetadata v2`.

Return type str

`dl_common.type_conversions.str_to_v2_pyxb(xml_str)`

Convert a API v1 XML doc to v2 PyXB object.

All v1 elements are valid for v2, so only changes namespace.

Parameters `xml_str` – str API v1 XML doc. E.g.: `SystemMetadata v1`.

Returns API v2 PyXB object. E.g.: `SystemMetadata v2_0`.

Return type PyXB object

`dl_common.type_conversions.is_pyxb(pyxb_obj)`

Returns:

bool: **True** if `pyxb_obj` is a PyXB object.

`dl_common.type_conversions.is_pyxb_dl_type(pyxb_obj)`

Returns:

bool: **True** if `pyxb_obj` is a PyXB object holding a DataONE API type.

`dl_common.type_conversions.is_pyxb_dl_type_name(pyxb_obj, expected_pyxb_type_name)`

Parameters

- `pyxb_obj` – object May be a PyXB object and may hold a DataONE API type.
- `expected_pyxb_type_name` – str Case sensitive name of a DataONE type.
E.g.: `SystemMetadata`, `LogEntry`, `ObjectInfo`.

Returns **True** if object is a PyXB object holding a value of the specified type.

Return type bool

`dl_common.type_conversions.pyxb_get_type_name(obj_pyxb)`

Args: `obj_pyxb`: PyXB object.

Returns

Name of the type the PyXB object is holding.

E.g.: `SystemMetadata`, `LogEntry`, `ObjectInfo`.

Return type str

`dl_common.type_conversions.pyxb_get_namespace_name(obj_pyxb)`

Args: `obj_pyxb`: PyXB object.

Returns

Namespace and Name of the type the PyXB object is holding.

E.g.: {http://ns.dataone.org/service/types/v2.0}SystemMetadata

Return type str

`d1_common.type_conversions.str_is_v1(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is a DataONE API v1 type.

Return type bool

`d1_common.type_conversions.str_is_v2(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is a DataONE API v2 type.

Return type bool

`d1_common.type_conversions.str_is_error(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is a DataONE Exception type.

Return type bool

`d1_common.type_conversions.str_is_identifier(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is a DataONE Identifier type.

Return type bool

`d1_common.type_conversions.str_is_objectList(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is a DataONE ObjectList type.

Return type bool

`d1_common.type_conversions.str_is_well_formed(xml_str)`

Parameters `xml_str` – str DataONE API XML doc.

Returns True if XML doc is well formed.

Return type bool

`d1_common.type_conversions.pyxb_is_v1(pyxb_obj)`

Parameters `pyxb_obj` – PyXB object PyXB object holding an unknown type.

Returns True if `pyxb_obj` holds an API v1 type.

Return type bool

`d1_common.type_conversions.pyxb_is_v2(pyxb_obj)`

Parameters `pyxb_obj` – PyXB object PyXB object holding an unknown type.

Returns True if `pyxb_obj` holds an API v2 type.

Return type bool

`d1_common.type_conversions.str_to_pyxb(xml_str)`

Deserialize API XML doc to PyXB object.

Parameters `xml_str` – str DataONE API XML doc

Returns Matching the API version of the XML doc.

Return type PyXB object

`d1_common.type_conversions.str_to_etree(xml_str, encoding='utf-8')`
 Deserialize API XML doc to an ElementTree.

Parameters

- `xml_str` – bytes DataONE API XML doc
- `encoding` – str Decoder to use when converting the XML doc bytes to a Unicode str.

Returns Matching the API version of the XML doc.

Return type ElementTree

`d1_common.type_conversions.pyxb_to_str(pyxb_obj, encoding='utf-8')`
 Serialize PyXB object to XML doc.

Parameters

- `pyxb_obj` – PyXB object
- `encoding` – str Encoder to use when converting the Unicode strings in the PyXB object to XML doc bytes.

Returns API XML doc, matching the API version of `pyxb_obj`.

Return type str

`d1_common.type_conversions.etree_to_str(etree_obj, encoding='utf-8')`
 Serialize ElementTree to XML doc.

Parameters

- `etree_obj` – ElementTree
- `encoding` – str Encoder to use when converting the Unicode strings in the ElementTree to XML doc bytes.

Returns XML doc.

Return type str

`d1_common.type_conversions.etree_to_pretty_xml(etree_obj, encoding='unicode')`
 Serialize ElementTree to pretty printed XML doc.

Parameters

- `etree_obj` – ElementTree
- `encoding` – str Encoder to use when converting the Unicode strings in the ElementTree to XML doc bytes.

Returns Pretty printed XML doc.

Return type str

`d1_common.type_conversions.pyxb_to_etree(pyxb_obj)`
 Convert PyXB object to ElementTree.

Parameters `pyxb_obj` – PyXB object

Returns Matching the API version of the PyXB object.

Return type ElementTree

`d1_common.type_conversions.etree_to_pyxb(etree_obj)`

Convert ElementTree to PyXB object.

Parameters `etree_obj` – ElementTree

Returns Matching the API version of the ElementTree object.

Return type PyXB object

`d1_common.type_conversions.replace_namespace_with_prefix(tag_str,
ns_reverse_dict=None)`

Convert XML tag names with namespace on the form {namespace}tag to form prefix:tag.

Parameters

- **tag_str** – str Tag name with namespace. E.g.: {http://www.openarchives.org/ore/terms/}ResourceMap.
- **ns_reverse_dict** – dict A dictionary of namespace to prefix to use for the conversion. If not supplied, a default dict with the namespaces used in DataONE XML types is used.

Returns Tag name with prefix. E.g.: ore:ResourceMap.

Return type str

`d1_common.type_conversions.etree_replace_namespace(etree_obj, ns_str)`

In-place change the namespace of elements in an ElementTree.

Parameters

- **etree_obj** – ElementTree
- **ns_str** – str The namespace to set. E.g.: http://ns.dataone.org/service/types/v1.

`d1_common.type_conversions.strip_v2_elements(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types.

Args: `etree_obj`: ElementTree ElementTree holding one of the DataONE API types that changed between v1 and v2.

`d1_common.type_conversions.strip_system_metadata(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types from v1 System Metadata.

Args: `etree_obj`: ElementTree ElementTree holding a v1 SystemMetadata.

`d1_common.type_conversions.strip_log(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types from v1 Log.

Args: `etree_obj`: ElementTree ElementTree holding a v1 Log.

`d1_common.type_conversions.strip_logEntry(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types from v1 LogEntry.

Args: `etree_obj`: ElementTree ElementTree holding a v1 LogEntry.

`d1_common.type_conversions.strip_node(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types from v1 Node.

Args: `etree_obj`: ElementTree ElementTree holding a v1 Node.

`d1_common.type_conversions.strip_node_list(etree_obj)`

In-place remove elements and attributes that are only valid in v2 types from v1 NodeList.

Args: `etree_obj`: ElementTree ElementTree holding a v1 NodeList.

`d1_common.type_conversions.v2_0_tag(element_name)`

Add a v2 namespace to a tag name.

Parameters `element_name` – str The name of a DataONE v2 type. E.g.: `NodeList`.

Returns The tag name with DataONE API v2 namespace. E.g.: `{http://ns.dataone.org/service/types/v2.0}NodeList`

Return type str

d1_common.url module

Utilities for handling URLs in DataONE.

`d1_common.url.parseUrl(url)`

Return a dict containing scheme, netloc, url, params, query, fragment keys.

query is a dict where the values are always lists. If the query key appears only once in the URL, the list will have a single value.

`d1_common.url.isHttpOrHttps(url)`

URL is HTTP or HTTPS protocol.

Upper and lower case protocol names are recognized.

`d1_common.url.encodePathElement(element)`

Encode a URL path element according to RFC3986.

`d1_common.url.decodePathElement(element)`

Decode a URL path element according to RFC3986.

`d1_common.url.encodeQueryElement(element)`

Encode a URL query element according to RFC3986.

`d1_common.url.decodeQueryElement(element)`

Decode a URL query element according to RFC3986.

`d1_common.url.stripElementSlashes(element)`

Strip any slashes from the front and end of an URL element.

`d1_common.url.joinPathElements(*elements)`

Join two or more URL elements, inserting '/' as needed.

Note: Any leading and trailing slashes are stripped from the resulting URL. An empty element ('') causes an empty spot in the path ('//').

`d1_common.url.encodeAndJoinPathElements(*elements)`

Encode URL path element according to RFC3986 then join them, inserting '/' as needed.

Note: Any leading and trailing slashes are stripped from the resulting URL. An empty element ('') causes an empty spot in the path ('//').

`d1_common.url.normalizeTarget(target)`

If necessary, modify target so that it ends with '/'.

`d1_common.url.urlencode(query, doseq=0)`

Modified version of the standard `urllib.urlencode` that conforms to RFC3986. The `urllib` version encodes spaces as '+' which can lead to inconsistency. This version will always encode spaces as '%20'.

Encode a sequence of two-element tuples or dictionary into a URL query string.

If any values in the query arg are sequences and `doseq` is true, each sequence element is converted to a separate parameter.

If the query arg is a sequence of two-element tuples, the order of the parameters in the output will match the order of parameters in the input.

`d1_common.url.makeCNBaseURL(url)`

Attempt to create a valid CN BaseURL when one or more sections of the URL are missing.

`d1_common.url.makeMNBaseURL(url)`

Attempt to create a valid MN BaseURL when one or more sections of the URL are missing.

`d1_common.url.find_url_mismatches(a_url, b_url)`

Given two URLs, return a list of any mismatches.

If the list is empty, the URLs are equivalent. Implemented by parsing and comparing the elements. See RFC 1738 for details.

`d1_common.url.is_urls_equivalent(a_url, b_url)`

d1_common.util module

General utilities often needed by DataONE clients and servers.

`d1_common.util.log_setup(is_debug=False, is_multiprocess=False)`

Set up a standardized log format for the DataONE Python stack. All Python components should use this function. If `is_multiprocess` is True, include process ID in the log so that logs can be separated for each process.

Output only to stdout and stderr.

`d1_common.util.get_content_type(content_type)`

Extract the MIME type value from a content type string.

Removes any subtype and parameter values that may be present in the string.

Parameters `content_type` – str String with content type and optional subtype and parameter fields.

Returns String with only content type

Return type str

Example:

```
Input:  multipart/form-data; boundary=aBoundaryString
Returns: multipart/form-data
```

`d1_common.util.nested_update(d, u)`

Merge two nested dicts.

Nested dicts are sometimes used for representing various recursive structures. When updating such a structure, it may be convenient to present the updated data as a corresponding recursive structure. This function will then apply the update.

Parameters

- **d** – dict dict that will be updated in-place. May or may not contain nested dicts.
- **u** – dict dict with contents that will be merged into d. May or may not contain nested dicts.

```
class d1_common.util.EventCounter(logger_=<module      'logging'      from
                                  '/home/docs/.pyenv/versions/3.7.3/lib/python3.7/logging/__init__.py'>)
```

Bases: object

Count events during a lengthy operation and write running totals and/or a summary to a logger when the operation has completed.

The summary contains the name and total count of each event that was counted.

Example

Summary written to the log:

```
Events:
Creating SciObj DB representations: 200
Retrieving revision chains: 200
Skipped Node registry update: 1
Updating obsoletedBy: 42
Whitelisted subject: 2
```

property event_dict

Provide direct access to the underlying dict where events are recorded.

Returns: dict: Events and event counts.

count (*event_str, inc_int=1*)

Count an event.

Parameters

- **event_str** – The name of an event to count. Used as a key in the event dict. The same name will also be used in the summary.
- **inc_int** – int Optional argument to increase the count for the event by more than 1.

log_and_count (*event_str, msg_str=None, inc_int=None*)

Count an event and write a message to a logger.

Parameters

- **event_str** – str The name of an event to count. Used as a key in the event dict. The same name will be used in the summary. This also becomes a part of the message logged by this function.
- **msg_str** – str Optional message with details about the events. The message is only written to the log. While the `event_str` functions as a key and must remain the same for the same type of event, `log_str` may change between calls.
- **inc_int** – int Optional argument to increase the count for the event by more than 1.

dump_to_log ()

Write summary to logger with the name and number of times each event has been counted.

This function may be called at any point in the process. Counts are not zeroed.

d1_common.util.print_logging ()

Context manager to temporarily suppress additional information such as timestamps when writing to loggers.

This makes logging look like `print()`. The main use case is in scripts that mix logging and `print()`, as Python uses separate streams for those, and output can and does end up getting shuffled if `print()` and logging is used interchangeably.

When entering the context, the logging levels on the current handlers are saved then modified to WARNING levels. A new DEBUG level handler with a formatter that does not write timestamps, etc, is then created.

When leaving the context, the DEBUG handler is removed and existing loggers are restored to their previous levels.

By modifying the log levels to WARNING instead of completely disabling the loggers, it is ensured that potentially serious issues can still be logged while the context manager is in effect.

`d1_common.util.save_json(py_obj, json_path)`

Serialize a native object to JSON and save it normalized, pretty printed to a file.

The JSON string is normalized by sorting any dictionary keys.

Parameters

- **py_obj** – object Any object that can be represented in JSON. Some types, such as date-times are automatically converted to strings.
- **json_path** – str File path to which to write the JSON file. E.g.: The path must exist. The filename will normally end with “.json”.

See also:

`ToJsonCompatibleTypes()`

`d1_common.util.load_json(json_path)`

Load JSON file and parse it to a native object.

Parameters `json_path` – str File path from which to load the JSON file.

Returns Typically a nested structure of `list` and `dict` objects.

Return type object

`d1_common.util.format_json_to_normalized_pretty_json(json_str)`

Normalize and pretty print a JSON string.

The JSON string is normalized by sorting any dictionary keys.

Parameters `json_str` – A valid JSON string.

Returns normalized, pretty printed JSON string.

Return type str

`d1_common.util.serialize_to_normalized_pretty_json(py_obj)`

Serialize a native object to normalized, pretty printed JSON.

The JSON string is normalized by sorting any dictionary keys.

Parameters `py_obj` – object Any object that can be represented in JSON. Some types, such as datetimes are automatically converted to strings.

Returns normalized, pretty printed JSON string.

Return type str

`d1_common.util.serialize_to_normalized_compact_json(py_obj)`

Serialize a native object to normalized, compact JSON.

The JSON string is normalized by sorting any dictionary keys. It will be on a single line without whitespace between elements.

Parameters `py_obj` – object Any object that can be represented in JSON. Some types, such as datetimes are automatically converted to strings.

Returns normalized, compact JSON string.

Return type str

```
class dl_common.util.ToJsonCompatibleTypes (*, skipkeys=False, ensure_ascii=True,
                                             check_circular=True, allow_nan=True,
                                             sort_keys=False, indent=None, separa-
                                             tors=None, default=None)
```

Bases: `json.encoder.JSONEncoder`

Some native objects such as `datetime.datetime` are not automatically converted to strings for use as values in JSON.

This helper adds such conversions for types that the DataONE Python stack encounters frequently in objects that are to be JSON encoded.

default (*o*)

Implement this method in a subclass such that it returns a serializable object for *o*, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`dl_common.util.format_sec_to_dhm(sec)`

Format seconds to days, hours, minutes.

Parameters `sec` – float or int Number of seconds in a period of time

Returns `00h:00m“`.

Return type Period of time represented as a string on the form `“0d`

dl_common.xml module

Utilities for handling XML docs.

`dl_common.xml.deserialize(doc_xml, pyxb_binding=None)`

Deserialize DataONE XML types to PyXB.

Parameters

- `doc_xml` – UTF-8 encoded bytes
- `pyxb_binding` – PyXB binding object. If not specified, the correct one should be
- **selected automatically.**

Returns PyXB object

See also:

`deserialize_dl_exception()` for deserializing DataONE Exception types.

`dl_common.xml.deserialize_dl_exception(doc_xml)`

Args: `doc_xml`: UTF-8 encoded bytes An XML doc that conforms to the dataoneErrors XML Schema.

Returns: DataONEException object

```
d1_common.xml.serialize_gen(obj_pyxb, encoding='utf-8', pretty=False, strip_prolog=False,
                             xslt_url=None)
```

Serialize PyXB object to XML.

Parameters

- **obj_pyxb** – PyXB object PyXB object to serialize.
- **encoding** – str Encoding to use for XML doc bytes
- **pretty** – bool True: Use pretty print formatting for human readability.
- **strip_prolog** – True: remove any XML prolog (e.g., `<?xml version="1.0" encoding="utf-8" ?>`), from the resulting XML doc.
- **xslt_url** – str If specified, add a processing instruction to the XML doc that specifies the download location for an XSLT stylesheet.

Returns XML document

```
d1_common.xml.serialize_for_transport(obj_pyxb, pretty=False, strip_prolog=False,
                                       xslt_url=None)
```

Serialize PyXB object to XML bytes with UTF-8 encoding for transport over the network, filesystem storage and other machine usage.

Parameters

- **obj_pyxb** – PyXB object PyXB object to serialize.
- **pretty** – bool True: Use pretty print formatting for human readability.
- **strip_prolog** – True: remove any XML prolog (e.g., `<?xml version="1.0" encoding="utf-8" ?>`), from the resulting XML doc.
- **xslt_url** – str If specified, add a processing instruction to the XML doc that specifies the download location for an XSLT stylesheet.

Returns UTF-8 encoded XML document

Return type bytes

See also:

```
serialize_for_display()
```

```
d1_common.xml.serialize_to_xml_str(obj_pyxb, pretty=True, strip_prolog=False,
                                   xslt_url=None)
```

Serialize PyXB object to pretty printed XML str for display.

Parameters

- **obj_pyxb** – PyXB object PyXB object to serialize.
- **pretty** – bool False: Disable pretty print formatting. XML will not have line breaks.
- **strip_prolog** – True: remove any XML prolog (e.g., `<?xml version="1.0" encoding="utf-8" ?>`), from the resulting XML doc.
- **xslt_url** – str If specified, add a processing instruction to the XML doc that specifies the download location for an XSLT stylesheet.

Returns Pretty printed XML document

Return type str

```
d1_common.xml.reformat_to_pretty_xml(doc_xml)
```

Pretty print XML doc.

Parameters `doc_xml` – str Well formed XML doc

Returns Pretty printed XML doc

Return type str

`d1_common.xml.are_equivalent_pyxb(a_pyxb, b_pyxb)`

Return True if two PyXB objects are semantically equivalent, else False.

`d1_common.xml.are_equivalent(a_xml, b_xml, encoding=None)`

Return True if two XML docs are semantically equivalent, else False.

- TODO: Include test for tails. Skipped for now because tails are not used in any D1 types.

`d1_common.xml.are_equal_or_superset(superset_tree, base_tree)`

Return True if `superset_tree` is equal to or a superset of `base_tree`

- Checks that all elements and attributes in `superset_tree` are present and contain the same values as in `base_tree`. For elements, also checks that the order is the same.
- Can be used for checking if one XML document is based on another, as long as all the information in `base_tree` is also present and unmodified in `superset_tree`.

`d1_common.xml.are_equal_xml(a_xml, b_xml)`

Normalize and compare XML documents for equality. The document may or may not be a DataONE type.

Parameters

- `a_xml` – str
- `b_xml` – str XML documents to compare for equality.

Returns True if the XML documents are semantically equivalent.

Return type bool

`d1_common.xml.are_equal_pyxb(a_pyxb, b_pyxb)`

Normalize and compare PyXB objects for equality.

Parameters

- `a_pyxb` – PyXB object
- `b_pyxb` – PyXB object PyXB objects to compare for equality.

Returns True if the PyXB objects are semantically equivalent.

Return type bool

`d1_common.xml.are_equal_elements(a_el, b_el)`

Normalize and compare ElementTrees for equality.

Parameters

- `a_el` – ElementTree
- `b_el` – ElementTree ElementTrees to compare for equality.

Returns True if the ElementTrees are semantically equivalent.

Return type bool

`d1_common.xml.sort_value_list_pyxb(obj_pyxb)`

In-place sort complex value siblings in a PyXB object.

Args: `obj_pyxb`: PyXB object

`dl_common.xml.sort_elements_by_child_values(obj_pyxb, child_name_list)`

In-place sort simple or complex elements in a PyXB object by values they contain in child elements.

Parameters

- **obj_pyxb** – PyXB object
- **child_name_list** – list of str List of element names that are direct children of the PyXB object.

`dl_common.xml.format_diff_pyxb(a_pyxb, b_pyxb)`

Create a diff between two PyXB objects.

Parameters

- **a_pyxb** – PyXB object
- **b_pyxb** – PyXB object

Returns *Differ*-style delta

Return type str

`dl_common.xml.format_diff_xml(a_xml, b_xml)`

Create a diff between two XML documents.

Parameters

- **a_xml** – str
- **b_xml** – str

Returns *Differ*-style delta

Return type str

`dl_common.xml.is_valid_utf8(o)`

Determine if object is valid UTF-8 encoded bytes.

Parameters **o** – str

Returns True if object is bytes containing valid UTF-8.

Return type bool

Notes

- An empty bytes object is valid UTF-8.
- Any type of object can be checked, not only bytes.

`dl_common.xml.get_auto(obj_pyxb)`

Return value from simple or complex PyXB element.

PyXB complex elements have a `.value()` member which must be called in order to retrieve the value of the element, while simple elements represent their values directly. This function allows retrieving element values without knowing the type of element.

Parameters **obj_pyxb** – PyXB object

Returns Value of the PyXB object.

Return type str

`dl_common.xml.get_opt_attr(obj_pyxb, attr_str, default_val=None)`

Get an optional attribute value from a PyXB element.

The attributes for elements that are optional according to the schema and not set in the PyXB object are present and set to None.

PyXB validation will fail if required elements are missing.

Parameters

- **obj_pyxb** – PyXB object
- **attr_str** – str Name of an attribute that the PyXB object may contain.
- **default_val** – any object Value to return if the attribute is not present.

Returns Value of the attribute if present, else `default_val`.

Return type str

`dl_common.xml.get_opt_val(obj_pyxb, attr_str, default_val=None)`

Get an optional Simple Content value from a PyXB element.

The attributes for elements that are optional according to the schema and not set in the PyXB object are present and set to None.

PyXB validation will fail if required elements are missing.

Parameters

- **obj_pyxb** – PyXB object
- **attr_str** – str Name of an attribute that the PyXB object may contain.
- **default_val** – any object Value to return if the attribute is not present.

Returns Value of the attribute if present, else `default_val`.

Return type str

`dl_common.xml.get_req_val(obj_pyxb)`

Get a required Simple Content value from a PyXB element.

The attributes for elements that are required according to the schema are always present, and provide a `value()` method.

PyXB validation will fail if required elements are missing.

Getting a Simple Content value from PyXB with `.value()` returns a PyXB object that lazily evaluates to a native Unicode string. This confused parts of the Django ORM that check types before passing values to the database. This function forces immediate conversion to Unicode.

Parameters **obj_pyxb** – PyXB object

Returns Value of the element.

Return type str

exception `dl_common.xml.CompareError`

Bases: `Exception`

Raised when objects are compared and found not to be semantically equivalent.

4.7 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.8 DataONE Client Library for Python

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

The *DataONE Client Library for Python* works together with the *DataONE Common Library for Python* to provide functionality commonly needed by client software that connects to DataONE nodes.

The main functionality provided by this library is a complete set of wrappers for all DataONE API methods. There are many details related to interacting with the DataONE API, such as creating MIME multipart messages, encoding parameters into URLs and handling Unicode. The wrappers hide these details, allowing the developer to communicate with nodes by calling native Python methods which take and return native Python objects.

The wrappers also convert any errors received from the nodes into native exceptions, enabling clients to use Python's concise exception handling system to handle errors.

Contents:

4.8.1 Installing DataONE Client Library for Python

DataONE Common Library for Python is distributed via PyPI, the Python Package Index.

Set up server packages:

- The build environment for DataONE Python extensions and lxml
- Commands used in the install

```
$ sudo apt install --yes build-essential python-dev libssl-dev \
libxml2-dev libxslt-dev openssl
```

Install pip (Python package installer):

```
$ sudo apt install --yes python-pip; sudo pip install pip --upgrade;
```

Install the DataONE Client Library for Python and all its dependencies. This will also automatically build several Python C extensions:

```
$ pip install dataone.libclient
```

4.8.2 Unit Tests

This library is shipped with unit tests that verify correct operation. It is recommended that these are executed after installation.

4.8.3 Updating the library

To update your copy of the library to the latest version available on PyPI, run `pip install` with the `--upgrade` option:

```
$ pip install --upgrade dataone.libclient
```

4.8.4 API

d1_client package

DataONE Client Library.

The `/client/index` works together with the `/common/index` to provide functionality commonly needed by client software that connects to DataONE nodes.

The main functionality provided by this library is a complete set of wrappers for all DataONE API methods. There are many details related to interacting with the DataONE API, such as creating MIME multipart messages, encoding parameters into URLs and handling Unicode. The wrappers hide these details, allowing the developer to communicate with nodes by calling native Python methods which take and return native objects.

The wrappers also convert any errors received from the nodes into native exceptions, enabling clients to use Python's concise exception handling system to handle errors.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_client.aio package

asyncio based utilities.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

d1_client.aio.async_client module

```
class d1_client.aio.async_client.AsyncDataONEClient (base_url='https://cn.dataone.org/cn',
                                                    timeout_sec=None,
                                                    cert_pub_path=None,
                                                    cert_key_path=None,      dis-
                                                    able_server_cert_validation=False,
                                                    max_concurrent_connections=100,
                                                    try_count=3,   verify_tls=False,
                                                    user_agent=None,
                                                    charset=None)
```

Bases: `object`

```
__init__ (base_url='https://cn.dataone.org/cn', timeout_sec=None, cert_pub_path=None,
          cert_key_path=None, disable_server_cert_validation=False,
          max_concurrent_connections=100, try_count=3, verify_tls=False, user_agent=None,
          charset=None)
```

Args:

base_url: timeout_sec: cert_pub_path: cert_key_path: disable_server_cert_validation:
max_concurrent_connections: Limit on concurrent outgoing connections enforced internally by
aiohttp. try_count:

property session

async close()

async get()

Retrieve the SciObj bytes and write them to a file or other stream.

Parameters

- **file_stream** – Open file-like object Stream to which the SciObj bytes will be written.
- **pid** – str
- **vendor_specific** – dict Custom HTTP headers to include in the request

See also:

MNRead.get().

async get_system_metadata (pid, vendor_specific=None)

async list_objects (fromDate=None, toDate=None, formatId=None, identifier=None, replicaS-
tatus=None, nodeId=None, start=0, count=100, vendor_specific=None)

async get_log_records (fromDate=None, toDate=None, event=None, pidFilter=None, idFil-
ter=None, start=0, count=100, vendor_specific=None)

async describe (pid, vendor_specific=None)

Get headers describing an object.

async synchronize (pid, vendor_specific=None)

Send an object synchronization request to the CN.

async list_nodes (vendor_specific=None)

async get_capabilities (*arg_list, **arg_dict)

dump_headers (header_dict)

d1_client.iter package

This package contains iterators that provide a convenient way to retrieve and iterate over Node contents.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Submodules

d1_client.iter.base_multi module

Base for multiprocessed DataONE type iterator.

```

class dl_client.iter.base_multi.MultiprocessedIteratorBase (base_url, page_size,
                                                            max_workers,
                                                            max_result_queue_size,
                                                            max_task_queue_size,
                                                            api_major,
                                                            client_arg_dict,
                                                            page_arg_dict,
                                                            item_proc_arg_dict,
                                                            page_func, iter_func,
                                                            item_proc_func)

Bases: object

property total

dl_client.iter.base_multi.create_client (base_url='https://cn.dataone.org/cn',
                                         api_major=2, client_arg_dict=None)

```

dl_client.iter.logrecord module

Log Record Iterator.

Iterator that provides a convenient way to retrieve log records from a DataONE node and iterate over the results.

Log records are automatically retrieved from the node in batches as required.

The LogRecordIterator takes a CoordinatingNodeClient or MemberNodeClient together with filters to select a set of log records. It returns an iterator object which enables using a Python `for` loop for iterating over the matching log records.

Log records are retrieved from the Node only when required. This avoids storing a large list of records in memory.

The LogRecordIterator repeatedly calls the Node's `getLogRecords()` API method. The CN implementation of this method yields log records for objects for which the caller has access. Log records are not provided for public objects. This is also how `getLogRecords()` is implemented in the *Metacat* Member Node. In *GMN*, the requirements for authentication for this method are configurable. Other MNs are free to choose how or if to implement access control for this method.

To authenticate to the target Node, provide a valid CILogon signed certificate when creating the CoordinatingNodeClient or MemberNodeClient.

See the `CNCore.getLogRecords()` and `MNCore.getLogRecords()` specifications in the [DataONE Architecture Documentation](#) for more information.

Example

```

#!/usr/bin/env python

import dl_client.client
import sys

logging.basicConfig(level=logging.INFO)
target = "https://mn-unm-1.dataone.org/mn"
client = dl_client.client.MemberNodeClient(target=target)
log_record_iterator = LogRecordIterator(client)
for event in log_record_iterator:
    print "Event      = %s" % event.event
    print "Timestamp = %s" % event.dateLogged.isoformat()

```

(continues on next page)

(continued from previous page)

```

print "IP Address = %s" % event.ipAddress
print "Identifier = %s" % event.identifier
print "User agent = %s" % event.userAgent
print "Subject = %s" % event.subject
print '-' * 79

```

```

class d1_client.iter.logrecord.LogRecordIterator (client,
                                                    get_log_records_arg_dict=None,
                                                    start=0, count=100)

```

Bases: object

Log Record Iterator.

```

__init__ (client, get_log_records_arg_dict=None, start=0, count=100)
Log Record Iterator.

```

Parameters

- **client** – d1_client.cnclient.CoordinatingNodeClient or
- **d1_client.mnclient.MemberNodeClient** – A client that has been initialized with the `base_url` and, optionally, other connection parameters for the DataONE node from which log records are to be retrieved.

Log records for an object are typically available only to subjects that have elevated permissions on the object, so an unauthenticated (public) connection may not receive any log records. See the `CoordinatingNodeClient` and `MemberNodeClient` classes for details on how to authenticate.

- **get_log_records_arg_dict** – dict

If this argument is set, it is passed as keyword arguments to `getLogRecords()`.

The iterator calls the `getLogRecords()` API method as necessary to retrieve the log records. The method supports a limited set of filtering capabilities. Currently, *from-Date*, *toDate*, *event*, *pidFilter* and *idFilter*.

To access these filters, use this argument to pass a dict which matching keys and the expected values. E.g.:

```
{ 'fromDate': datetime.datetime(2009, 1, 1) }
```

- **start** – int

If a section of the log records have been retrieved earlier, they can be skipped by setting a start value.

- **count** – int

The number of log records to retrieve in each `getLogRecords()` call.

Depending on network conditions and Node implementation, changing this value from its default may affect performance and resource usage.

d1_client.iter.logrecord_multi module

Multiprocessed LogRecord Iterator.

Fast retrieval of event log records from a DataONE Node.

See additional notes in SysMeta iter docstring.

```
class d1_client.iter.logrecord_multi.LogRecordIteratorMulti (base_url='https://cn.dataone.org/cn',
                                                            page_size=1000,
                                                            max_workers=16,
                                                            max_result_queue_size=100,
                                                            max_task_queue_size=16,
                                                            api_major=2,
                                                            client_arg_dict=None,
                                                            get_log_records_arg_dict=None)
    Bases: d1_client.iter.base_multi.MultiprocessedIteratorBase
```

d1_client.iter.node module

Iterate over the nodes that are registered in a DataONE environment.

For each Node in the environment, returns a PyXB representation of a DataONE Node document.

<https://releases.dataone.org/online/api-documentation-v2.0/apis/Types.html#Types.Node>

```
class d1_client.iter.node.NodeListIterator (cn_client)
    Bases: object
```

d1_client.iter.objectlist module

Implements an iterator that iterates over the entire ObjectList for a DataONE node. Data is retrieved from the target only when required.

The ObjectListIterator takes a CoordinatingNodeClient or MemberNodeClient together with filters to select a set of objects. It returns an iterator object which enables using a Python `for` loop for iterating over the matching objects. Using the ObjectListIterator is appropriate in circumstances where a large percentage of the total number of objects is expected to be returned or when one of the limited number of filters can be used for selecting the desired set of objects.

If more fine grained filtering is required, DataONE's Solr index should be used. It can be accessed using the Solr Client.

Object information is retrieved from the Node only when required. This avoids storing a large object list in memory.

The ObjectListIterator repeatedly calls the Node's `listObjects()` API method. The CN implementation of this method yields only public objects and objects for which the caller has access. This is also how `listObjects()` is implemented in the *Metacat* and *GMN* Member Nodes. However, other MNs are free to chose how or if to implement access control for this method.

To authenticate to the target Node, provide a valid CILogon signed certificate when creating the CoordinatingNodeClient or MemberNodeClient.

Example:

```
#!/usr/bin/env python
from d1_client import dbaseclient
from d1_client.objectlistiterator import ObjectListIterator

# The Base URL for a DataONE Coordinating Node or Member Node.
base_url = 'https://cn.dataone.org/cn'
# Start retrieving objects from this position.
start = 0
# Maximum number of entries to retrieve.
max = 500
# Maximum number of entries to retrieve per call.
```

(continues on next page)

(continued from previous page)

```

pagesize = 100

client = dlbaseclient.DataONEBaseClient(base_url)
ol = ObjectListIterator(client, start=start, pagesize=pagesize, max=max)
counter = start
print "---"
print "total: %d" % len(ol)
print "---"
for o in ol:
    print "-"
    print "  item      : %d" % counter
    print "  pid        : %s" % o.identifier.value()
    print "  modified   : %s" % o.dateSysMetadataModified
    print "  format     : %s" % o.formatId
    print "  size       : %s" % o.size
    print "  checksum   : %s" % o.checksum.value()
    print "  algorithm  : %s" % o.checksum.algorithm
    counter += 1

```

Output:

```

---
total: 5
---
-
  item      : 1
  pid       : knb-lter-lno.9.1
  modified  : 2011-01-13 18:42:32.469000
  format    : eml://ecoinformatics.org/eml-2.0.1
  size      : 6751
  checksum  : 9039F0388DC76B1A13B0F139520A8D90
  algorithm : MD5
-
  item      : 2
  pid       : LB30XX_030MTV2021R00_20080516.50.1
  modified  : 2011-01-12 22:51:00.774000
  format    : eml://ecoinformatics.org/eml-2.0.1
  size      : 14435
  checksum  : B2200FB7FAE18A3517AA9E2EA680EE09
  algorithm : MD5
-
  ...

```

```

class dl_client.iter.objectlist.ObjectListIterator (client, start=0, fromDate=None,
                                                    pagesize=500,           max=-1,
                                                    nodeId=None)

```

Bases: object

Implements an iterator that iterates over the entire ObjectList for a DataONE node.

Data is retrieved from the target only when required.

__init__ (client, start=0, fromDate=None, pagesize=500, max=-1, nodeId=None)
 Initializes the iterator.

TODO: Extend this with date range and other restrictions

Parameters

- **client** (*DataONEBaseClient* or *derivative*) – The client instance for retrieving stuff.
- **start** (*integer*) – The zero based starting index value (0)
- **fromDate** (*DateTime*) –
- **pagesize** (*integer*) – Number of items to retrieve in a single request (page, 500)
- **max** (*integer*) – Maximum number of items to retrieve (all)

d1_client.iter.objectlist_multi module

Multiprocessed ObjectList Iterator.

Fast retrieval of ObjectList from a DataONE Node.

See additional notes in SysMeta iter docstring.

```
class d1_client.iter.objectlist_multi.ObjectListIteratorMulti (base_url='https://cn.dataone.org/cn',
                                                             page_size=1000,
                                                             max_workers=16,
                                                             max_result_queue_size=100,
                                                             max_task_queue_size=16,
                                                             api_major=2,
                                                             client_arg_dict=None,
                                                             list_objects_arg_dict=None)

Bases: d1_client.iter.base_multi.MultiprocessedIteratorBase
```

d1_client.iter.sysmeta_multi module

Multiprocessed System Metadata iterator.

Parallel download of a set of SystemMetadata documents from a CN or MN. The SystemMetadata to download can be selected by the filters that are available in the MNRead.listObjects() and CNRead.listObjects() API calls. For MNs, these include: fromDate, toDate, formatId and identifier. For CNs, these include the ones supported by MNs plus nodeId.

Note: Unhandled exceptions raised in client code while iterating over results from this iterator, or in the iterator itself, will not be shown and may cause the client code to hang. This is a limitation of the multiprocessing module.

If there is an error when retrieving a System Metadata, such as NotAuthorized, an object that is derived from d1_common.types.exceptions.DataONEException is returned instead.

Will create the same number of DataONE clients and HTTP or HTTPS connections as the number of workers. A single connection is reused, first for retrieving a page of results, then all System Metadata objects in the result.

There is a bottleneck somewhere in this iterator, but it's not pickle/unpickle of sysmeta_pyxb.

Notes on MAX_QUEUE_SIZE:

Queues that become too large can cause deadlocks: <https://stackoverflow.com/questions/21641887/python-multiprocessing-process-hangs-on-join-for-large-queue> Each item in the queue is a potentially large SysMeta PyXB object, so we set a low max queue size.

```
class d1_client.iter.sysmeta_multi.SystemMetadataIteratorMulti (base_url='https://cn.dataone.org/cn',  
                                                                page_size=1000,  
                                                                max_workers=16,  
                                                                max_result_queue_size=100,  
                                                                max_task_queue_size=16,  
                                                                api_major=2,  
                                                                client_arg_dict=None,  
                                                                list_objects_arg_dict=None,  
                                                                get_system_metadata_arg_dict=None)  
  
    Bases: d1_client.iter.base_multi.MultiprocessedIteratorBase
```

d1_client.tests package

Submodules

d1_client.tests.shared_context module

Holds only dynamic data.

All tests import this module and use it for sharing information that is used across tests.

d1_client.tests.test_baseclient_1_0 module

d1_client.tests.test_baseclient_1_1 module

d1_client.tests.test_baseclient_2_0 module

d1_client.tests.test_cnclient_1_0 module

d1_client.tests.test_cnclient_1_1 module

d1_client.tests.test_cnclient_2_0 module

d1_client.tests.test_d1client module

d1_client.tests.test_iter_log_record module

d1_client.tests.test_iter_log_record_multi module

d1_client.tests.test_iter_object_list_multi module

d1_client.tests.test_iter_sysmeta_multi module

d1_client.tests.test_mnclient_1_0 module

d1_client.tests.test_mnclient_1_1 module

d1_client.tests.test_mnclient_2_0 module

d1_client.tests.test_session module

d1_client.tests.test_solr_client module

d1_client.tests.test_util module

Submodules

d1_client.baseclient module

class d1_client.baseclient.**DataONEBaseClient** (*base_url='https://cn.dataone.org/cn', *args, **kwargs*)

Bases: *d1_client.session.Session*

Extend Session by adding REST API wrappers for APIs that are available on both Coordinating Nodes and Member Nodes, and that have the same signature on both:

CNCore/MNCore.getLogRecords() CNRead/MNRead.get() CNRead/MNRead.getSystemMetadata() CNRead/MNRead.describe() CNRead/MNRead.listObjects() CNAuthorization/MNAuthorization.isAuthorized() CNCore/MNCore.ping()

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

On error response, raises a DataONEException.

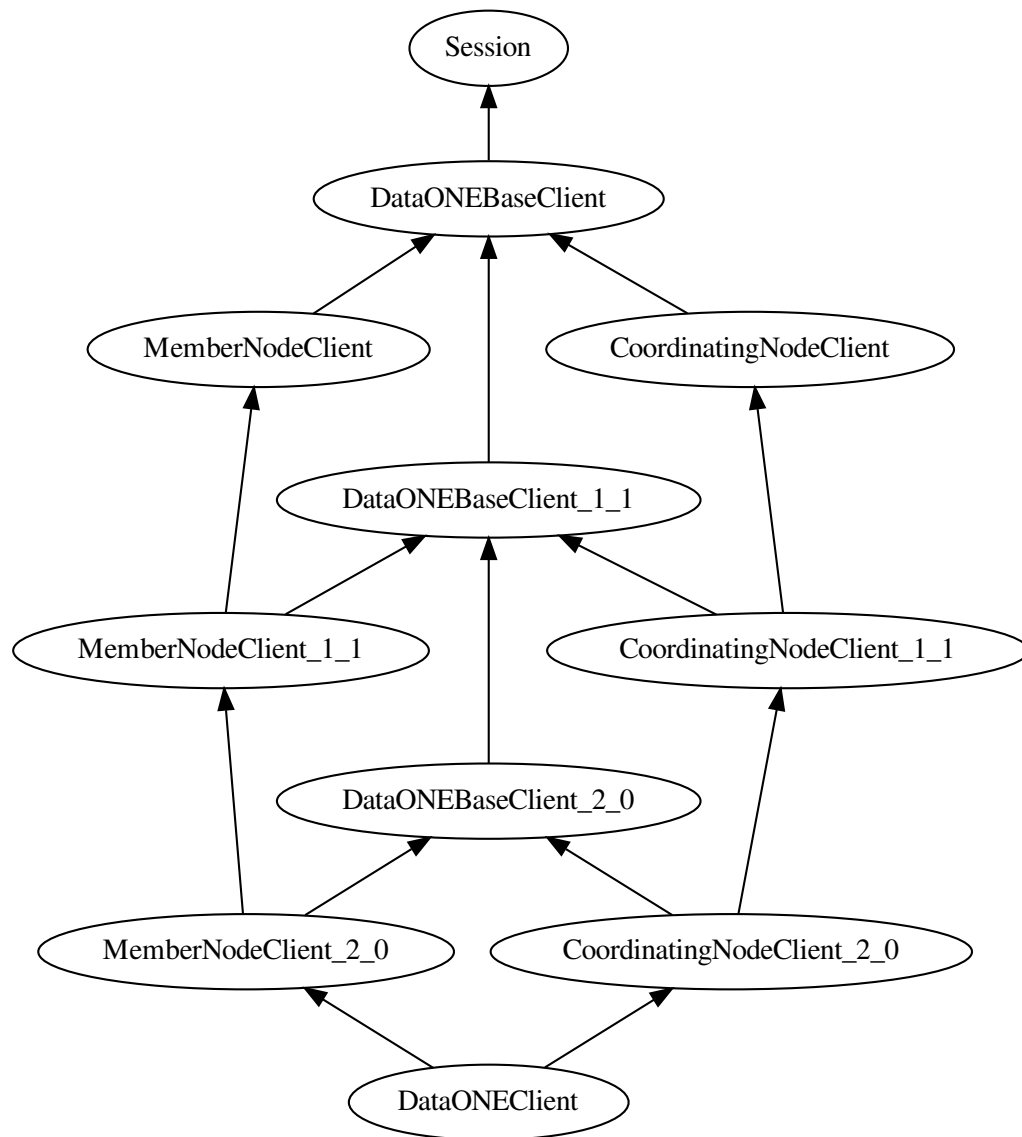
Methods with names that end in “Response” return the HTTPResponse object directly for manual processing by the client. The **Response* methods are only needed in rare cases where the default handling is inadequate, e.g., for dealing with nodes that don’t fully comply with the spec.

The client classes wrap all the DataONE API methods, hiding the many details related to interacting with the DataONE API, such as creating MIME multipart messages, encoding parameters into URLs and handling Unicode.

The clients allow the developer to communicate with nodes by calling native Python methods which take and return native objects.

The clients also convert any errors received from the nodes into native exceptions, enabling clients to use Python’s concise exception handling system to handle errors.

The clients are arranged into the following class hierarchy:



The classes without version designators implement functionality defined in v1.0 of the DataONE service specifications. The classes with version designators implement support for the corresponding DataONE service specifications.

DataONEBaseClient

The DataONEBaseClient classes contain methods that allow access to APIs that are common to Coordinating Nodes and Member Nodes.

- `d1_client.d1baseclient`
- `d1_client.d1baseclient_1_1`
- `d1_client.d1baseclient_2_0`

MemberNodeClient

The MemberNodeClient classes contain methods that allow access to APIs that are specific to Member Nodes.

- `d1_client.mnclient`
- `d1_client.mnclient_1_1`
- `d1_client.mnclient_2_0`

CoordinatingNodeClient

The CoordinatingNodeClient classes contain methods that allow access to APIs that are specific to Coordinating Nodes.

- `d1_client.cnclient`
- `d1_client.cnclient_1_1`
- `d1_client.cnclient_2_0`

DataONEClient

The DataONEClient uses CN- and MN clients to perform high level operations against the DataONE infrastructure.

- `d1_client.d1client`

DataONEObject

Wraps a single DataONE Science Object and adds functionality such as resolve and get.

- `d1_client.d1client`

SolrConnection

Provides functionality for working with DataONE's Solr index, which powers the ONEMercury science data search engine.

- `d1_client.solr_client`

__init__ (*base_url*='https://cn.dataone.org/cn', *args, **kwargs)

Create a DataONEBaseClient. See Session for parameters.

Parameters

- **api_major** (*integer*) – Major version of the DataONE API
- **api_minor** (*integer*) – Minor version of the DataONE API

Returns None

property api_version_tup

property pyxb_binding

getLogRecordsResponse (*fromDate*=None, *toDate*=None, *event*=None, *pidFilter*=None, *idFilter*=None, *start*=0, *count*=100, *vendorSpecific*=None)

getLogRecords (*fromDate*=None, *toDate*=None, *event*=None, *pidFilter*=None, *idFilter*=None, *start*=0, *count*=100, *vendorSpecific*=None)

pingResponse (*vendorSpecific*=None)

ping (*vendorSpecific*=None)

getResponse (*pid*, *stream*=False, *vendorSpecific*=None)

get (*pid*, *stream=False*, *vendorSpecific=None*)

Initiate a `MNRead.get()`. Return a Requests Response object from which the object bytes can be retrieved.

When `stream` is `False`, Requests buffers the entire object in memory before returning the Response. This can exhaust available memory on the local machine when retrieving large science objects. The solution is to set `stream` to `True`, which causes the returned Response object to contain a stream. However, see note below.

When `stream = True`, the Response object will contain a stream which can be processed without buffering the entire science object in memory. However, failure to read all data from the stream can cause connections to be blocked. Due to this, the `stream` parameter is `False` by default.

Also see:

- <http://docs.python-requests.org/en/master/user/advanced/body-content-workflow>
- `get_and_save()` in this module.

get_and_save (*pid*, *sciobj_stream*, *vendorSpecific=None*)

Like `MNRead.get()`, but also retrieve the object bytes and write them to the provided stream. This method does not have the potential issue with excessive memory usage that `get()` with “`stream=False`” has.

Also see `MNRead.get()`.

getSystemMetadataResponse (*pid*, *vendorSpecific=None*)

getSystemMetadata (*pid*, *vendorSpecific=None*)

describeResponse (*pid*, *vendorSpecific=None*)

describe (*pid*, *vendorSpecific=None*)

Note: If the server returns a status code other than 200 OK, a `ServiceFailure` will be raised, as this method is based on a HEAD request, which cannot carry exception information.

listObjectsResponse (*fromDate=None*, *toDate=None*, *formatId=None*, *identifier=None*, *replicaStatus=None*, *nodeId=None*, *start=0*, *count=100*, *vendorSpecific=None*)

listObjects (*fromDate=None*, *toDate=None*, *formatId=None*, *identifier=None*, *replicaStatus=None*, *nodeId=None*, *start=0*, *count=100*, *vendorSpecific=None*)

generateIdentifierResponse (*scheme*, *fragment=None*, *vendorSpecific=None*)

generateIdentifier (*scheme*, *fragment=None*, *vendorSpecific=None*)

archiveResponse (*pid*, *vendorSpecific=None*)

archive (*pid*, *vendorSpecific=None*)

isAuthorizedResponse (*pid*, *action*, *vendorSpecific=None*)

isAuthorized (*pid*, *action*, *vendorSpecific=None*)

Return `True` if user is allowed to perform `action` on `pid`, else `False`.

d1_client.baseclient_1_1 module

class `d1_client.baseclient_1_1.DataONEBaseClient_1_1` (**args*, ***kwargs*)

Bases: `d1_client.baseclient.DataONEBaseClient`

Extend `DataONEBaseClient` with functionality common between Member and Coordinating nodes that was added in v1.1 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__ (*args, **kwargs)

See `d1_client.baseclient.DataONEBaseClient` for args.

queryResponse (queryEngine, query_str, vendorSpecific=None, do_post=False, **kwargs)

`CNRead.query(session, queryEngine, query) → OctetStream` https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.query `MNQuery.query(session, queryEngine, query) → OctetStream` http://jenkins.1.dataone.org/jenkins/job/API%20Documentation%20-%20trunk/ws/api-documentation/build/html/apis/MN_APIs.html#MNQuery.query.

-1.dataone.org/jenkins/job/API%20Documentation%20-%20trunk/ws/api-documentation/build/html/apis/MN_APIs.html#MNQuery.query.

Parameters

- **queryEngine**
- **query_str**
- **vendorSpecific**
- **do_post**
- ****kwargs**

Returns:

query (queryEngine, query_str, vendorSpecific=None, do_post=False, **kwargs)

See Also: `queryResponse()`

Parameters

- **queryEngine**
- **query_str**
- **vendorSpecific**
- **do_post**
- ****kwargs**

Returns:

getQueryEngineDescriptionResponse (queryEngine, vendorSpecific=None, **kwargs)

`CNRead.getQueryEngineDescription(session, queryEngine) → QueryEngineDescription` https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.getQueryEngineDescription `MNQuery.getQueryEngineDescription(session, queryEngine) → QueryEngineDescription` http://jenkins-1.dataone.org/jenkins/job/API%20Documentation%20-%20trunk/ws./api-documentation/build/html/apis/MN_APIs.html#MNQuery.getQueryEngineDescription.

/api-documentation/build/html/apis/MN_APIs.html#MNQuery.getQueryEngineDescription.

Parameters

- **queryEngine**
- ****kwargs**

Returns:

getQueryEngineDescription (queryEngine, **kwargs)

See Also: `getQueryEngineDescriptionResponse()`

Parameters

- **queryEngine**

- ****kwargs**

Returns:

d1_client.baseclient_1_2 module

class `d1_client.baseclient_1_2.DataONEBaseClient_1_2` (*args, **kwargs)

Bases: `d1_client.baseclient_1_1.DataONEBaseClient_1_1`

Extend DataONEBaseClient with functionality common between Member and Coordinating nodes that was added in v1.1 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__ (*args, **kwargs)

See `d1_client.baseclient.DataONEBaseClient` for args.

d1_client.baseclient_2_0 module

class `d1_client.baseclient_2_0.DataONEBaseClient_2_0` (*args, **kwargs)

Bases: `d1_client.baseclient_1_2.DataONEBaseClient_1_2`

Extend DataONEBaseClient_1_2 with functionality common between Member and Coordinating nodes that was added in v2.0 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__ (*args, **kwargs)

See `baseclient.DataONEBaseClient` for args.

updateSystemMetadataResponse (*pid*, *sysmeta_pyxb*, *vendorSpecific=None*)

`MNStorage.updateSystemMetadata(session, pid, sysmeta)` → boolean http://jenkins-1.dataone.org/documentation/unstable/API-Documentation-development/apis/MN_APIs.html#MNStorage.updateSystemMetadata.

Parameters

- **pid**
- **sysmeta_pyxb**
- **vendorSpecific**

Returns:

updateSystemMetadata (*pid*, *sysmeta_pyxb*, *vendorSpecific=None*)

d1_client.cnclient module

class `d1_client.cnclient.CoordinatingNodeClient` (*args, **kwargs)

Bases: `d1_client.baseclient.DataONEBaseClient`

Extend DataONEBaseClient by adding REST API wrappers for APIs that are available on Coordinating Nodes.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__ (*args, **kwargs)

See `d1_client.baseclient.DataONEBaseClient` for args.

listFormatsResponse (vendorSpecific=None)

`CNCore.ping()` → null https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.ping Implemented in `d1_client.baseclient.py`.

`CNCore.create(session, pid, object, sysmeta)` → Identifier https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.create CN INTERNAL

`CNCore.listFormats()` → ObjectFormatList https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.listFormats

Parameters vendorSpecific

Returns:

listFormats (vendorSpecific=None)

See Also: `listFormatsResponse()`

Parameters vendorSpecific

Returns:

getFormatResponse (formatId, vendorSpecific=None)

`CNCore.getFormat(formatId)` → ObjectFormat https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.getFormat.

Parameters

- **formatId**
- **vendorSpecific**

Returns:

getFormat (formatId, vendorSpecific=None)

See Also: `getFormatResponse()`

Parameters

- **formatId**
- **vendorSpecific**

Returns:

reserveIdentifierResponse (pid, vendorSpecific=None)

`CNCore.getLogRecords(session[, fromDate][, toDate][, event][, start][, count])` → Log https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.getLogRecords Implemented in `d1_client.baseclient.py`.

`CNCore.reserveIdentifier(session, pid)` → Identifier https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.reserveIdentifier

Parameters

- **pid**
- **vendorSpecific**

Returns:

reserveIdentifier (*pid, vendorSpecific=None*)

See Also: `reserveIdentifierResponse()`

Parameters

- **pid**
- **vendorSpecific**

Returns:

listChecksumAlgorithmsResponse (*vendorSpecific=None*)

`CNCore.listChecksumAlgorithms()` → `ChecksumAlgorithmList` https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.listChecksumAlgorithms.

Parameters vendorSpecific

Returns:

listChecksumAlgorithms (*vendorSpecific=None*)

See Also: `listChecksumAlgorithmsResponse()`

Parameters vendorSpecific

Returns:

setObsoletedByResponse (*pid, obsoletedByPid, serialVersion, vendorSpecific=None*)

`CNCore.setObsoletedBy(session, pid, obsoletedByPid, serialVersion)` → `boolean` https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.setObsoletedBy.

Parameters

- **pid**
- **obsoletedByPid**
- **serialVersion**
- **vendorSpecific**

Returns:

setObsoletedBy (*pid, obsoletedByPid, serialVersion, vendorSpecific=None*)

See Also: `setObsoletedByResponse()`

Parameters

- **pid**
- **obsoletedByPid**
- **serialVersion**
- **vendorSpecific**

Returns:

listNodesResponse (*vendorSpecific=None*)

`CNCore.listNodes()` → `NodeList` https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.listNodes.

Parameters vendorSpecific

Returns:

listNodes (*vendorSpecific=None*)

See Also: `listNodesResponse()`

Parameters vendorSpecific

Returns:

hasReservationResponse (*pid, subject, vendorSpecific=None*)

CNCore.registerSystemMetadata(session, pid, sysmeta) → Identifier CN INTERNAL.

CNCore.hasReservation(session, pid) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.hasReservation**Parameters**

- **pid**
- **subject**
- **vendorSpecific**

Returns:

hasReservation (*pid, subject, vendorSpecific=None*)

See Also: hasReservationResponse()

Parameters

- **pid**
- **subject**
- **vendorSpecific**

Returns:

resolveResponse (*pid, vendorSpecific=None*)

CNRead.get(session, pid) → OctetStream Implemented in d1_client.baseclient.py.

CNRead.getSystemMetadata(session, pid) → SystemMetadata Implemented in d1_client.baseclient.py

CNRead.resolve(session, pid) → ObjectLocationList https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.resolve**Parameters**

- **pid**
- **vendorSpecific**

Returns:

resolve (*pid, vendorSpecific=None*)

See Also: resolveResponse()

Parameters

- **pid**
- **vendorSpecific**

Returns:

getChecksumResponse (*pid, vendorSpecific=None*)CNRead.getChecksum(session, pid) → Checksum https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.getChecksum.**Parameters**

- **pid**
- **vendorSpecific**

Returns:

getChecksum (*pid*, *vendorSpecific=None*)

See Also: `getChecksumResponse()`

Parameters

- **pid**
- **vendorSpecific**

Returns:

searchResponse (*queryType*, *query*, *vendorSpecific=None*, ***kwargs*)

CNRead.search(session, queryType, query) → ObjectList https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.search.

Parameters

- **queryType**
- **query**
- **vendorSpecific**
- ****kwargs**

Returns:

search (*queryType*, *query=None*, *vendorSpecific=None*, ***kwargs*)

See Also: `searchResponse()`

Parameters

- **queryType**
- **query**
- **vendorSpecific**
- ****kwargs**

Returns:

queryResponse (*queryEngine*, *query=None*, *vendorSpecific=None*, ***kwargs*)

CNRead.query(session, queryEngine, query) → OctetStream https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.query.

Parameters

- **queryEngine**
- **query**
- **vendorSpecific**
- ****kwargs**

Returns:

query (*queryEngine*, *query=None*, *vendorSpecific=None*, ***kwargs*)

See Also: `queryResponse()`

Parameters

- **queryEngine**
- **query**

- **vendorSpecific**
- ****kwargs**

Returns:

getQueryEngineDescriptionResponse (*queryEngine*, *vendorSpecific=None*, ***kwargs*)

CNRead.getQueryEngineDescription(session, queryEngine) → QueryEngineDescription https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.getQueryEngineDescription.

Parameters

- **queryEngine**
- **vendorSpecific**
- ****kwargs**

Returns:

getQueryEngineDescription (*queryEngine*, *vendorSpecific=None*, ***kwargs*)

See Also: getQueryEngineDescriptionResponse()

Parameters

- **queryEngine**
- **vendorSpecific**
- ****kwargs**

Returns:

setRightsHolderResponse (*pid*, *userId*, *serialVersion*, *vendorSpecific=None*)

CNAuthorization.setRightsHolder(session, pid, userId, serialVersion)

→ Identifier https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNAuthorization.setRightsHolder.

Parameters

- **pid**
- **userId**
- **serialVersion**
- **vendorSpecific**

Returns:

setRightsHolder (*pid*, *userId*, *serialVersion*, *vendorSpecific=None*)

See Also: setRightsHolderResponse()

Parameters

- **pid**
- **userId**
- **serialVersion**
- **vendorSpecific**

Returns:

setAccessPolicyResponse (*pid*, *accessPolicy*, *serialVersion*, *vendorSpecific=None*)

CNAuthorization.setAccessPolicy(session, pid, accessPolicy, serialVersion) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNAuthorization.setAccessPolicy.

Parameters

- **pid**
- **accessPolicy**
- **serialVersion**
- **vendorSpecific**

Returns:

setAccessPolicy (*pid, accessPolicy, serialVersion, vendorSpecific=None*)

See Also: setAccessPolicyResponse()

Parameters

- **pid**
- **accessPolicy**
- **serialVersion**
- **vendorSpecific**

Returns:

registerAccountResponse (*person, vendorSpecific=None*)

CNIdentity.registerAccount(session, person) → Subject https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.registerAccount.

Parameters

- **person**
- **vendorSpecific**

Returns:

registerAccount (*person, vendorSpecific=None*)

See Also: registerAccountResponse()

Parameters

- **person**
- **vendorSpecific**

Returns:

updateAccountResponse (*subject, person, vendorSpecific=None*)

CNIdentity.updateAccount(session, person) → Subject https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.updateAccount.

Parameters

- **subject**
- **person**
- **vendorSpecific**

Returns:

updateAccount (*subject, person, vendorSpecific=None*)

See Also: updateAccountResponse()

Parameters

- **subject**
- **person**
- **vendorSpecific**

Returns:

verifyAccountResponse (*subject, vendorSpecific=None*)

CNIdentity.verifyAccount(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.verifyAccount.

Parameters

- **subject**
- **vendorSpecific**

Returns:

verifyAccount (*subject, vendorSpecific=None*)

See Also: verifyAccountResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

getSubjectInfoResponse (*subject, vendorSpecific=None*)

CNIdentity.getSubjectInfo(session, subject) → SubjectList https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.getSubjectInfo.

Parameters

- **subject**
- **vendorSpecific**

Returns:

getSubjectInfo (*subject, vendorSpecific=None*)

See Also: getSubjectInfoResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

listSubjectsResponse (*query, status=None, start=None, count=None, vendorSpecific=None*)

CNIdentity.listSubjects(session, query, status, start, count) → SubjectList https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.listSubjects.

Parameters

- **query**
- **status**
- **start**
- **count**
- **vendorSpecific**

Returns:

listSubjects (*query, status=None, start=None, count=None, vendorSpecific=None*)

See Also: listSubjectsResponse()

Parameters

- **query**
- **status**
- **start**
- **count**
- **vendorSpecific**

Returns:

mapIdentityResponse (*primarySubject, secondarySubject, vendorSpecific=None*)

CNIdentity.mapIdentity(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.mapIdentity.

Parameters

- **primarySubject**
- **secondarySubject**
- **vendorSpecific**

Returns:

mapIdentity (*primarySubject, secondarySubject, vendorSpecific=None*)

See Also: mapIdentityResponse()

Parameters

- **primarySubject**
- **secondarySubject**
- **vendorSpecific**

Returns:

removeMapIdentityResponse (*subject, vendorSpecific=None*)

CNIdentity.removeMapIdentity(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.removeMapIdentity.

Parameters

- **subject**
- **vendorSpecific**

Returns:

removeMapIdentity (*subject, vendorSpecific=None*)

See Also: removeMapIdentityResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

denyMapIdentityResponse (*subject, vendorSpecific=None*)

CNIdentity.denyMapIdentity(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.denyMapIdentity.

Parameters

- **subject**
- **vendorSpecific**

Returns:

denyMapIdentity (*subject, vendorSpecific=None*)

See Also: denyMapIdentityResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

requestMapIdentityResponse (*subject, vendorSpecific=None*)

CNIdentity.requestMapIdentity(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.requestMapIdentity.

Parameters

- **subject**
- **vendorSpecific**

Returns:

requestMapIdentity (*subject, vendorSpecific=None*)

See Also: requestMapIdentityResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

confirmMapIdentityResponse (*subject, vendorSpecific=None*)

CNIdentity.confirmMapIdentity(session, subject) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.confirmMapIdentity.

Parameters

- **subject**
- **vendorSpecific**

Returns:

confirmMapIdentity (*subject, vendorSpecific=None*)

See Also: confirmMapIdentityResponse()

Parameters

- **subject**
- **vendorSpecific**

Returns:

createGroupResponse (*group*, *vendorSpecific=None*)

CNIdentity.createGroup(session, groupName) → Subject https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.createGroup.

Parameters

- **group**
- **vendorSpecific**

Returns:

createGroup (*group*, *vendorSpecific=None*)

See Also: createGroupResponse()

Parameters

- **group**
- **vendorSpecific**

Returns:

updateGroupResponse (*group*, *vendorSpecific=None*)

CNIdentity.addGroupMembers(session, groupName, members) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNIdentity.addGroupMembers.

Parameters

- **group**
- **vendorSpecific**

Returns:

updateGroup (*group*, *vendorSpecific=None*)

See Also: updateGroupResponse()

Parameters

- **group**
- **vendorSpecific**

Returns:

setReplicationStatusResponse (*pid*, *nodeRef*, *status*, *dataoneError=None*, *vendorSpecific=None*)

CNReplication.setReplicationStatus(session, pid, nodeRef, status, failure) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNReplication.setReplicationStatus.

Parameters

- **pid**
- **nodeRef**
- **status**
- **dataoneError**
- **vendorSpecific**

Returns:

setReplicationStatus (*pid*, *nodeRef*, *status*, *dataoneError=None*, *vendorSpecific=None*)

See Also: setReplicationStatusResponse()

Parameters

- **pid**
- **nodeRef**
- **status**
- **dataoneError**
- **vendorSpecific**

Returns:

updateReplicationMetadataResponse (*pid*, *replicaMetadata*, *serialVersion*, *vendorSpecific=None*)

CNReplication.updateReplicationMetadata(session, pid, replicaMetadata, serialVersion)
 → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNReplication.updateReplicationMetadata Not implemented.

Parameters

- **pid**
- **replicaMetadata**
- **serialVersion**
- **vendorSpecific**

Returns:

updateReplicationMetadata (*pid*, *replicaMetadata*, *serialVersion*, *vendorSpecific=None*)

See Also: updateReplicationMetadataResponse()

Parameters

- **pid**
- **replicaMetadata**
- **serialVersion**
- **vendorSpecific**

Returns:

setReplicationPolicyResponse (*pid*, *policy*, *serialVersion*, *vendorSpecific=None*)

CNReplication.setReplicationPolicy(session, pid, policy, serialVersion) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNReplication.setReplicationPolicy.

Parameters

- **pid**
- **policy**
- **serialVersion**
- **vendorSpecific**

Returns:

setReplicationPolicy (*pid*, *policy*, *serialVersion*, *vendorSpecific=None*)

See Also: setReplicationPolicyResponse()

Parameters

- **pid**
- **policy**

- **serialVersion**
- **vendorSpecific**

Returns:

isNodeAuthorizedResponse (*targetNodeSubject*, *pid*, *vendorSpecific=None*)
CNReplication.isNodeAuthorized(session, targetNodeSubject, pid, replicatePer-
mission) → boolean() [https://releases.dataone.org/online/api-
v2.0.1/apis/CN_APIs.html#CNReplication.isNodeAuthorized](https://releases.dataone.org/online/api-v2.0.1/apis/CN_APIs.html#CNReplication.isNodeAuthorized). documentation-

Parameters

- **targetNodeSubject**
- **pid**
- **vendorSpecific**

Returns:

isNodeAuthorized (*targetNodeSubject*, *pid*, *vendorSpecific=None*)
See Also: isNodeAuthorizedResponse()

Parameters

- **targetNodeSubject**
- **pid**
- **vendorSpecific**

Returns:

deleteReplicationMetadataResponse (*pid*, *nodeId*, *serialVersion*, *vendorSpecific=None*)
CNReplication.deleteReplicationMetadata(session, pid, policy, serialVersion)
→ boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNReplication.deleteReplicationMetadata.

Parameters

- **pid**
- **nodeId**
- **serialVersion**
- **vendorSpecific**

Returns:

deleteReplicationMetadata (*pid*, *nodeId*, *serialVersion*, *vendorSpecific=None*)
See Also: deleteReplicationMetadataResponse()

Parameters

- **pid**
- **nodeId**
- **serialVersion**
- **vendorSpecific**

Returns:

updateNodeCapabilitiesResponse (*nodeId, node, vendorSpecific=None*)
 CNRegister.updateNodeCapabilities(session, nodeId, node) → boolean https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRegister.updateNodeCapabilities.

Parameters

- **nodeId**
- **node**
- **vendorSpecific**

Returns:

updateNodeCapabilities (*nodeId, node, vendorSpecific=None*)
 See Also: updateNodeCapabilitiesResponse()

Parameters

- **nodeId**
- **node**
- **vendorSpecific**

Returns:

registerResponse (*node, vendorSpecific=None*)
 CNRegister.register(session, node) → NodeReference https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRegister.register.

Parameters

- **node**
- **vendorSpecific**

Returns:

register (*node, vendorSpecific=None*)
 See Also: registerResponse()

Parameters

- **node**
- **vendorSpecific**

Returns:

d1_client.cnclient_1_1 module

class d1_client.cnclient_1_1.**CoordinatingNodeClient_1_1** (*args, **kwargs)
 Bases: d1_client.baseclient_1_1.DataONEBaseClient_1_1, d1_client.cnclient.CoordinatingNodeClient

Extend DataONEBaseClient_1_1 and CoordinatingNodeClient with functionality for Coordinating nodes that was added in v1.1 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__ (*args, **kwargs)
 See baseclient.DataONEBaseClient for args.

d1_client.cnclient_1_2 module

class d1_client.cnclient_1_2.CoordinatingNodeClient_1_2(*args, **kwargs)

Bases: d1_client.baseclient_1_2.DataONEBaseClient_1_2, d1_client.cnclient.CoordinatingNodeClient

Extend DataONEBaseClient_1_2 and CoordinatingNodeClient with functionality for Coordinating nodes that was added in v1.1 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__(*args, **kwargs)

See baseclient.DataONEBaseClient for args.

d1_client.cnclient_2_0 module

class d1_client.cnclient_2_0.CoordinatingNodeClient_2_0(*args, **kwargs)

Bases: d1_client.baseclient_2_0.DataONEBaseClient_2_0, d1_client.cnclient_1_2.CoordinatingNodeClient_1_2

Extend DataONEBaseClient_2_0 and CoordinatingNodeClient_1_2 with functionality for Coordinating nodes that was added in v2.0 of the DataONE infrastructure.

Updated in v2:

- CNCore.listFormats() → ObjectFormatList
- CNRead.listObjects(session[, fromDate][, toDate][, formatId]
- MNRead.listObjects(session[, fromDate][, toDate][, formatId]

The base implementations of listFormats() and listObjects() handle v2 when called through this class.

https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html

__init__(*args, **kwargs)

See baseclient.DataONEBaseClient for args.

deleteResponse(pid)

CNCore.delete(session, id) → Identifier DELETE /object/{id}

Parameters pid

Returns:

delete(pid)

See Also: deleteResponse()

Parameters pid

Returns:

synchronizeResponse(pid, vendorSpecific=None)

CNRead.synchronize(session, pid) → boolean POST /synchronize.

Args: pid: vendorSpecific:

synchronize(pid, vendorSpecific=None)

See Also: synchronizeResponse() Args: pid: vendorSpecific:

Returns:

```

viewResponse (theme, did, vendorSpecific=None)
view (theme, did)
listViewsResponse (vendorSpecific=None)
listViews ()
echoCredentialsResponse (vendorSpecific=None)
echoCredentials (vendorSpecific=None)
echoSystemMetadataResponse (sysmeta_pyxb, vendorSpecific=None)
echoSystemMetadata (sysmeta_pyxb, vendorSpecific=None)
echoIndexedObjectResponse (queryEngine, sysmeta_pyxb, obj, vendorSpecific=None)
echoIndexedObject (queryEngine, sysmeta_pyxb, obj, vendorSpecific=None)

```

d1_client.command_line module

Utilities for command line tools that instantiate *DataONEClient()*, *CoordinatingNodeClient()*, or *MemberNodeClient()* objects.

The intention is to both reduce the amount of boilerplate code in command line tools that interact with the DataONE infrastructure and to standardize the behavior of the scripts.

```

d1_client.command_line.get_standard_arg_parser (description_str=None,          for-
                                                matter_class=<class          'arg-
                                                parse.ArgumentParserDefaultsHelpFormatter'>,
                                                add_base_url=False)

```

Return an `argparse.ArgumentParser` populated with a standard set of command line arguments.

Command line tools that interact with the DataONE infrastructure typically instantiate a DataONE Client with all arguments either set to their defaults or specified as command line arguments by the user.

This module makes it convenient for scripts to add a standardized set of command line arguments that allow the user to override the default settings in the DataONE Client as needed.

The script that calls this function will typically add its own specific arguments by making additional `parser.add_argument()` calls before extracting the command line arguments with `args = parser.parse_args()`.

When creating the DataONE Client, simply pass the command line arguments to the client via the `command_line_adapter()`.

Parameters

- **description_str** – Description of the command The description is included in the automatically generated help message.
- **formatter_class** – Modify the help message format. See the *argparse* module for available *Formatter* classes.
- **add_base_url** – Require a BaseURL to be provided as a positional command line argument.

If the script will be creating a CN Client, leave this set to `False` to enable automatically connecting to the CN in the environment specified by `--env`, which is the Production environment by default.

If the script will be creating a MN Client, set this to `True` to require a MN BaseURL to be specified on the command line.

Returns

Prepulated with command line arguments that allow overriding DataONEClient defaults.

Return type argparse.ArgumentParser()

Example

def main():

```
    parser = d1_client.command_line.get_standard_d1_client_arg_parser( __doc__,
        add_base_url=True
    ) parser.add_argument(
        "--my-additional-arg", ...
    args = parser.parse_args() ... client = d1_client.cnclient_2_0.CoordinatingNodeClient_2_0(
        d1_client.command_line.args_adapter(args)
```

`d1_client.command_line.args_adapter(args)`

Convert a command line arguments object to a dict suitable for passing to a D1Client create call via argument unpacking.

Parameters args – Object returned from *parser.parse_args()*

Returns Arguments valid for passing to a D1Client create call.

Return type dict

Example

```
args = parser.parse_args() ... client = d1_client.cnclient_2_0.CoordinatingNodeClient_2_0(
    **d1_client.command_line.args_adapter(args)
)
```

`d1_client.command_line.log_setup(is_debug, disable_existing_loggers=False)`

Set up a log format that is suitable for writing to the console by command line tools.

d1_client.d1client module

class `d1_client.d1client.DataONEClient(*args, **kwargs)`

Bases: `d1_client.mnclient_2_0.MemberNodeClient_2_0`, `d1_client.cnclient_2_0.CoordinatingNodeClient_2_0`

Perform high level operations against the DataONE infrastructure.

The other Client classes are specific to CN or MN and to architecture version. This class provides a more abstract interface that can be used for interacting with any DataONE node regardless of type and version.

__init__ (*args, **kwargs)

See baseclient.DataONEBaseClient for args.

create_sciobj (pid, format_id, sciobj, vendor_specific_dict=None, **sysmeta_dict)

Create a Science Object on a Member Node.

Wrapper for MNSStorage.create() that includes semi-automatic generation of System Metadata.

Parameters

- **pid** – str Persistent Identifier.
- **format_id** – str formatId of the Science Object.
- **sciobj** – str, bytes or file-like stream str: Path to file bytes: Bytes file-like stream: lxml.etree of XML doc to validate
- **vendor_specific_dict** – dict Pass additional, vendor specific parameters.
- ****sysmeta_dict** – dict

Parameters to customize the System Metadata.

See also:

`d1_common.system_metadata.generate_system_metadata_pyxb()`

create_sysmeta (*pid, format_id, sciobj_stream, **sysmeta_dict*)

get_node_id ()

`d1_client.d1client.get_api_major_by_base_url` (*base_url='https://cn.dataone.org/cn',*
**client_arg_list, **client_arg_dict*)

Read the Node document from a node and return an int containing the latest D1 API version supported by the node.

The Node document can always be reached through the v1 API and will list services for v1 and any later APIs versions supported by the node.

`d1_client.d1client.get_client_type` (*d1_client_obj*)

`d1_client.d1client.get_version_tag_by_d1_client` (*d1_client_obj*)

`d1_client.d1client.get_client_class_by_version_tag` (*api_major*)

d1_client.mnclient module

class `d1_client.mnclient.MemberNodeClient` (**args, **kwargs*)

Bases: `d1_client.baseclient.DataONEBaseClient`

Extend DataONEBaseClient by adding REST API wrappers for APIs that are available on Member Nodes.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html

__init__ (**args, **kwargs*)

See baseclient.DataONEBaseClient for args.

getCapabilitiesResponse (*vendorSpecific=None*)

getCapabilities (*vendorSpecific=None*)

getChecksumResponse (*pid, checksumAlgorithm=None, vendorSpecific=None*)

getChecksum (*pid, checksumAlgorithm=None, vendorSpecific=None*)

synchronizationFailedResponse (*message, vendorSpecific=None*)

synchronizationFailed (*message, vendorSpecific=None*)

createResponse (*pid, obj, sysmeta_pyxb, vendorSpecific=None*)

create (*pid, obj, sysmeta_pyxb, vendorSpecific=None*)

```
updateResponse (pid, obj, newPid, sysmeta_pyxb, vendorSpecific=None)
update (pid, obj, newPid, sysmeta_pyxb, vendorSpecific=None)
deleteResponse (pid, vendorSpecific=None)
delete (pid, vendorSpecific=None)
systemMetadataChangedResponse (pid, serialVersion, dateSysMetaLastModified, vendorSpecific=None)
systemMetadataChanged (pid, serialVersion, dateSysMetaLastModified, vendorSpecific=None)
replicateResponse (sysmeta_pyxb, sourceNode, vendorSpecific=None)
replicate (sysmeta_pyxb, sourceNode, vendorSpecific=None)
getReplicaResponse (pid, vendorSpecific=None)
getReplica (pid, vendorSpecific=None)
```

d1_client.mnclient_1_1 module

```
class d1_client.mnclient_1_1.MemberNodeClient_1_1 (*args, **kwargs)
    Bases: d1_client.baseclient_1_1.DataONEBaseClient_1_1, d1_client.mnclient.MemberNodeClient

    Extend DataONEBaseClient_1_1 and MemberNodeClient with functionality for Member nodes that was added
    in v1.1 of the DataONE infrastructure.

    For details on how to use these methods, see:
    https://releases.dataone.org/online/api-documentation-v2.0/apis/MN\_APIs.html

    __init__ (*args, **kwargs)
        See baseclient.DataONEBaseClient for args.
```

d1_client.mnclient_1_2 module

```
class d1_client.mnclient_1_2.MemberNodeClient_1_2 (*args, **kwargs)
    Bases: d1_client.baseclient_1_2.DataONEBaseClient_1_2, d1_client.mnclient.MemberNodeClient

    Extend DataONEBaseClient_1_2 and MemberNodeClient with functionality for Member nodes that was added
    in v1.2 of the DataONE infrastructure.

    For details on how to use these methods, see:
    https://releases.dataone.org/online/api-documentation-v2.0/apis/MN\_APIs.html

    __init__ (*args, **kwargs)
        See baseclient.DataONEBaseClient for args.

    viewResponse (theme, did, vendorSpecific=None, **kwargs)
    view (theme, did, **kwargs)
    listViewsResponse (vendorSpecific=None, **kwargs)
    listViews (**kwargs)
    getPackageResponse (did, packageType='application/bagit-097', vendorSpecific=None,
                        **kwargs)
```

`getPackage (did, packageType='application/bagit-097', **kwargs)`

d1_client.mnclient_2_0 module

class `d1_client.mnclient_2_0.MemberNodeClient_2_0 (*args, **kwargs)`
 Bases: `d1_client.baseclient_2_0.DataONEBaseClient_2_0`, `d1_client.mnclient_1_2.MemberNodeClient_1_2`

Extend `DataONEBaseClient_2_0` and `MemberNodeClient_1_2` with functionality for Member nodes that was added in v2.0 of the DataONE infrastructure.

For details on how to use these methods, see:

https://releases.dataone.org/online/api-documentation-v2.0/apis/MN_APIs.html

__init__ (**args, **kwargs*)
 See `baseclient.DataONEBaseClient` for args.

d1_client.session module

class `d1_client.session.Session (base_url='https://cn.dataone.org/cn', cert_pem_path=None, cert_key_path=None, **kwargs_dict)`

Bases: `object`

__init__ (*base_url='https://cn.dataone.org/cn', cert_pem_path=None, cert_key_path=None, **kwargs_dict*)

The Session improves performance by keeping connection related state and allowing it to be reused in multiple API calls to a DataONE Coordinating Node or Member Node. This includes:

- A connection pool
- HTTP persistent connections (HTTP/1.1 and keep-alive)

Based on Python Requests: - <http://docs.python-requests.org/en/master/> - <http://docs.python-requests.org/en/master/user/advanced/#session-objects>

Parameters

- **base_url** – DataONE Node REST service BaseURL.
- **cert_pem_path** – Path to a PEM formatted certificate file. If provided and

accepted by the remote node, the subject for which the certificate was issued is added to the authenticated context in which API calls are made by the client. Equivalent subjects and group subjects may be implicitly included as well. If the certificate is used together with an JWT token, the two sets of subjects are combined. :type `cert_pem_path`: `string`

Parameters

- **cert_key_path** (*string*) – Path to a PEM formatted file that contains the private key for the certificate file. Only required if the certificate file does not itself contain the private key.
- **jwt_token** – Base64 encoded JSON Web Token. If provided and accepted by the

remote node, the subject for which the token was issued is added to the authenticated context in which API calls are made by the client. Equivalent subjects and group subjects may be implicitly included as well. If the token is used together with an X.509 certificate, the two sets of subjects are combined. :type `token`: `string`

Parameters

- **timeout_sec** (*float, int, None*) – Time in seconds that requests will wait for a response. None, 0, 0.0 disables timeouts. Default is DEFAULT_HTTP_TIMEOUT, currently 60 seconds.
- **try_count** (*int*) – Set number of times to try a request before failing. If not set, retries are still performed, using the default number of retries. To disable retries, set to 1.
- **headers** (*dictionary*) – headers that will be included with all connections.
- **query** (*dictionary*) – URL query parameters that will be included with all connections.
- **use_stream** (*bool*) – Use streaming response. When enabled, responses must be completely read to free up the connection for reuse. (default:False)
- **verify_tls** (*bool or path*) – Verify the server side TLS/SSL certificate. (default: True). Can also hold a path that points to a trusted CA bundle
- **suppress_verify_warnings** (*bool*) – Suppress the warnings issued when verify_tls is set to False.
- **user_agent** (*str*) – Override the default User-Agent string used by d1client.
- **charset** (*str*) – Override the default Charset used by d1client. (default: utf-8)
- **mmp_boundary** (*str*) – By default, boundary strings used in Mime Multipart (MMP) documents are automatically generated as required. If provided, this string will be used instead. This is typically required for creating reproducible test results and may be required by non-compliant MMP parsers.

Returns None

property `base_url`

property `auth_subj_tup`

This property contains the DataONE subjects for which connections created by the client may be authenticated on the remote node.

Returns

primary subject string, equivalent identities set

- If a certificate was passed when the client was created:
 - primary subject string: Extracted from the certificate DN
 - equivalent identities set: group memberships and inferred symbolic subjects extracted from the SubjectInfo (if present.)
 - All returned subjects are DataONE compliant serializations.
 - A copy of the primary subject is always included in the set of equivalent identities.
- If a certificate was not passed when the client was created:

Both primary subject string and equivalent identities set contain the The DataONE public symbolic subject

Return type 2-tuple

GET (*rest_path_list, **kwargs*)

Send a GET request. See requests.sessions.request for optional parameters.

Returns Response object

HEAD (*rest_path_list*, ***kwargs*)

Send a HEAD request. See `requests.sessions.request` for optional parameters.

Returns Response object

POST (*rest_path_list*, ***kwargs*)

Send a POST request with optional streaming multipart encoding. See `requests.sessions.request` for optional parameters. To post regular data, pass a string, iterator or generator as the `data` argument. To post a multipart stream, pass a dictionary of multipart elements as the `fields` argument. E.g.:

```
fields = { 'field0': 'value', 'field1': 'value', 'field2': ('filename.xml', open('file.xml', 'rb'), 'application/xml')
}
```

Returns Response object

PUT (*rest_path_list*, ***kwargs*)

Send a PUT request with optional streaming multipart encoding. See `requests.sessions.request` for optional parameters. See `post()` for parameters.

Returns Response object

DELETE (*rest_path_list*, ***kwargs*)

Send a DELETE request. See `requests.sessions.request` for optional parameters.

Returns Response object

OPTIONS (*rest_path_list*, ***kwargs*)

Send a OPTIONS request. See `requests.sessions.request` for optional parameters.

Returns Response object

get_curl_command_line (*method*, *url*, ***kwargs*)

Get request as cURL command line for debugging.

dump_request_and_response (*response*)

Return a string containing a nicely formatted representation of the request and response objects for logging and debugging.

- Note: Does not work if the request or response body is a `MultipartEncoder` object.

d1_client.solr_client module

Basic Solr client.

Based on: <http://svn.apache.org/viewvc/lucene/solr/tags/release-1.2.0/client/python/solr.py>

DataONE provides an index of all objects stored in the Member Nodes that form the DataONE federation. The index is stored in an Apache *Solr* database and can be queried with the `SolrClient`.

The DataONE Solr index provides information only about objects for which the caller has access. When querying the index without authenticating, only records related to public objects can be retrieved. To authenticate, provide a certificate signed by *CILogon* when creating the client.

Example:

```
# Connect to the DataONE Coordinating Nodes in the default (production) environment.
c = d1_client.solr_client.SolrConnection()

search_result = c.search({
    'q': 'id:[* TO *]', # Filter for search
```

(continues on next page)

(continued from previous page)

```
'rows': 10, # Number of results to return
'fl': 'formatId', # List of fields to return for each result
})

pprint.pprint(search_result)
```

```
class d1_client.solr_client.Param(field, param)
```

Bases: object

Solr Query Parameter

```
class d1_client.solr_client.SolrClient(base_url='https://cn.dataone.org/cn', *args,
                                     **kwargs)
```

Bases: `d1_client.baseclient_1_2.DataONEBaseClient_1_2`

Extend DataONEBaseClient_1_2 with functions for querying Solr indexes hosted on CNs and MNs.

Example:

To connect to DataONE's production environment:

```
solr_client = SolrClient()
```

- To connect to a non-production environment, pass the baseURL for the environment. E.g.:

```
solr_client = SolrClient('https://cn-stage.test.dataone.org/cn')
```

For the supported keyword args, see:

```
d1_client.session.Session()
```

- Most methods take a ****query_dict** as a parameter. It allows passing any number of query parameters that will be sent to Solr.

Pass the query parameters as regular keyword arguments. E.g.:

```
solr_client.search(q=Param('id', 'abc*'), fq=Param('id', 'def*'))
```

To pass multiple query parameters of the same type, pass a list. E.g., to pass multiple filter query (fq) parameters:

```
solr_client.search( q=Param('id', 'abc*'), fq=[Param('id', 'def*'), Param('id', 'ghi')]
)
```

- For more information about DataONE's Solr index, see:

<https://releases.dataone.org/online/api-documentation-v2.0/design/SearchMetadata.html>

```
search (**query_dict)
```

Search the Solr index.

Example:

```
result_dict = search(q=['id:abc*'], fq=['id:def*', 'id:ghi'])
```

```
get (doc_id)
```

Retrieve the specified document.

```
get_ids (start=0, rows=1000, **query_dict)
```

Retrieve a list of identifiers for documents matching the query.

count (***query_dict*)

Return the number of entries that match query.

get_field_values (*name, maxvalues=-1, sort=True, **query_dict*)

Retrieve the unique values for a field, along with their usage counts.

Parameters

- **name** (*string*) – Name of field for which to retrieve values
- **sort** – Sort the result
- **maxvalues** (*int*) – Maximum number of values to retrieve. Default is -1, which causes retrieval of all values.

Returns dict of {fieldname: [[value, count], ...], }

get_field_min_max (*name, **query_dict*)

Returns the minimum and maximum values of the specified field. This requires two search calls to the service, each requesting a single value of a single field.

@param name(string) Name of the field @param q(string) Query identifying range of records for min and max values @param fq(string) Filter restricting range of query

@return list of [min, max]

field_alpha_histogram (*name, n_bins=10, include_queries=True, **query_dict*)

Generates a histogram of values from a string field.

Output is: [[low, high, count, query], ...]. Bin edges is determined by equal division of the fields.

delete (*doc_id*)

delete_by_query (*query*)

add (***fields*)

add_docs (*docs*)

docs is a list of fields that are a dictionary of name:value for a record.

commit (*waitFlush=True, waitSearcher=True, optimize=False*)

class `d1_client.solr_client.SolrRecordTransformerBase`

Bases: object

Base for Solr record transformers.

Used to transform a Solr search response document into some other form, such as a dictionary or list of values.

transform (*record*)

class `d1_client.solr_client.SolrArrayTransformer` (*cols=None*)

Bases: `d1_client.solr_client.SolrRecordTransformerBase`

A transformer that returns a list of values for the specified columns.

transform (*record*)

class `d1_client.solr_client.SolrSearchResponseIterator` (*client, page_size=100, max_records=1000, transformer=<d1_client.solr_client.SolrRecordTransformer object>, **query_dict*)

Bases: object

Performs a search against a Solr index and acts as an iterator to retrieve all the values.

process_row(row)

Override this method in derived classes to reformat the row response.

class `d1_client.solr_client.SolrArrayResponseIterator`(client, page_size=100,
cols=None, **query_dict)

Bases: `d1_client.solr_client.SolrSearchResponseIterator`

Returns an iterator that operates on a Solr result set.

The output for each document is a list of values for the columns specified in the cols parameter of the constructor.

class `d1_client.solr_client.SolrSubsampleResponseIterator`(client, q, fq=None,
fields='*',
page_size=100,
n_samples=10000,
trans-
former=<d1_client.solr_client.SolrRecordTrans-
object>)

Bases: `d1_client.solr_client.SolrSearchResponseIterator`

Returns a pseudo-random subsample of the result set.

Works by calculating the number of pages required for the entire data set and taking a random sample of pages until n_samples can be retrieved. So pages are random, but records within a page are not.

class `d1_client.solr_client.SolrValuesResponseIterator`(client, field,
page_size=1000,
**query_dict)

Bases: `object`

Iterates over a Solr get values response.

This returns a list of distinct values for a particular field.

__init__(client, field, page_size=1000, **query_dict)
Initialize.

@param client(SolrConnection) An instance of a solr connection to use. @param field(string) name of the field from which to retrieve values @param q(string) The Solr query to restrict results @param fq(string) A facet query, restricts the set of rows that q is applied to @param fields(string) A comma delimited list of field names to return @param page_size(int) Number of rows to retrieve in each call.

d1_client.util module

`d1_client.util.normalize_request_response_dump`(dump_str)

4.9 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.10 DataONE Science Metadata library for Python

See *DataONE Python Products* for an overview of the DataONE libraries and other products implemented in Python.

The *DataONE Science Metadata library for Python* is a component of the DataONE Investigator Toolkit (ITK). It currently provides schema validation of DataONE Science Metadata XML documents.

Contents:

4.10.1 Synchronizing the schemas with the Coordinating Node

The goal of the *dl_scimeta* Science Metadata validator is to match the validation results of the CN as closely as possible. As such, *dl_scimeta* uses the same set of schemas as that used by the CN, and they are included in the *dl_scimeta* package. If changes are made to the schemas used by the CN, the schemas included in the *dl_scimeta* package should be replaced with a fresh set of schemas from the CN.

The *dl_scimeta* validator is based on the *lxml* library. *lxml* cannot easily be blocked from following HTTP URLs referenced in *schemaLocation* attributes in *xs:include* and *xs:import* elements in XSD schemas used while validating an XML doc. As outgoing network connections and associated delays are not acceptable in many validation scenarios, we use schemas in which URLs have been rewritten to point to existing local XSD files. Or, where there were no existing local XSDs, we include cached versions which were downloaded while preparing the schemas for distribution.

See the *prepare_schema.py* module for details.

4.10.2 Schema synchronization procedure

Synchronize the XSD files used by the *dl_scimeta* Science Metadata validator with the schemas used by the CN.

- Delete the old schema directory and cache

```
$ rm -r ./schema
```

- Download the new schema files, either from GitHub, or from a CN.

- GitHub

```
https://github.com/NCEAS/metacat/tree/master/lib/schema
```

- CN

```
$ rsync -recursive cn:/var/lib/tomcat8/webapps/metacat/schema/ ./schema
```

- Prepare the new schema files for use by *lxml* and create a local cache of schemas not directly included in the CN schema set:

```
$ ./prepare_schema.py
```

- If support for new schemas have been added, add them to the *format_id_to_schema.json* file.
- Run *dl_scimeta* unit tests to ensure that validation still works as expected.

4.10.3 Troubleshooting

As *lxml* is a thin wrapper around the *libxml2* C library, there is no easy way to get insight into the validation process from Python. To help with investigation of validation errors, a command, *resolve_schema.py* is included in *dl_scimeta*. *resolve_schema.py* recursively follows XSD *xs:include* and *xs:import* elements and lists any issues as they are encountered.

4.10.4 API

d1_scimeta package

DataONE Science Metadata Library.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_scimeta.tests package

Submodules

d1_scimeta.tests.test_validate module

d1_scimeta.tests.validate_test_docs module

```
d1_scimeta.tests.validate_test_docs.log = <Logger d1_scimeta.tests.validate_test_docs (DEBUG)>
    Validate a large number of SciObj downloaded from the CN.
```

These should all validate.

```
d1_scimeta.tests.validate_test_docs.main()
d1_scimeta.tests.validate_test_docs.validate_format_id(format_id,
                                                         progress_logger)
d1_scimeta.tests.validate_test_docs.validate_xml(format_id,          test_xml_path,
                                                  progress_logger)
d1_scimeta.tests.validate_test_docs.get_test_xml_path_list(out_dir_path)
```

Submodules

d1_scimeta.gen_root_xsd module

Generate preliminary root XSD docs for each formatId that is supported for validation.

The XSD docs much be further edited by hand to correctly import all namespaces required for validation of each given formatId.

```
d1_scimeta.gen_root_xsd.main()
d1_scimeta.gen_root_xsd.gen_target_ns_to_rel_xsd_path_tup(branch_path,
                                                            root_xsd_path)
d1_scimeta.gen_root_xsd.gen_xsd_tree(ns_xsd_path_tup)
```

d1_scimeta.lxml_validate module

```
d1_scimeta.lxml_validate.main()
```

d1_scimeta.schema_prepare module

d1_scimeta.schema_resolve module

Determine which schema locations will be accessed when validating a given XML doc.

Recursively follows *xs:include* and *xs:import schemaLocation* and issue warnings if XSD docs are missing, invalid or require network access.

Intended for troubleshooting of validation issues.

```

d1_scimeta.schema_resolve.main()

d1_scimeta.schema_resolve.resolve_schemas(xsd_uri_tup,    schema_branch_path,    vis-
                                          ited_xsd_uri_set=None, indent=1)

d1_scimeta.schema_resolve.load_schema(xsd_uri)

d1_scimeta.schema_resolve.download_schema(xsd_url)

exception d1_scimeta.schema_resolve.ResolveError
    Bases: Exception

```

d1_scimeta.util module

```

d1_scimeta.util.get_xsi_schema_location_tup(xml_tree)

```

Extract xsi:schemaLocation from the root of an XML doc.

The root schemaLocation consists of (namespace, uri) pairs stored as a list of strings and designates XSD namespaces and schema locations required for validation.

For schemaLocation in xs:include and xs:import in XSD docs, see other function.

Parameters xml_tree

Returns tup of 2-tups.

Examples

```

xsi:schemaLocation=" http://www.isotc211.org/2005/gmi http://files.axds.co/isobio/gmi/gmi.xsd http://www.
isotc211.org/2005/gmd http://files.axds.co/isobio/gmd/gmd.xsd

```

<>

->

```

( ('http://www.isotc211.org/2005/gmi', 'http://files.axds.co/isobio/gmi/gmi.xsd'), ('http://www.isotc211.org/
2005/gmd', 'http://files.axds.co/isobio/gmd/gmd.xsd')
)

```

```

d1_scimeta.util.get_xs_include_xs_import_schema_location_tup(xsd_tree)

```

Extract xs:schemaLocation from xs:include and xs:import elements in XSD doc.

The schemaLocation consists of a single uri.

For schemaLocation in the root of XML docs, see other function.

```

d1_scimeta.util.get_root_ns(xml_tree)

```

Extract the root namespace for the XML doc.

Returns Extracted from the prefix used for the root element which is also declared as an xmlns in the root element.

Return type str

Examples

```
<xs:schema targetNamespace="http://www.w3.org/XML/1998/namespace" xmlns:xs="http://www.w3.org/2001/XMLSchema">
-> http://www.w3.org/2001/XMLSchema
```

```
d1_scimeta.util.get_target_ns(xml_tree)
Extract the target namespace for the XML doc.
```

Returns

Extracted from the *targetNamespace* attribute of the root element.

If the root element does not have a *targetNamespace* attribute, return an empty string, "".

Return type

str

Examples:

```
<xs:schema targetNamespace="http://www.w3.org/XML/1998/namespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
-> http://www.w3.org/XML/1998/namespace
```

```
d1_scimeta.util.get_abs_root_xsd_path(format_id)
Get abs path to root XSD by formatId.
```

Returns

str

Path to the pre-generated XSD that should import all XSDs required for validating any XML of the given formatId.

E.g.: format_id = <http://www.isotc211.org/2005/gmd> -> /d1_scimeta/ext/isotc211.xsd

Return type

 xsd_path

```
d1_scimeta.util.get_schema_name(format_id)
Get the directory name of a schema by formatId.
```

Returns

str

The name (not path) of the root directory for the XSD files for a given formatId.

This is also the basename of the root XSD file for a given formatId.

E.g.: format_id = <http://www.isotc211.org/2005/gmd> -> isotc211

Return type

 schema_dir_name

```
d1_scimeta.util.get_supported_format_id_list()
Get list of formatIds that are supported by the validator.
```

Returns

list

List of the formatId strings that can be passed to the validate*() functions.

Return type list of format_id

```
d1_scimeta.util.is_installed_scimeta_format_id(format_id)
```

Return True if validation is supported for *format_id*.

```
d1_scimeta.util.gen_abs_xsd_path_list(branch_path)
```

Generate a list of abs paths to XSD files under *branch_path*.

Excludes *.ORIGINAL.* files, which are inferred from their .xsd counterparts.

```
d1_scimeta.util.get_abs_schema_branch_path(format_id)
```

Get absolute path to a branch holding all the XSD files for a single formatId.

The returned path will always have a trailing slash.

Returns

str

E.g.:

```
format_id      =      http://www.isotc211.org/2005/gmd      ->
/schema/lib_scimeta/src/d1_scimeta/schema/isotc211/
```

Return type abs_xsd_path

```
d1_scimeta.util.gen_xsd_name_dict(branch_path, xsd_path_list)
```

Generate a dict of XSD name to abs path to the XSD file.

The key is the part of the XSD path that follows under *branch_path*.

E.g.: path = /schema/isotc211/gmd/applicationSchema.xsd -> key = /gmd/applicationSchema.xsd val = /schema/isotc211/gmd/applicationSchema.xsd

```
d1_scimeta.util.gen_rel_xsd_path(branch_path, xsd_path)
```

Generate the relative part of the XSD path that follows under *branch_path*.**Parameters**

- **branch_path** – str Absolute path to a branch holding all the XSD files for a single formatId.
- **xsd_path** – str Absolute path to an XSD file under the *branch_path*.

Returns

str

E.g.: branc_path = /schema/isotc211/ xsd_path = /schema/isotc211/gmd/applicationSchema.xsd
-> gmd/applicationSchema.xsd

Return type path

```
d1_scimeta.util.get_rel_path(parent_xsd_path, child_xsd_path)
```

Generate a relative path suitable for use as a *schemaLocation* URI.**Parameters**

- **parent_xsd_path** – str Abs path to XSD file that has the *schemaLocation*.

- **child_xsd_path** – str Abs path to XSD file that the *schemaLocation* should be rewritten to.

Returns

Relative path

E.g.: parent = /schema/isotc211/gmd/maintenance.xsd child =
/schema/isotc211/gmd/citation.xsd -> ../gmd/citation.xsd

Return type str

`d1_scimeta.util.gen_abs_uri(abs_url_or_path, rel_path)`

Create an absolute URL or local filesystem path given at least one absolute component.

Parameters

- **abs_url_or_path** – str URL or absolute filesystem path
- **rel_path** – URL or relative filesystem path

Returns Absolute URL or filesystem path.

`d1_scimeta.util.get_xsd_path(xsd_name_dict, uri)`

Get abs path to the XSD that has a key that matches the end of the URI.

Works for file paths, URLs and URIs. E.g.:

<http://www.w3.org/2001/xml.xsd> -> xml.xsd xml.xsd -> schema/_cache/http_www.w3.org_2001__xml.xsd

`d1_scimeta.util.load_xml_file_to_tree(xml_path)`

`d1_scimeta.util.parse_xml_bytes(xml_bytes, xml_path)`

Parse XML bytes to tree.

Passing in the path to the file enables relative imports to work.

`d1_scimeta.util.get_error_log_as_str(lxml_obj)`

Create a basic message with results from the last XMLParser() or lxml.etree.XMLSchema() run.

`lxml.etree.XMLParser()`, `lxml.etree.XMLSchema()` and some exception objects, such as `lxml.etree.XMLSchemaParseError()` have an `error_log` attribute which contains a list of errors and warnings from the most recent run.

Each error element in the list has attributes:

message: the message text domain: the domain ID (see the `lxml.etree.ErrorDomains` class)
type: the message type ID (see the `lxml.etree.ErrorTypes` class) level: the log level ID (see the `lxml.etree.ErrorLevels` class) line: the line at which the message originated (if applicable) column: the character column at which the message originated (if applicable) filename: the name of the file in which the message originated (if applicable)

For convenience, there are also three properties that provide readable names for the ID values:

domain_name type_name level_name

Parameters `lxml_obj` – `lxml.etree.XMLParser()` or `lxml.etree.XMLSchema()`

Returns Selected elements from the `error_log` of the `lxml_obj`.

Return type str

`d1_scimeta.util.load_bytes_from_file(xml_path)`

`d1_scimeta.util.save_tree_to_file(xml_tree, xml_path)`

Write pretty formatted XML tree to file.

```
d1_scimeta.util.save_bytes_to_file(xml_path, xml_bytes)
    Write bytes to file.

d1_scimeta.util.dump_pretty_tree(xml_tree, msg_str='XML Tree', logger=<bound method Logger.debug of <Logger d1_scimeta.util (DEBUG)>>)

d1_scimeta.util.dump(o, msg_str='Object dump')

d1_scimeta.util.pretty_format_tree(xml_tree)

d1_scimeta.util.is_valid_xml_file(xml_path)

d1_scimeta.util.is_valid_xsd_file(xsd_path)

d1_scimeta.util.is_url(s)
    Return True if s is a URL.
```

```
d1_scimeta.util.strip_whitespace(xml_tree)
    Strip whitespace that might interfere with validation from XSD while maintaining overall formatting.
```

E.g., whitespace in a `gco:DateTime` trips up the validation:

```
<gco:DateTime> 2011-03-18T15:39:17Z
</gco:DateTime>
```

This changes it to:

```
<gco:DateTime>2011-03-18T15:39:17Z</gco:DateTime>
```

Parameters `xml_tree`

Returns stripped `xml_tree`

```
d1_scimeta.util.remove_empty_elements(xml_tree)
    Remove empty elements that might interfere with validation from XSD while maintaining overall formatting.
```

Parameters `xml_tree`

Returns stripped `xml_tree`

```
d1_scimeta.util.apply_xslt_transform(xml_tree, xslt_path)
```

```
d1_scimeta.util.create_lxml_obj(xml_tree, lxml_obj_class)
```

Create an object from an lxml class that takes a tree as parameter.

Parameters

- `xml_tree` – etree
 - `lxml_obj_class` – lxml object
- lxml.etree.XMLSchema lxml.etree.XSLT

exception `d1_scimeta.util.SciMetaError`

Bases: `Exception`

d1_scimeta.validate module

Validate Science Metadata.

Usage:

```
import d1_scimeta.validate

try: d1_scimeta.validate.assert_valid(format_id, xml)
```

```
except d1_scimeta.util.SciMetaError as e: log.error(e)
```

```
d1_scimeta.validate.assert_valid(format_id, xml)
```

Validate an Science Metadata XML file.

Parameters

- **format_id** – str DataONE formatId. Must be one of the keys from the *format_id_to_schema.json* document. E.g., <http://www.isotc211.org/2005/gmd>.
- **xml** – str, bytes or tree str: Path to XML file to validate. bytes: UTF-8 encoded bytes of XML doc to validate tree: lxml.etree of XML doc to validate

Raises On validation error – d1_scimeta.util.SciMetaError

Returns None

Return type On successful validation

```
d1_scimeta.validate.validate_tree(format_id, xml_tree)
```

```
d1_scimeta.validate.validate_bytes(format_id, xml_bytes, xml_path=None)
```

```
d1_scimeta.validate.validate_path(format_id, xml_path)
```

```
d1_scimeta.validate.apply_xerces_adaption_schema_transform(root_xsd_path,  
xml_tree)
```

4.11 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.12 DataONE Test Utilities

The *DataONE Test Utilities* package contains various utilities for testing DataONE infrastructure components and clients. These include the *Instance Generator*, used for creating randomized System Metadata documents, and the *Stress Tester*, used for stress testing of Member Node implementations. The *stress_tester* can create many concurrent connections to a Member Node and simultaneously create any number of randomly generated objects while running queries and object retrievals. There are also various *Utilities*.

Contents:

4.12.1 Build and install

The *DataONE Test Utilities for Python* are distributed via PyPI, the Python Package Index.

Set up server packages:

- The build environment for DataONE Python extensions and lxml
- Commands used in the install

```
$ sudo apt install --yes build-essential python-dev libssl-dev \  
libxml2-dev libxslt-dev openssl
```


Install pip (Python package installer):

```
$ sudo apt install --yes python-pip; sudo pip install pip --upgrade;
```

Install the Test Utilities and their dependencies, including [Multi-Mechanize](#). This will also automatically build several Python C extensions:

```
$ sudo pip install dataone.test_utilities
```

4.12.2 DataONE Instance Generator for Python

The Instance Generator is used for generating randomized instances of the DataONE types for testing.

It is part of the DataONE Test Utilities for Python and is installed as part of the test utilities.

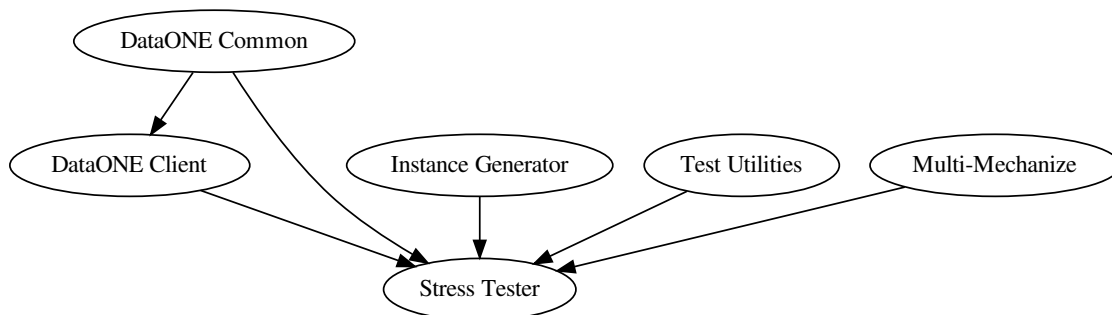
Contents:

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.12.3 DataONE Member Node Stress Tester

The Member Node Stress Tester is a system that generates synthetic queries for Member Nodes. It can perform load testing on DataONE Member Node APIs such as *MNCore.getLogRecords()*, *MNRead.listObjects()* and *MNStorage.create()* and combinations of these.



Contents:

Configuration

Each test has a configuration file that specifies the Multi-Mechanize parameters, such as how long the test should run and how many threads to use. The file is called *config.cfg* and is in the root folder for each test. For instance, the test for *MNStorage.create()* has this file in *projects/create/config.cfg*. See the [Multi-Mechanize home page](#) for information

on how to use this file. In the descriptions on how to run the tests, it is assumed that the settings in *config.cfg* have already been configured.

Shared settings

To avoid duplication of settings that are likely to be the same for each test, the tests each read some of their configuration from the file stored in *./shared/settings.py*. The main setting in this file is the Base URL for the Member Node which is being tested.

Subjects

Certificates

The tests rely on a set of certificates. See [Certificates](#) for details.

Certificates

As many of the stress tests exercise Member Node functionality that is not accessible to unauthenticated clients, a set of test certificates, with which the connections can be established, must be prepared. The certificates must be trusted by the Member Node being tested and each certificate must contain one or more DataONE subjects that are allowed to perform the operations on the MN which a given stress test is exercising.

This section describes how to generate and set up the required certificates.

The generated client side certificates are stored in *./generated/certificates/client_side_certs*. For each connection, a given test selects one or more certificates from the *client_side_certs* folder, depending on which functionality is being tested.

For instance, the test for *MNStorage.create()* will establish all its connections with a certificate called *subject_with_create_permissions*. The test for *MNRead.listObjects()* will select random certificates to stress test the connections with certificates randomly selected from the *certificates/create_update_delete* folder. If there is only one certificate in the folder, that certificate will be used for all the connections created by the test.

CA and Member Node setup

A Member Node that runs in the DataONE production environment must trust the CILogon CAs. But, because only CILogon can sign certificates with that CA, a Member Node is typically set up to trust a locally generated CA for testing. The DataONE Member Node Stress Tester is based on such a setup. This section outlines generating the local CA and then describes procedures and requirements for setting up the certificates that are required by each test.

Setting up the local CA

The first step in setting up certificates for testing is to set up a local CA that will be used for signing the test certificates.

Enter the *./generated/certificates* folder:

```
$ cd ./generated/certificates
```

Create the private key for the local test CA:

```
$ openssl genrsa -des3 -out local_test_ca.key 1024
```

For convenience, remove the password from the key:

```
$ openssl rsa -in local_test_ca.key -out local_test_ca.nopassword.key
```

Create the local test CA. You will be prompted for the information that OpenSSL will use for generating the DN of the certificate. The information you enter is not important, but it is recommended to indicate, in one or more of the fields, that the CA is for testing only. As the DN of the signing CA is included in all signed certificates, it helps with marking those certificates as being for testing only as well.

```
$ openssl req -new -x509 -days 3650 -key local_test_ca.nopassword.key -out local_test_
↪ca.crt
```

Setting up local CA trust

The MN must be set up to trust client side certificates that are signed by the local CA.

The procedure to set up your MN to trust the local CA depends on the software stack on which your MN is based. If it's based on Apache, the procedure is likely to be similar to the following:

- Find the location in which Apache is storing CAs for your MN by reading the configuration file for the MN service, for instance, */etc/apache2/sites-enabled/default-ssl*.
- Note the certificate path set in *SSLCACertificatePath*.
- Move the new local CA, *local_test_ca.crt*, to the certificate path.
- Enter the certificate path and recreate the certificate hashes with:

```
$ c_rehash .
```

Setting up the server side certificate

The MN proves its identity by returning a server side certificate when a client connects.

Enter the *./generated/certificates* folder:

```
$ cd ./generated/certificates
```

Generate the private key:

```
$ openssl genrsa -des3 -out local_test_server_cert.key 1024
```

For convenience, remove the password from the private key:

```
$ openssl rsa -in local_test_server_cert.key -out local_test_server_cert.nopassword.
↪key
```

Create a certificate request. Only the Common Name (CN) field is important for the tester. It must match the name of your server, as seen from the tester. For instance, if the Base URL for your server is *https://my-mn.org/mn*, the Common Name should be *my-mn.org*. An IP address can also be used.

```
$ openssl req -new -key local_test_server_cert.nopassword.key -out local_test_server_
↪cert.csr
```

Sign the CSR with the CA:

```
$ openssl x509 -req -days 36500 -in local_test_server_cert.csr -CA local_test_ca.crt -  
↳CAkey local_test_ca.nopassword.key -set_serial 01 -out local_test_server_cert.crt
```

Setting up the shared key pair

The normal procedure for setting up a new certificate involves creating a private key and a certificate request. The certificate request is then signed with the private key and sent to the signing entity.

Generating a private key is computationally expensive because it requires gathering entropy. When generating a set of certificates for testing, it is convenient to generate the private key up front and reuse it for all the generated certificates.

Enter the *./generated/certificates* folder:

```
$ cd ./generated/certificates
```

Generate the private key:

```
$ openssl genrsa -des3 -out local_test_client_cert.key 1024
```

For convenience, remove the password from the private key:

```
$ openssl rsa -in local_test_client_cert.key -out local_test_client_cert.nopassword.  
↳key
```

The private key implicitly contains the public key. The public key is derived from the private key whenever the private key is passed to a procedure in which the public key is required. For better performance, generate the public key in a separate step:

```
$ openssl rsa -in local_test_client_cert.nopassword.key -pubout -out local_test_  
↳client_cert.public.key
```

Generate list of test subjects

The stress tests randomly pick subjects from a list of subjects. These subjects can be set up automatically with the *generate_subject_list.py* script, or the list can be created manually. The advantage of creating this list manually is that subjects that already known to Member Node can be selected. However, if a completely random list of subjects is sufficient, simply run the script with the desired number of subjects as the only argument. 100 subjects may be a good starting point for the tests.

```
$ ./generate_subject_list.py 100
```

Generate certificates

The final step is to generate the certificates. A script, *generate_certificates.py*, has been provided for this. It uses the subjects file, certificates and keys that were set up in the earlier sections to create the certificates.

```
$ ./generate_certificates.py
```

Before the certificates can be used by the stress tester, the MN must be set up to allow the subjects to create science objects.

Tests

Checking the test setup

Before the tests are run via `multimech-run`, they can be checked by running them directly. When a test script is run directly, it will execute a single instance of the test and any issues are displayed directly as an exception trace. When tests are run via `multimech-run`, exceptions are only counted, not displayed. For example, to execute a single instance of the `MNStorage.create()` test, normally started with `multimech-run projects/create`, run `./projects/create/test_scripts/tier_3_mn_storage_create.py`.

Checking for valid responses

To keep the load on the computer running the stress tests to a minimum, the tests do not attempt to deserialize the documents returned by the APIs being tested. Instead, they perform a simple check for the “200 OK” HTTP status code that the APIs are expected to return together with a valid DataONE data type upon successful completion.

Creating and using test objects

Many of the tests require a set of test objects to be available on the MN being tested and a list of the objects that are available for tests. The `MNStorage.create()` stress test has the secondary purpose of creating the test objects.

The tests use either publicly accessible test objects or access controlled objects. Depending on the required type of object, a test reads a file containing a list of publicly accessible objects or a file containing a list of access controlled objects plus a list of subjects that have access to each object.

Each line in the file used for publicly accessible objects is an object identifier. The default location is `./generated/public_objects.txt` and can be modified in `settings.py`. The file is UTF-8 encoded.

Each line in the file used for access controlled objects contains object identifier, a tab separator and a subject that has at least read access to the object. If the object is readable by more than one subject, the object identifier is repeated on multiple lines, each with a separate subject.

GMN includes a management command to generate these files. Typical usage is:

```
$ ./manage.py generate_object_list --public public_objects.txt
$ ./manage.py generate_object_list private_objects.txt
```

The location of the file can be configured in `settings.py`. The default locations are:

```
./generated/public_objects.txt
./generated/private_objects.txt
```

General test procedure

The tests all follow the same basic pattern:

- The tests work with two lists of objects, one for public and one for private objects. See [Creating and using test objects](#) for more information on how to set up these lists. Public objects can be read by any subject without authentication. Private objects are access controlled. They are accessible only to a specific list of one or more subjects.
- Read the public and private object lists from disk and store them as tables in memory.

- Create a number of threads, as specified in the `threads` section of the `./projects/<test>/config.cfg`. The number of threads equals the number of concurrent connections that will be made to the MN.
- Each thread repeatedly selects random objects and subjects from the object tables and:
- Create a public or authenticated connection to the MN designated in the `BASEURL` setting in `settings.py`. Public connections are made without a certificate and authenticated connections use one of the *generated certificates*.
- Issue the API call to be stress tested to MN.
- Read and discard the entire returned stream.
- Check for a valid status code.

Some tests issue combinations of API calls concurrently.

MNStorage.create()

Stress testing of the `MNStorage.create()` API.

This test concurrently creates many small, randomized science objects on the MN. Each science object has System Metadata with a number of randomly selected parameters.

In addition to stress testing the `MNStorage.create()` API, this test serves a second purpose, which is to populate the MN with test objects with a varied set of permissions for use by other stress tests. By default, the generates objects are small (1024 bytes) to prevent network bandwidth from becoming the limiting factor for performance.

The test generates permissions for randomly selected subjects in the *Creating and using test objects*.

The test always connects with the `subject_with_create_permissions` certificate. This means that the MN must have been set up to allow the DataONE subject, `CN=subject_with_create_permissions, O=d1-stress-tester, C=US, DC=d1-stress-tester, DC=com` to create objects.

To run the test:

```
$ multimech-run projects/create/
```

MNRead.get()

Stress testing of the `MNRead.get()` API with public objects.

This test creates concurrent unauthenticated connections to the MN. For each connection, a random public object is selected and retrieved. The retrieved stream is discarded and the status code is checked.

See the `MNRead.get_auth()` test for testing with authenticated connections.

To run the test:

```
$ multimech-run projects/get
```

MNRead.get_auth()

Stress testing of the `MNRead.get()` API with authenticated connections and access controlled objects.

This test creates concurrent authenticated connections to the MN. For each connection, a random private object and subject with read access to the object is selected. A connection is made to the MN using the subject's certificate and the private object is retrieved. The retrieved stream id discarded and the status code is checked.

To run the test:

```
$ multimech-run projects/get_auth/
```

MNRead.listObjects(), called by Coordinating Node

Stress testing of the `MNRead.listObjects()` API.

This test concurrently retrieves object lists with with random offset and page size, selected from the full range of objects. All connections are made with the `subject_with_cn_permissions` certificate. This means that the MN must be set up to allow the DataONE subject, `CN=subject_with_cn_permissions, O=d1-stress-tester, C=US, DC=d1-stress-tester, DC=com` to act as a Coordinating Node.

To run the test:

```
$ multimech-run projects/list_objects/
```

MNRead.getLogRecords() called by client

Stress testing of the `MNRead.getLogRecords()` API as used for getting the log records for a single private object by a client with regular permissions.

When called by a regular authenticated client, individual access control is applied to each object.

The test selects a random private object. It then creates an authenticated connection using the certificate for one of the subjects which have read access to the object.

To run the test:

```
$ multimech-run projects/get_log_records_client/
```

MNRead.getLogRecords() called by Coordinating Node

Stress testing of the `MNRead.getLogRecords()` API as used by Coordinating Nodes to retrieve a large number of log records created within a given time period.

When a client successfully authenticates as a Coordinating Node, individual access control is not applied to objects.

The test selects a random private object. It then creates a connection and authenticates as a CN. It then retrieves all log records created within a given, random, date range.

To run the test:

```
$ multimech-run projects/get_log_records_client/
```

Combination 1

Stress testing using a combination of the `MNRead.get()`, `MNRead.listObjects()` and `MNStorage.create()` tests described above.

Before running this test, the MN must be populated with test objects, for instance by running the test for `MNStorage.create()`. The objects that are created during this test do not themselves become available for testing until the list of public and private objects is updated as described in *Creating and using test objects*.

The individual stress tests use configuration values from the `config.cfg` file in the combination project directory, not the values in the `config.cfg` files in their own project directories.

Combination 2

Stress testing using a combination of `MNRead.get()` and `MNRead.getLogRecords()`.

Otherwise, like [Combination 1](#).

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

4.12.4 DataONE Test Utilities

See [DataONE Python Products](#) for an overview of the DataONE libraries and other products implemented in Python.

4.12.5 API

d1_test package

DataONE Test Utilities.

The [DataONE Test Utilities](#) package contains various utilities for testing DataONE infrastructure components and clients. These include:

Instance Generator: Used for creating randomized System Metadata documents

Stress Tester: Used for stress testing of Member Node implementations. The `stress_tester` creates a configurable number of concurrent connections to a Member Node and populates the MN with randomly generated objects while running queries and object retrievals.

Utilities: Misc test utilities.

Although this directory is not a package, this `__init__.py` file is required for pytest to be able to reach test directories below this directory.

Subpackages

d1_test.instance_generator package

Subpackages

d1_test.instance_generator.tests package

Instance generator tests.

Submodules

d1_test.instance_generator.tests.test_access_policy module

d1_test.instance_generator.tests.test_checksum module

d1_test.instance_generator.tests.test_datetime module

d1_test.instance_generator.tests.test_format_id module

d1_test.instance_generator.tests.test_identifier module

d1_test.instance_generator.tests.test_person module

d1_test.instance_generator.tests.test_random_data module

d1_test.instance_generator.tests.test_replica module

d1_test.instance_generator.tests.test_replication_policy module

d1_test.instance_generator.tests.test_sciobj module

d1_test.instance_generator.tests.test_subject module

d1_test.instance_generator.tests.test_systemmetadata module

d1_test.instance_generator.tests.test_user_agent module

Submodules

d1_test.instance_generator.access_policy module

d1_test.instance_generator.checksum module

Generate random Checksum.

```
d1_test.instance_generator.checksum.random_checksum_algorithm()
```

```
d1_test.instance_generator.checksum.generate()
```

Generate a Checksum object for a random string, using random algorithm.

d1_test.instance_generator.date_time module

d1_test.instance_generator.format_id module

Generate random formatId.

```
class d1_test.instance_generator.format_id.Generate
    Bases: object
```

d1_test.instance_generator.identifier module

Generate random Identifier.

```
d1_test.instance_generator.identifier.generate_pid (prefix_str='PID_')
d1_test.instance_generator.identifier.generate_sid (prefix_str='SID_',      probabil-
                                                    ity=1.0)
    Generate a SID “probability” * 100 percent of the time.
    Else return None.
d1_test.instance_generator.identifier.generate (prefix_str='DID_',      min_len=5,
                                                    max_len=20)
    Generate instance of Identifier holding a random Unicode string.
d1_test.instance_generator.identifier.generate_bare (prefix_str='DID_', min_len=5,
                                                    max_len=20)
    Generate bare Identifier holding a random Unicode string min and max length does not include the length of the
    prefix.
```

d1_test.instance_generator.media_type module

Generate random MediaType.

```
d1_test.instance_generator.media_type.generate (min_properties=0, max_properties=5)
```

d1_test.instance_generator.names module

Random person first names.

```
d1_test.instance_generator.names.random_names (count=10)
    Returns a random selection of count names.
    No repetitions.
```

d1_test.instance_generator.node_ref module

Generate random NodeReference.

```
d1_test.instance_generator.node_ref.generate (prefix='urn:node:',      min_len=5,
                                                    max_len=20)
    Generate instance of nodeReference holding a urn:node:<random> string.
d1_test.instance_generator.node_ref.generate_bare (prefix=", min_len=5, max_len=20)
    Generate a random Unicode string.
```

d1_test.instance_generator.person module

Generate random Person.

```
d1_test.instance_generator.person.generate ()
```

d1_test.instance_generator.random_data module

Generate random data of various types.

`d1_test.instance_generator.random_data.random_mn(min_len=1, max_len=2)`

`d1_test.instance_generator.random_data.random_cn(min_len=1, max_len=1)`

`d1_test.instance_generator.random_data.random_subj(min_len=1, max_len=2,
fixed_len=None,
group_chance=0.1)`

`d1_test.instance_generator.random_data.random_regular_or_symbolic_subj(min_len=1,
max_len=2,
fixed_len=None,
sym-
bolic_chance=0.1,
group_chance=0.1)`

Return a random regular subject on the form `subj_xx` or a random symbolic subject.

`d1_test.instance_generator.random_data.random_symbolic_subject()`

Return a random symbolic subject.

`public`, `verifiedUser` or `authenticatedUser`.

`d1_test.instance_generator.random_data.random_lower_ascii(min_len=2, max_len=2,
fixed_len=None)`

`d1_test.instance_generator.random_data.random_bytes(num_bytes, max_bytes=None)`

Return a bytes object containing random bytes.

- If only `num_bytes` is set, exactly `num_bytes` are returned.
- If both `num_bytes` and `max_bytes` are set, a random number of bytes between `num_bytes` and `max_bytes` (including) is returned.

`d1_test.instance_generator.random_data.random_bytes_file(num_bytes,
max_bytes=None)`

Return a file-like object containing random bytes.

- If only `num_bytes` is set, exactly `num_bytes` are returned.
- If both `num_bytes` and `max_bytes` is set, a random number of bytes between `num_bytes` and `max_bytes` (including) is returned.

`d1_test.instance_generator.random_data.random_unicode_name()`

Return a random Unicode name.

`d1_test.instance_generator.random_data.random_unicode_name_list(n_names)`

Return a list of random Unicode names.

Names may be repeated

`d1_test.instance_generator.random_data.random_unicode_name_unique_list(n_names)`

Return a list of random Unicode names.

Names are unique

`d1_test.instance_generator.random_data.random_word()`

`d1_test.instance_generator.random_data.random_3_words()`

Return 3 random words separated by a random separator.

`d1_test.instance_generator.random_data.random_word_list(n_words)`

Return a list of random words.

Words may be repeated

`d1_test.instance_generator.random_data.random_word_unique_list(n_names)`

Return a list of random words.

Words are unique

`d1_test.instance_generator.random_data.random_unicode_char()`

Return a random Unicode character (from a limited set)

`d1_test.instance_generator.random_data.random_unicode_char_no_whitespace()`

Return a random Unicode character (from a limited set, no whitespace)

`d1_test.instance_generator.random_data.random_unicode_str(num_chars=5,
max_chars=None)`

Return a str containing random Unicode characters.

- If only `num_chars` is set, exactly `num_chars` characters are returned.
- If both `num_chars` and `max_chars` are set, a random number of characters between `num_chars` and `max_chars` (including) is returned.

`d1_test.instance_generator.random_data.random_email()`

`d1_test.instance_generator.random_data.random_bool()`

`d1_test.instance_generator.random_data.random_bool_factor(f=0.5)`

Return random bool value that is more likely to be True the closer `f` is to 1.0.

- `f == [0, 1)`
- `f = 1.0`: Always return True
- `f = 0.1`: Return True 10% of the time

`d1_test.instance_generator.random_data.random_sized_sample(seq, min_size=1,
max_size=10)`

Return a random number of randomly selected values from `seq`

If it's not possible to meet the `min_size` and/or `max_size` criteria due to the number of values in `seq`, a best effort is made.

`d1_test.instance_generator.random_data.random_sized_sample_pop(seq,
min_size=1,
max_size=10)`

Return a random number of randomly selected values from `seq`, then remove them from `seq`.

If it's not possible to meet the `min_size` and/or `max_size` criteria due to the number of values in `seq`, a best effort is made.

`d1_test.instance_generator.random_data.random_choice_pop(seq)`

`d1_test.instance_generator.random_data.random_within_range(num_bytes,
max_bytes=None)`

Return a random int within range.

- If only `num_bytes` is set, return `num_bytes`
- If both `num_bytes` and `max_bytes` are set, return random int within between `num_bytes` and `max_bytes` (including).

d1_test.instance_generator.replica module

d1_test.instance_generator.replication_policy module

Generate random ReplicationPolicy.

```
d1_test.instance_generator.replication_policy.generate (min_pref=0,    max_pref=4,  
                                                       min_block=0,  
                                                       max_block=4)
```

d1_test.instance_generator.replication_status module

Generate random ReplicationStatus.

```
d1_test.instance_generator.replication_status.generate ()
```

d1_test.instance_generator.sciobj module

d1_test.instance_generator.subject module

Generate random Subject.

```
d1_test.instance_generator.subject.generate ()  
d1_test.instance_generator.subject.generate_bare ()
```

d1_test.instance_generator.system_metadata module

d1_test.instance_generator.unicode_names module

Unicode test names.

Source: <http://www.i18nguy.com/unicode-example.html>.

d1_test.instance_generator.unicode_test_strings module

A set of Unicode strings that are particularly likely to trip up the unwary.

d1_test.instance_generator.user_agent module

Return a randomly selected user agent string, picked from a list of common user agents.

```
class d1_test.instance_generator.user_agent.Generate  
    Bases: object
```

d1_test.instance_generator.words module

Random words.

A selection of 1000 words pulled randomly from /usr/share/dict/words using the randomWords method below.

`d1_test.instance_generator.words.random_words(count=100, supplemental_word_file_path='/usr/share/dict/words')`
Return a random selection of count words from WORDS_1K.
Include words from file if number of words requested is more than available in WORDS_1K.

d1_test.mock_api package

Subpackages

d1_test.mock_api.tests package

Mock API tests.

Submodules

d1_test.mock_api.tests.test_catch_all module

d1_test.mock_api.tests.test_create module

d1_test.mock_api.tests.test_d1_exception module

d1_test.mock_api.tests.test_describe module

d1_test.mock_api.tests.test_generate_identifier module

d1_test.mock_api.tests.test_get module

d1_test.mock_api.tests.test_get_capabilities module

d1_test.mock_api.tests.test_get_format module

d1_test.mock_api.tests.test_get_log_records module

d1_test.mock_api.tests.test_get_query_engine_description module

d1_test.mock_api.tests.test_get_system_metadata module

d1_test.mock_api.tests.test_is_authorized module

d1_test.mock_api.tests.test_list_formats module

d1_test.mock_api.tests.test_list_nodes module

d1_test.mock_api.tests.test_list_objects module

d1_test.mock_api.tests.test_ping module

d1_test.mock_api.tests.test_post module

d1_test.mock_api.tests.test_query module

d1_test.mock_api.tests.test_resolve module

d1_test.mock_api.tests.test_solr_search module

d1_test.mock_api.tests.test_util module

Submodules

d1_test.mock_api.catch_all module

d1_test.mock_api.create module

Mock a generic POST request by echoing the posted body.

A DataONEException can be triggered by adding a custom header. See d1_exception.py

```
d1_test.mock_api.create.add_callback(base_url)
```

```
d1_test.mock_api.create.pack_echo_header(body_bytes, headers, url_obj)
```

```
d1_test.mock_api.create.unpack_echo_header(header_dict)
```

d1_test.mock_api.d1_exception module

Mock DataONEException.

A DataONEException can be triggered in any of the mock APIs by adding a custom header named “trigger” with the status code of the error to trigger, using the vendorSpecific parameter.

E.g.:

```
client.create(..., vendorSpecific={'trigger': '401'})
```

```
d1_test.mock_api.d1_exception.trigger_by_pid(request, pid)
```

```
d1_test.mock_api.d1_exception.trigger_by_header(request)
```

```
d1_test.mock_api.d1_exception.trigger_by_status_code(request, status_code_int)
```

```
d1_test.mock_api.d1_exception.create_regular_d1_exception(status_code_int)
```

```
d1_test.mock_api.d1_exception.create_header_d1_exception(status_code_int)
```

d1_test.mock_api.describe module

d1_test.mock_api.django_client module

Mock Requests to issue requests through the Django test client.

Django includes a test framework with a test client that provides an interface that's similar to that of an HTTP client, but calls Django internals directly. The client enables testing of most functionality of a Django app without actually starting the app as a network service.

For testing GMN's D1 REST interfaces, we want to issue the test requests via the D1 MN client. Without going through the D1 MN client, we would have to reimplement much of what the client does, related to formatting and parsing D1 REST requests.

This module is typically used in tests running under `django.test.TestCase` and requires an active Django context, such as the one provided by `./manage.py test`.

Usage:

```
import d1_test.mock_api.django_client as mock_django_client
```

```
@responses.activate def test_1000(self):
```

```
    mock_django_client.add_callback(MOCK_MN_BASE_URL)                d1_client                =
    d1_client.mnclient_2_0.MemberNodeClient_2_0(MOCK_MN_BASE_URL)    node_pyxb                =
    d1_client.getCapabilities()
```

```
d1_test.mock_api.django_client.add_callback(base_url)
```

d1_test.mock_api.echo_credentials module

d1_test.mock_api.generate_identifier module

Mock:

`CNCore.generateIdentifier(session, scheme[, fragment])` → Identifier https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.generateIdentifier `MNStorage.generateIdentifier(session, scheme[, fragment])` → Identifier https://releases.dataone.org/online/api-documentation-v2.0.1/apis/MN_APIs.html#MNStorage.generateIdentifier

A `DataONEException` can be triggered by adding a custom header. See `d1_exception.py`

```
d1_test.mock_api.generate_identifier.add_callback(base_url)
```

d1_test.mock_api.get module

d1_test.mock_api.get_capabilities module

d1_test.mock_api.get_format module

d1_test.mock_api.get_log_records module

d1_test.mock_api.get_system_metadata module

d1_test.mock_api.is_authorized module

d1_test.mock_api.list_formats module

d1_test.mock_api.list_nodes module

d1_test.mock_api.list_objects module**d1_test.mock_api.ping module**

Mock a ping response.

CNCore.ping() → null https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNCore.ping
 MNRead.ping() → null https://releases.dataone.org/online/api-documentation-v2.0.1/apis/MN_APIs.html#MNCore.ping

A DataONEException can be triggered by adding a custom header. See d1_exception.py

`d1_test.mock_api.ping.add_callback(base_url)`

d1_test.mock_api.post module

Mock a generic POST request by echoing the posted body.

A DataONEException can be triggered by adding a custom header. See d1_exception.py

`d1_test.mock_api.post.add_callback(base_url)`

d1_test.mock_api.query_engine_description module**d1_test.mock_api.resolve module****d1_test.mock_api.solr_query module****d1_test.mock_api.solr_search module****d1_test.mock_api.util module****d1_test.replication_tester package****Submodules****d1_test.replication_tester.replication_error module**

exception `d1_test.replication_tester.replication_error.ReplicationTesterError` (*value*)
 Bases: Exception

d1_test.replication_tester.replication_server module

class `d1_test.replication_tester.replication_server.TestHTTPServer` (*options*,
 pid_unknown,
 pid_not_authorized,
 pid_known_and_authorized,
 src_existing_pid_approve,
 src_existing_pid_deny,
 queue)
 Bases: `threading.Thread`

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

stop()

```
class d1_test.replication_tester.replication_server.Handler(request,  
                                                         client_address,  
                                                         server)
```

Bases: http.server.SimpleHTTPRequestHandler

do_GET()

Serve a GET request.

do_PUT()

d1_test.replication_tester.replication_tester module

Test replication request handling in source and destination MNs.

RepTest is documented in the Utilities section of the dataone.test_utilities package on PyPI.

```
d1_test.replication_tester.replication_tester.main()
```

```
d1_test.replication_tester.replication_tester.create_test_object_on_mn(base_url,  
                                                                      pid)
```

```
class d1_test.replication_tester.replication_tester.ReplicationTester(options,  
                                                                      pid_unknown,  
                                                                      pid_not_authorized,  
                                                                      pid_known_and_authorized,  
                                                                      src_existing_pid_approve,  
                                                                      src_existing_pid_deny,  
                                                                      dst_existing_pid)
```

Bases: object

test_src_mn()

test_dst_mn()

d1_test.replication_tester.run_replication_tester module

d1_test.replication_tester.test_object_generator module

```
d1_test.replication_tester.test_object_generator.generate_random_ascii(prefix)
```

```
d1_test.replication_tester.test_object_generator.generate_science_object_with_sysmeta(pid,  
                                                                                       in-  
                                                                                       clude_
```

d1_test.tests package

Submodules

`d1_test.tests.test_d1_test_case` module

`d1_test.tests.test_sample` module

`d1_test.tests.test_slender_node_test_client` module

`d1_test.tests.test_xml_normalize` module

`d1_test.utilities` package

Submodules

`d1_test.utilities.create_dataone_test_certificate` module

`d1_test.utilities.create_from_file` module

Create an object on a Member Node based on a local file.

```
d1_test.utilities.create_from_file.main()
```

```
d1_test.utilities.create_from_file.gen_sysmeta(pid, f, size, format_id, include_revision_bool, use_v1_bool)
```

Args:

pid: f: size: format_id: include_revision_bool: use_v1_bool:

`d1_test.utilities.generate_sysmeta_for_sciobj` module

`d1_test.utilities.generate_test_subject_certs` module

Create set of test certificates signed by the DataONE Test CA.

```
d1_test.utilities.generate_test_subject_certs.create_key_pair(key_type, n_bits)
```

Create a public/private key pair.

Parameters

- **key_type** (*crypto.TYPE_RSA* or *crypto.TYPE_DSA*) – Key type (RSA or DSA).
- **n_bits** (*int*) – Number of bits to use in the key.

Returns Public/private key pair.

Return type PKey

```
d1_test.utilities.generate_test_subject_certs.create_cert_request(pkey, digest='md5', **name)
```

Create a certificate request.

Parameters

- **pkey** (*PKey*) – Key to associate with the request.
- **digest** (*str*) – Message-Digest algorithm to use for signing.

- ****name** – Name of the subject of the request.

Returns Certificate request.

Return type X509Req

Possible keyword arguments (name):** C - Country name ST - State or province name L - Locality name O - Organization name OU - Organizational unit name CN - Common name emailAddress - E-mail address

```
d1_test.utilities.generate_test_subject_certs.create_session_extension(subject,
                                                                    per-
                                                                    sons,
                                                                    groups)
```

Create the custom X.509 extension object in which DataONE passes session information.

Parameters **subjects** (*list*) – Subjects to store in session.

Returns X.509 v3 certificate extension.

Return type X509Extension

```
d1_test.utilities.generate_test_subject_certs.create_certificate(req,
                                                                xxx_todo_changeme,
                                                                serial,
                                                                xxx_todo_changeme1,
                                                                di-
                                                                gest='md5')
```

Generate a certificate given a certificate request.

Parameters

- **req** (*X509Req*) – Certificate request.
- **issuer_cert** – Certificate of the issuer.
- **issuer_key** – Private key of the issuer.
- **serial** (*str*) – Serial number for certificate.
- **not_before** (*int*) – Timestamp (relative to now) for when the certificate starts being valid.
- **not_after** (*int*) – Timestamp (relative to now) for when the certificate stops being valid.
- **digest** (*str*) – Digest method to use for signing.

Returns The signed certificate.

Return type X509

```
d1_test.utilities.generate_test_subject_certs.main()
```

d1_test.utilities.list_effective_subjects module

d1_test.utilities.my_subject module

Given a certificate, show the subject in DataONE format and optionally display included subject information such as mapped identities and group memberships.

```
d1_test.utilities.my_subject.getSubjectFromName (xName)
```

Given a DN, returns a DataONE subject TODO: This assumes that RDNs are in reverse order. . .

@param

```
d1_test.utilities.my_subject.dumpExtensions (x509)
```

```
d1_test.utilities.my_subject.getSubjectInfoFromCert (x509)
```

Retrieve the SubjectInfo xml from the certificate, if present.

```
d1_test.utilities.my_subject.getSubjectFromCertFile (certFileName)
```

d1_test.utilities.pem_in_http_header module

Convert PEM formatted certificates to and from HTTP header compatible values.

For debugging certificate processing logic, it is sometimes convenient to pass the certificates via HTTP headers instead of HTTPS.

```
d1_test.utilities.pem_in_http_header.pem_in_string_to_pem_in_http_header (pem_str)
```

```
d1_test.utilities.pem_in_http_header.pem_in_http_header_to_pem_in_string (header_str)
```

d1_test.utilities.populate_mn module

Populate a Member Node with randomly generated objects.

```
d1_test.utilities.populate_mn.main ()
```

d1_test.utilities.test_object_generator module

```
d1_test.utilities.test_object_generator.generate_random_ascii (prefix,  
                                                                num_chars=10)
```

```
d1_test.utilities.test_object_generator.generate_science_object_with_sysmeta (pid,  
                                                                                num_min_bytes,  
                                                                                num_max_bytes,  
                                                                                format_id='application',  
                                                                                stream,  
                                                                                in-  
                                                                                clude_revision_bool,  
                                                                                use_v1_bool=False)
```

d1_test.utilities.urlencode module

URL encode / decode provided string.

Examples

```
$ python urlencode.py "http://example.com/data/mydata?row=24" http://example.com/data/mydata?row%3D24
```

```
$ python urlencode.py -d "http://example.com/data/mydata?row%3D24" http://example.com/data/mydata?row=24
```

```
$ python urlencode.py -p "http://example.com/data/mydata?row=24" http:%2F%2Fexample.com%2Fdata%2Fmydata%3Frow=24
```

```
$ python urlencode.py -d -p "http:%2F%2Fexample.com%2Fdata%2Fmydata%3Frow=24" http://example.com/data/mydata?row=24
```

```
$ python urlencode.py -p "http://example.com/data/mydata?row=24" | python urlencode.py -d -p -s http://example.com/data/mydata?row=24
```

```
$ python urlencode.py "" %E0%B8%89%E0%B8%B1%E0%B8%99%E0%B8%81%E0%B8%B4%E0%B8%99%E0%B8%81%E0%B8%81%E0%B9%84%E0%B8%94%E0%B9%89
```

```
d1_test.utilities.urlencode.process_input (input, decode=False, path=False)
```

Submodules

d1_test.d1_test_case module

d1_test.pycharm module

d1_test.sample module

d1_test.slender_node_test_client module

```
class d1_test.slender_node_test_client.SlenderNodeTestClient (sciobj_store_path='./sciobj_store',  
                                                             keep_existing=False,  
                                                             *_args,  
                                                             **_kwargs)
```

Bases: object

A simple drop-in replacement for a MN client, for use when developing and testing SlenderNode scripts.

- MN is simulated to the bare minimum required by SN scripts
- Objects are stored in local files instead of on an MN
- SID to PID dict is held in memory and dumped to file
- Most args are simply ignored

```
__init__ (sciobj_store_path='./sciobj_store', keep_existing=False, *_args, **_kwargs)  
    Create the test client.
```

- Store the sciobj and sysmeta in sciobj_store_path
- sciobj_store_path is created if it does not exist
- If delete_existing is True, delete any existing files in sciobj_store_path

```
create (pid, sciobj_file, sysmeta_pyxb, *_args, **kwargs)
```

```
update (old_pid, sciobj_file, new_pid, new_sysmeta_pyxb, *_args, **kwargs)
```

```
get (did, *_args, **kwargs)  
    Return a file-like object with the sciobj bytes.
```

```
getSystemMetadata (did, *_args, **kwargs)  
    Return sysmeta_pyxb.
```

d1_test.test_files module

Utilities for loading test files.

```
d1_test.test_files.get_abs_test_file_path(rel_path)
d1_test.test_files.load_bin(rel_path)
d1_test.test_files.load_utf8_to_str(rel_path)
    Load file, decode from UTF-8 and return as str.
d1_test.test_files.load_xml_to_pyxb(filename)
d1_test.test_files.load_xml_to_str(filename)
d1_test.test_files.load_xml_to_bytes(filename)
d1_test.test_files.load_json(filename)
d1_test.test_files.load_cert(filename)
d1_test.test_files.load_jwt(filename)
d1_test.test_files.save(obj_str, rel_path, encoding='utf-8')
```

d1_test.xml_normalize module

Generate a str that contains a normalized representation of an XML document.

For unit testing, we want to be able to store and compare samples representing XML documents that are guaranteed to be stable.

Often, XML docs have various sections containing unordered sets of elements where there are no semantics associated with the order in which they appear in the doc. The same is true for element attributes. For DataONE, typical examples are lists of subjects, permission rules and services.

Since the source for such elements are often dict and set based containers that themselves don't provide deterministic ordering, serializing a group of such objects can generate a large number of possible XML docs that, while semantically identical, cannot be directly compared as text or in the DOM.

Normalizing the formatting can be done with a single deserialize to DOM and back to XML, but that will not normalize the ordering of the elements. Without a schema, automated tools cannot rearrange elements in an XML doc, since it is not known if the order is significant. However, for generating and comparing XML doc samples, a stable document that contains all the information from the XML doc is sufficient.

The strategy for generating a stable representation of an XML doc is as follows:

- All sibling XML elements must be sorted regardless of where they are in the tree.
- Each element is the root of a branch of the node tree. Sorting, of course, is based on comparing individual elements in order to determine their relative orderings. If the information in the elements themselves is identical, it is necessary to break the tie by recursively comparing their descendants until either a difference is found, or the two elements are determined to be the roots of two identical branches.
- To enable the sort algorithm to compare the branches, sort keys that hold all information in the branch are generated and passed to the sort. For comparisons to properly compare elements in the most to least significant order, each node in the branch must be in a single list item. So the key is a nested list of lists.
- Finally, since the sort keys are generated from the descendants, siblings in a given element can only be sorted after all their descendants in the tree have been sorted. So the tree must be traversed depth first, and the sort performed as the algorithm is stepping up from a completed level.
- To avoid having to build a new tree depth first, inline sort is used.

Notes

RDF-XML

Although the hierarchical structure of elements is almost always significant in XML, there are instances where semantically identical XML docs can have different hierarchies. This often occurs when generating RDF-XML docs from RDF.

This module only normalizes the ordering of sibling elements and attributes. Parent-child relationships are never changed. So RDF-XML docs generated in such a way that parent-child relationships may differ without change in semantics are not supported.

Background

RDF is an unordered set of subject-predicate-object triples. Triples cannot share values, so when there are multiple triples for a subject, each triple must contain a copy of the subject.

RDF-XML supports expressing triples with less redundancy by factoring shared values out to parent elements. E.g., a set of triples for a subject can be expressed as a series of predicate-object children with a single subject parent.

When generating RDF-XML from RDF that contains many triples that share values, the same set of triples can be represented by many different hierarchies. The hierarchy that is actually generated depends on the algorithm and may also depend on the order in which the triples are processed. If the triples are retrieved from an unordered set, the processing order is pseudo-random, causing pseudo-random variations in the generated hierarchy.

```
d1_test.xml_normalize.get_normalized_xml_representation(xml)
```

Return a str that contains a normalized representation of an XML document.

```
d1_test.xml_normalize.xml_to_stabletree(xml)
```

Return a StableTree that contains a normalized representation of an XML document.

```
d1_test.xml_normalize.etree_to_stable_tree(et_node)
```

Convert an ElementTree to a StableTree.

- Node attributes become @key:string - Text elements become @text:string - name is the name of the xml element

```
class d1_test.xml_normalize.StableNode(name, child_node=None)
```

Bases: object

Tree structure that uses lists instead of dicts, as lists have deterministic ordering.

```
__init__(name, child_node=None)
```

child is E or str.

```
add_child(e)
```

```
get_str(s, indent)
```

```
get_sort_key_()
```

```
sort(p=None)
```

```
d1_test.xml_normalize.StableTree
```

alias of `d1_test.xml_normalize.StableNode`

4.12.6 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

PYTHON MODULE INDEX

d

- `dl_cli`, 76
- `dl_cli.dataone`, 85
- `dl_cli.impl`, 76
- `dl_cli.impl.access_control`, 76
- `dl_cli.impl.check_dependencies`, 76
- `dl_cli.impl.client`, 76
- `dl_cli.impl.command_parser`, 77
- `dl_cli.impl.command_processor`, 80
- `dl_cli.impl.const`, 81
- `dl_cli.impl.exceptions`, 81
- `dl_cli.impl.format_ids`, 81
- `dl_cli.impl.nodes`, 81
- `dl_cli.impl.operation_executer`, 81
- `dl_cli.impl.operation_formatter`, 81
- `dl_cli.impl.operation_maker`, 82
- `dl_cli.impl.operation_queue`, 82
- `dl_cli.impl.operation_validator`, 82
- `dl_cli.impl.replication_policy`, 83
- `dl_cli.impl.session`, 83
- `dl_cli.impl.system_metadata`, 84
- `dl_cli.impl.util`, 84
- `dl_cli.tests`, 84
- `dl_cli.version`, 85
- `dl_client`, 247
- `dl_client.aio`, 247
- `dl_client.aio.async_client`, 247
- `dl_client.baseclient`, 255
- `dl_client.baseclient_1_1`, 258
- `dl_client.baseclient_1_2`, 260
- `dl_client.baseclient_2_0`, 260
- `dl_client.cnclient`, 260
- `dl_client.cnclient_1_1`, 273
- `dl_client.cnclient_1_2`, 274
- `dl_client.cnclient_2_0`, 274
- `dl_client.command_line`, 275
- `dl_client.dlclient`, 276
- `dl_client.iter`, 248
- `dl_client.iter.base_multi`, 248
- `dl_client.iter.logrecord`, 249
- `dl_client.iter.logrecord_multi`, 250
- `dl_client.iter.node`, 251
- `dl_client.iter.objectlist`, 251
- `dl_client.iter.objectlist_multi`, 253
- `dl_client.iter.sysmeta_multi`, 253
- `dl_client.mnclient`, 277
- `dl_client.mnclient_1_1`, 278
- `dl_client.mnclient_1_2`, 278
- `dl_client.mnclient_2_0`, 279
- `dl_client.session`, 279
- `dl_client.solr_client`, 281
- `dl_client.tests`, 254
- `dl_client.tests.shared_context`, 254
- `dl_client.util`, 284
- `dl_common`, 163
- `dl_common.bagit`, 205
- `dl_common.cert`, 163
- `dl_common.cert.jwt`, 164
- `dl_common.cert.subject_info`, 167
- `dl_common.cert.subjects`, 172
- `dl_common.cert.x509`, 173
- `dl_common.checksum`, 205
- `dl_common.const`, 208
- `dl_common.date_time`, 208
- `dl_common.env`, 214
- `dl_common.ext`, 178
- `dl_common.ext.mimeparser`, 178
- `dl_common.iter`, 179
- `dl_common.iter.bytes`, 179
- `dl_common.iter.path`, 180
- `dl_common.iter.stream`, 183
- `dl_common.iter.string`, 183
- `dl_common.logging_context`, 214
- `dl_common.multipart`, 214
- `dl_common.node`, 215
- `dl_common.object_format_cache`, 216
- `dl_common.replication_policy`, 217
- `dl_common.resource_map`, 220
- `dl_common.revision`, 226
- `dl_common.system_metadata`, 227
- `dl_common.tests`, 183
- `dl_common.type_conversions`, 231
- `dl_common.types`, 184
- `dl_common.types.dataoneErrors`, 185

[dl_common.types.dataoneTypes, 185](#)
[dl_common.types.dataoneTypes_v1, 185](#)
[dl_common.types.dataoneTypes_v1_1, 185](#)
[dl_common.types.dataoneTypes_v1_2, 185](#)
[dl_common.types.dataoneTypes_v2_0, 186](#)
[dl_common.types.exceptions, 186](#)
[dl_common.url, 237](#)
[dl_common.util, 238](#)
[dl_common.utils, 190](#)
[dl_common.utils.filesystem, 190](#)
[dl_common.utils.progress_logger, 192](#)
[dl_common.wrap, 193](#)
[dl_common.wrap.access_policy, 194](#)
[dl_common.wrap.simple_xml, 200](#)
[dl_common.xml, 241](#)
[dl_gmn, 140](#)
[dl_gmn.app, 140](#)
[dl_gmn.app.context_processors, 149](#)
[dl_gmn.app.did, 149](#)
[dl_gmn.app.gmn, 150](#)
[dl_gmn.app.management, 140](#)
[dl_gmn.app.management.commands, 140](#)
[dl_gmn.app.management.commands.event, 141](#)
[dl_gmn.app.management.commands.logininfo_adapter, 141](#)
[dl_gmn.app.management.commands.objectlist_adapter, 142](#)
[dl_gmn.app.management.commands.util, 140](#)
[dl_gmn.app.management.commands.util.standard_mapper, 140](#)
[dl_gmn.app.middleware, 142](#)
[dl_gmn.app.middleware.detail_codes, 142](#)
[dl_gmn.app.middleware.exception_handler, 142](#)
[dl_gmn.app.middleware.profilng_handler, 143](#)
[dl_gmn.app.middleware.session_cert, 143](#)
[dl_gmn.app.middleware.session_jwt, 143](#)
[dl_gmn.app.migrations, 144](#)
[dl_gmn.app.migrations.0001_initial, 144](#)
[dl_gmn.app.migrations.0002_scienceobject_filename, 144](#)
[dl_gmn.app.migrations.0003_mediatype_mediatypeproperties, 144](#)
[dl_gmn.app.migrations.0004_auto_20170523_0113, 144](#)
[dl_gmn.app.migrations.0005_auto_20170527_1554, 144](#)
[dl_gmn.app.migrations.0006_chainidtopersistentid, 145](#)
[dl_gmn.app.migrations.0007_delete_chainidtopersistentid, 145](#)
[dl_gmn.app.migrations.0008_chainidtopersistentid, 145](#)
[dl_gmn.app.migrations.0009_auto_20170603_0546, 145](#)
[dl_gmn.app.migrations.0010_auto_20170805_0107, 145](#)
[dl_gmn.app.migrations.0011_resourcemap_resourcemap, 145](#)
[dl_gmn.app.migrations.0012_auto_20170821_2129, 146](#)
[dl_gmn.app.migrations.0013_auto_20171017_0724, 146](#)
[dl_gmn.app.migrations.0014_auto_20171017_0725, 146](#)
[dl_gmn.app.migrations.0015_auto_20171029_0246, 146](#)
[dl_gmn.app.migrations.0016_auto_20180518_1539, 146](#)
[dl_gmn.app.migrations.0018_auto_20180901_0115, 146](#)
[dl_gmn.app.migrations.0019_auto_20190418_1512, 147](#)
[dl_gmn.app.model_util, 151](#)
[dl_gmn.app.node, 151](#)
[dl_gmn.app.node_registry, 151](#)
[dl_gmn.app.proxy, 152](#)
[dl_gmn.app.psycpg_adapter, 152](#)
[dl_gmn.app.resource_map, 152](#)
[dl_gmn.app.revision, 153](#)
[dl_gmn.app.scimeta, 154](#)
[dl_gmn.app.sciobj_store, 154](#)
[dl_gmn.app.settings_default, 156](#)
[dl_gmn.app.util, 156](#)
[dl_gmn.app.views, 147](#)
[dl_gmn.app.views.assert_db, 147](#)
[dl_gmn.app.views.assert_sysmeta, 148](#)
[dl_gmn.app.views.slice, 148](#)
[dl_gmn.manage, 161](#)
[dl_gmn.settings_template, 161](#)
[dl_gmn.settings_test, 161](#)
[dl_gmn.tests, 157](#)
[dl_gmn.tests.conftest, 157](#)
[dl_gmn.tests.gmn_test_client, 157](#)
[dl_gmn.tests.gmn_wsgi, 158](#)
[dl_gmn.version, 161](#)
[dl_onedrive, 48](#)
[dl_onedrive.impl, 48](#)
[dl_onedrive.impl.attributes, 54](#)
[dl_onedrive.impl.cache_memory, 54](#)
[dl_onedrive.impl.clients, 48](#)
[dl_onedrive.impl.clients.onedrive_dl_client, 48](#)
[dl_onedrive.impl.clients.onedrive_solr_client, 48](#)

<code>dl_onedrive.impl.clients.onedrive_zotero_client</code> , 49	<code>dl_test.instance_generator.format_id</code> , 301
<code>dl_onedrive.impl.clients.query_engine_description</code> , 49	<code>dl_test.instance_generator.identifier</code> , 302
<code>dl_onedrive.impl.directory</code> , 54	<code>dl_test.instance_generator.media_type</code> , 302
<code>dl_onedrive.impl.disk_cache</code> , 55	<code>dl_test.instance_generator.names</code> , 302
<code>dl_onedrive.impl.drivers</code> , 49	<code>dl_test.instance_generator.node_ref</code> , 302
<code>dl_onedrive.impl.drivers.dokan</code> , 49	<code>dl_test.instance_generator.person</code> , 302
<code>dl_onedrive.impl.drivers.dokan.const</code> , 49	<code>dl_test.instance_generator.random_data</code> , 303
<code>dl_onedrive.impl.drivers.fuse</code> , 50	<code>dl_test.instance_generator.replication_policy</code> , 305
<code>dl_onedrive.impl.log_decorator</code> , 55	<code>dl_test.instance_generator.replication_status</code> , 305
<code>dl_onedrive.impl.object_tree</code> , 55	<code>dl_test.instance_generator.subject</code> , 305
<code>dl_onedrive.impl.onedrive_exceptions</code> , 56	<code>dl_test.instance_generator.tests</code> , 300
<code>dl_onedrive.impl.os_escape</code> , 56	<code>dl_test.instance_generator.unicode_names</code> , 305
<code>dl_onedrive.impl.resolver</code> , 50	<code>dl_test.instance_generator.unicode_test_strings</code> , 305
<code>dl_onedrive.impl.resolver.author</code> , 50	<code>dl_test.instance_generator.user_agent</code> , 305
<code>dl_onedrive.impl.resolver.dl_object</code> , 50	<code>dl_test.instance_generator.words</code> , 305
<code>dl_onedrive.impl.resolver.flat_space</code> , 51	<code>dl_test.mock_api</code> , 306
<code>dl_onedrive.impl.resolver.object_tree_resolver</code> , 51	<code>dl_test.mock_api.create</code> , 307
<code>dl_onedrive.impl.resolver.region</code> , 51	<code>dl_test.mock_api.dl_exception</code> , 307
<code>dl_onedrive.impl.resolver.resolver_base</code> , 51	<code>dl_test.mock_api.django_client</code> , 307
<code>dl_onedrive.impl.resolver.resource_map</code> , 52	<code>dl_test.mock_api.generate_identifier</code> , 308
<code>dl_onedrive.impl.resolver.root</code> , 52	<code>dl_test.mock_api.ping</code> , 309
<code>dl_onedrive.impl.resolver.single</code> , 52	<code>dl_test.mock_api.post</code> , 309
<code>dl_onedrive.impl.resolver.taxa</code> , 52	<code>dl_test.mock_api.tests</code> , 306
<code>dl_onedrive.impl.resolver.time_period</code> , 53	<code>dl_test.replication_tester</code> , 309
<code>dl_onedrive.impl.tests</code> , 53	<code>dl_test.replication_tester.replication_error</code> , 309
<code>dl_onedrive.impl.tests.command_echoer</code> , 53	<code>dl_test.replication_tester.replication_server</code> , 309
<code>dl_onedrive.impl.tests.object_tree_test_client</code> , 53	<code>dl_test.replication_tester.replication_tester</code> , 310
<code>dl_onedrive.impl.tests.test_zotero_client</code> , 54	<code>dl_test.replication_tester.test_object_generator</code> , 310
<code>dl_onedrive.impl.util</code> , 57	<code>dl_test.slender_node_test_client</code> , 314
<code>dl_onedrive.onedrive</code> , 57	<code>dl_test.test_files</code> , 315
<code>dl_scimeta</code> , 286	<code>dl_test.tests</code> , 310
<code>dl_scimeta.gen_root_xsd</code> , 286	<code>dl_test.utilities</code> , 311
<code>dl_scimeta.lxml_validate</code> , 286	<code>dl_test.utilities.create_from_file</code> , 311
<code>dl_scimeta.schema_resolve</code> , 287	<code>dl_test.utilities.generate_test_subject_certs</code> , 311
<code>dl_scimeta.tests</code> , 286	<code>dl_test.utilities.my_subject</code> , 312
<code>dl_scimeta.tests.validate_test_docs</code> , 286	<code>dl_test.utilities.pem_in_http_header</code> , 313
<code>dl_scimeta.util</code> , 287	<code>dl_test.utilities.populate_mn</code> , 313
<code>dl_scimeta.validate</code> , 291	<code>dl_test.utilities.test_object_generator</code> , 313
<code>dl_test</code> , 300	
<code>dl_test.instance_generator</code> , 300	
<code>dl_test.instance_generator.checksum</code> , 301	

313
d1_test.utilities.urlencode, 313
d1_test.xml_normalize, 315
d1_util, 9
d1_util.cert_check_cn, 9
d1_util.cert_check_local, 10
d1_util.cert_create_ca, 10
d1_util.cert_create_csr, 11
d1_util.cert_sign_csr, 11
d1_util.check_object_checksums, 11
d1_util.check_x509_certificate_cn, 11
d1_util.check_x509_certificate_local,
12
d1_util.create_data_packages, 12
d1_util.create_object_on_member_node,
13
d1_util.delete_all_objects_of_type, 13
d1_util.display_node_status, 14
d1_util.download_all_objects, 15
d1_util.download_and_display_data_package,
15
d1_util.download_mn_objects, 16
d1_util.download_sciobj, 16
d1_util.download_server_certs, 16
d1_util.download_sysmeta, 17
d1_util.download_sysmeta_multiproc, 17
d1_util.download_test_docs, 17
d1_util.generate_object_stats, 18
d1_util.jwt_token_tasks, 18
d1_util.parse_format_id_list, 19
d1_util.resolve_package_identifier, 19
d1_util.solr_query, 19
d1_util.tests, 9
d1_util.xml_apply_xslt, 19
d1_util.xml_remove_empty_elements, 19
d1_util.xml_strip_whitespace, 20

Symbols

<code>__init__()</code> (<i>d1_client.aio.async_client.AsyncDataONEClient</i> method), 247	<code>__init__()</code> (<i>d1_common.logging_context.LoggingContext</i> method), 214
<code>__init__()</code> (<i>d1_client.baseclient.DataONEBaseClient</i> method), 257	<code>__init__()</code> (<i>d1_common.object_format_cache.ObjectFormatListCache</i> method), 216
<code>__init__()</code> (<i>d1_client.baseclient_1_1.DataONEBaseClient_1_1</i> method), 259	<code>__init__()</code> (<i>d1_common.resource_map.ResourceMap</i> method), 222
<code>__init__()</code> (<i>d1_client.baseclient_1_2.DataONEBaseClient_1_2</i> method), 260	<code>__init__()</code> (<i>d1_common.types.exceptions.DataONEException</i> method), 187
<code>__init__()</code> (<i>d1_client.baseclient_2_0.DataONEBaseClient_2_0</i> method), 260	<code>__init__()</code> (<i>d1_common.utils.progress_logger.ProgressLogger</i> method), 192
<code>__init__()</code> (<i>d1_client.cnclient.CoordinatingNodeClient</i> method), 261	<code>__init__()</code> (<i>d1_gmn.tests.gmn_test_client.GMNTTestClient</i> method), 157
<code>__init__()</code> (<i>d1_client.cnclient_1_1.CoordinatingNodeClient_1_1</i> method), 273	<code>__init__()</code> (<i>d1_gmn.tests.gmn_wsgi.WSGIClient</i> method), 158
<code>__init__()</code> (<i>d1_client.cnclient_1_2.CoordinatingNodeClient_1_2</i> method), 274	<code>__init__()</code> (<i>d1_test.slender_node_test_client.SlenderNodeTestClient</i> method), 314
<code>__init__()</code> (<i>d1_client.cnclient_2_0.CoordinatingNodeClient_2_0</i> method), 274	<code>__init__()</code> (<i>d1_test.xml_normalize.StableNode</i> method), 316
<code>__init__()</code> (<i>d1_client.d1client.DataONEClient</i> method), 276	
<code>__init__()</code> (<i>d1_client.iter.logrecord.LogRecordIterator</i> method), 250	A
<code>__init__()</code> (<i>d1_client.iter.objectlist.ObjectListIterator</i> method), 252	<code>abs_path()</code> (in module <i>d1_common.utils.filesystem</i>), 191
<code>__init__()</code> (<i>d1_client.mnclient.MemberNodeClient</i> method), 277	<code>abs_path_from_base()</code> (in module <i>d1_common.utils.filesystem</i>), 191
<code>__init__()</code> (<i>d1_client.mnclient_1_1.MemberNodeClient_1_1</i> method), 278	<code>AccessControl</code> (class in <i>d1_cli.impl.access_control</i>), 76
<code>__init__()</code> (<i>d1_client.mnclient_1_2.MemberNodeClient_1_2</i> method), 278	<code>AccessPolicyWrapper</code> (class in <i>d1_common.wrap.access_policy</i>), 197
<code>__init__()</code> (<i>d1_client.mnclient_2_0.MemberNodeClient_2_0</i> method), 279	<code>adapt_pyxb_binding()</code> (in module <i>d1_gmn.app.psycopg_adapter</i>), 152
<code>__init__()</code> (<i>d1_client.session.Session</i> method), 279	<code>add()</code> (<i>d1_client.solr_client.SolrClient</i> method), 283
<code>__init__()</code> (<i>d1_client.solr_client.SolrValuesResponseIterator</i> method), 284	<code>add_allowed_subject()</code> (<i>d1_cli.impl.access_control.AccessControl</i> method), 76
<code>__init__()</code> (<i>d1_common.date_time.FixedOffset</i> method), 208	<code>add_argument()</code> (<i>d1_common.iter.path.ArgumentParser</i> method), 182
<code>__init__()</code> (<i>d1_common.iter.path.ArgumentParser</i> method), 182	<code>add_arguments()</code> (<i>d1_gmn.app.management.commands.event.Command</i> method), 141
<code>__init__()</code> (<i>d1_common.iter.stream.StreamIterator</i> method), 183	<code>add_arguments()</code> (in module <i>d1_gmn.app.management.commands.util.standard_args</i>), 140
	<code>add_authenticated_read()</code>

(d1_common.wrap.access_policy.AccessPolicyWrapper method), 223
method), 198
 add_authenticated_read() (in module *d1_common.wrap.access_policy*), 199
 add_blocked() (*d1_cli.impl.replication_policy.ReplicationPolicy* property), 257
method), 83
 add_blocked() (in module *d1_common.replication_policy*), 219
 add_callback() (in module *d1_test.mock_api.create*), 307
 add_callback() (in module *d1_test.mock_api.django_client*), 308
 add_callback() (in module *d1_test.mock_api.generate_identifier*), 308
 add_callback() (in module *d1_test.mock_api.ping*), 309
 add_callback() (in module *d1_test.mock_api.post*), 309
 add_child() (*d1_common.cert.subject_info.SubjectInfoNode* method), 171
 add_child() (*d1_test.xml_normalize.StableNode* method), 316
 add_docs() (*d1_client.solr_client.SolrClient* method), 283
 add_object_to_cache() (*d1_onedrive.impl.object_tree.ObjectTree* method), 55
 add_perm() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 add_perm() (in module *d1_common.wrap.access_policy*), 200
 add_preferred() (*d1_cli.impl.replication_policy.ReplicationPolicy* method), 83
 add_preferred() (in module *d1_common.replication_policy*), 219
 add_public_read() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 add_public_read() (in module *d1_common.wrap.access_policy*), 199
 add_slice_filter() (in module *d1_gmn.app.views.slice*), 148
 add_verified_read() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 add_verified_read() (in module *d1_common.wrap.access_policy*), 200
 addDataDocuments() (*d1_common.resource_map.ResourceMap* method), 224
 addMetadataDocument() (*d1_common.resource_map.ResourceMap* method), 224
 addResource() (*d1_common.resource_map.ResourceMap* method), 223
 Apache, 23
 api_version_tuple() (*d1_client.baseclient.DataONEBaseClient* property), 257
 append() (*d1_cli.impl.operation_queue.OperationQueue* method), 82
 apply_xerces_adaption_schema_transform() (in module *d1_scimeta.validate*), 292
 apply_xslt_transform() (in module *d1_scimeta.util*), 291
 archive() (*d1_cli.impl.operation_maker.OperationMaker* method), 82
 archive() (*d1_client.baseclient.DataONEBaseClient* method), 258
 archiveResponse() (*d1_client.baseclient.DataONEBaseClient* method), 258
 Node_checksums_equal() (in module *d1_common.checksum*), 206
 are_equal() (in module *d1_common.date_time*), 209
 are_equal_elements() (in module *d1_common.xml*), 243
 are_equal_or_superset() (in module *d1_common.xml*), 243
 are_equal_pyxb() (in module *d1_common.xml*), 243
 are_equal_xml() (in module *d1_common.xml*), 243
 are_equivalent() (in module *d1_common.xml*), 243
 are_equivalent_pyxb() (in module *d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 are_equivalent_pyxb() (in module *d1_common.replication_policy*), 219
 are_equivalent_pyxb() (in module *d1_common.system_metadata*), 228
 are_equivalent_pyxb() (in module *d1_common.wrap.access_policy*), 199
 are_equivalent_pyxb() (in module *d1_common.xml*), 243
 are_equivalent_xml() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 are_equivalent_xml() (in module *d1_common.replication_policy*), 219
 are_equivalent_xml() (in module *d1_common.system_metadata*), 229
 are_equivalent_xml() (in module *d1_common.wrap.access_policy*), 199
 are_modules_importable() (in module *d1_cli.impl.check_dependencies*), 76
 ARG_DICT (*d1_common.iter.path.ArgParser* attribute), 182

- ArgParser (class in *d1_common.iter.path*), 181
 args () (*d1_common.iter.path.ArgParser* property), 182
 args_adapter () (in module *d1_client.command_line*), 276
 asGraphvizDot () (*d1_common.resource_map.ResourceMap* method), 225
 assert_file_exists () (in module *d1_cli.impl.util*), 84
 assert_map_is_valid_for_create () (in module *d1_gmn.app.resource_map*), 152
 assert_sciobj_store_does_not_exist () (in module *d1_gmn.app.sciobj_store*), 155
 assert_sciobj_store_exists () (in module *d1_gmn.app.sciobj_store*), 155
 assert_sciobj_store_version_match () (in module *d1_gmn.app.sciobj_store*), 156
 assert_valid () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid () (in module *d1_gmn.app.scimeta*), 154
 assert_valid () (in module *d1_scimeta.validate*), 292
 assert_valid_archive () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid_create () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid_create_package () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid_path () (in module *d1_util.cert_sign_csr*), 11
 assert_valid_update () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid_update_access_policy () (*d1_cli.impl.operation_validator.OperationValidator* method), 82
 assert_valid_update_replication_policy () (*d1_cli.impl.operation_validator.OperationValidator* method), 83
 AsyncDataONEClient (class in *d1_client.aio.async_client*), 247
 Attributes (class in *d1_onedrive.impl.attributes*), 54
 auth_subj_tup () (*d1_client.session.Session* property), 280
 AuthenticationTimeout, 188
- ## B
- base_name_without_extension () (in module *d1_util.create_data_packages*), 13
 base_url () (*d1_client.session.Session* property), 280
 Bash, 23
 best_match () (in module *d1_common.ext.mimeparser*), 179
 BytesIterator (class in *d1_common.iter.bytes*), 179
 CA, 21
 CA certificate, 22
 CA signing key, 21
 Cache (class in *d1_onedrive.impl.cache_memory*), 54
 cache_add_last_in_slice () (in module *d1_gmn.app.views.slice*), 148
 cache_is_stale () (*d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient* method), 49
 CACreateError, 10
 calculate_checksum () (in module *d1_gmn.app.views.assert_sysmeta*), 148
 calculate_checksum_on_bytes () (in module *d1_common.checksum*), 206
 calculate_checksum_on_iterator () (in module *d1_common.checksum*), 206
 calculate_checksum_on_stream () (in module *d1_common.checksum*), 206
 cast_naive_datetime_to_tz () (in module *d1_common.date_time*), 212
 cert_cleanup () (in module *d1_util.jwt_token_tasks*), 18
 Certificate, 22
 Chain of trust, 22
 check_cert_type () (in module *d1_common.cert.x509*), 177
 CILogon, 24
 classify_identifier () (in module *d1_gmn.app.did*), 150
 clear () (*d1_cli.impl.access_control.AccessControl* method), 76
 clear () (*d1_cli.impl.operation_queue.OperationQueue* method), 82
 clear () (*d1_cli.impl.replication_policy.ReplicationPolicy* method), 83
 clear () (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 clear () (*d1_onedrive.impl.disk_cache.DiskCache* method), 55
 clear () (in module *d1_common.wrap.access_policy*), 199
 clear_elements () (in module *d1_common.system_metadata*), 229
 clear_replication_queue () (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), 157
 CLI (class in *d1_cli.impl.command_parser*), 77
 CLIBaseClient (class in *d1_cli.impl.client*), 76
 CLIClient (class in *d1_cli.impl.client*), 76
 CLICNClient (class in *d1_cli.impl.client*), 76

client, [21](#)
 Client side authentication, [22](#)
 Client side certificate, [22](#)
 CLIErrors, [81](#)
 CLIMNClient (class in *d1_cli.impl.client*), [76](#)
 close() (*d1_client.aio.async_client.AsyncDataONEClient* method), [248](#)
 CN, [21](#)
 coerce_put_post() (in module *d1_gmn.app.util*), [156](#)
 Command (class in *d1_gmn.app.management.commands.event*), [141](#)
 CommandEchoer (class in *d1_onedrive.impl.tests.command_echoer*), [53](#)
 CommandProcessor (class in *d1_cli.impl.command_processor*), [80](#)
 commit() (*d1_client.solr_client.SolrClient* method), [283](#)
 CompareError, [245](#)
 completed() (*d1_common.utils.progress_logger.ProgressLogger* method), [193](#)
 concurrency_clear() (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), [158](#)
 concurrency_get_dictionary_id() (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), [158](#)
 concurrency_read_lock() (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), [158](#)
 concurrency_write_lock() (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), [158](#)
 confirm() (in module *d1_cli.impl.util*), [84](#)
 confirmMapIdentity() (*d1_client.cnclient.CoordinatingNodeClient* method), [269](#)
 confirmMapIdentityResponse() (*d1_client.cnclient.CoordinatingNodeClient* method), [269](#)
 CoordinatingNodeClient (class in *d1_client.cnclient*), [260](#)
 CoordinatingNodeClient_1_1 (class in *d1_client.cnclient_1_1*), [273](#)
 CoordinatingNodeClient_1_2 (class in *d1_client.cnclient_1_2*), [274](#)
 CoordinatingNodeClient_2_0 (class in *d1_client.cnclient_2_0*), [274](#)
 copy() (*d1_onedrive.impl.cache_memory.Cache* method), [54](#)
 copy_file_like_object_to_file() (in module *d1_cli.impl.util*), [84](#)
 copy_requests_stream_to_file() (in module *d1_cli.impl.util*), [84](#)
 count() (*d1_client.solr_client.SolrClient* method), [282](#)
 count() (*d1_common.util.EventCounter* method), [239](#)
 create() (*d1_cli.impl.operation_maker.OperationMaker* method), [82](#)
 create() (*d1_client.mnclient.MemberNodeClient* method), [277](#)
 create() (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), [158](#)
 create() (*d1_test.slender_node_test_client.SlenderNodeTestClient* method), [314](#)
 create_access_policy() (*d1_cli.impl.system_metadata.SystemMetadataCreator* method), [84](#)
 create_bagit_stream() (in module *d1_common.bagit*), [205](#)
 create_ca() (in module *d1_util.cert_create_ca*), [10](#)
 create_cert_request() (in module *d1_test.utilities.generate_test_subject_certs*), [311](#)
 create_certificate() (in module *d1_test.utilities.generate_test_subject_certs*), [312](#)
 create_checksum_object_from_bytes() (in module *d1_common.checksum*), [206](#)
 create_checksum_object_from_iterator() (in module *d1_common.checksum*), [205](#)
 create_checksum_object_from_stream() (in module *d1_common.checksum*), [205](#)
 create_client() (in module *d1_client.iter.base_multi*), [249](#)
 create_csr() (in module *d1_util.cert_create_csr*), [11](#)
 create_exception_by_error_code() (in module *d1_common.types.exceptions*), [187](#)
 create_exception_by_name() (in module *d1_common.types.exceptions*), [186](#)
 create_filtered_tree() (*d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient* method), [49](#)
 create_header_d1_exception() (in module *d1_test.mock_api.d1_exception*), [307](#)
 create_http_echo_response() (in module *d1_gmn.app.util*), [156](#)
 create_key_pair() (in module *d1_test.utilities.generate_test_subject_certs*), [311](#)
 create_lxml_obj() (in module *d1_scimeta.util*), [291](#)
 create_missing_directories_for_dir() (in module *d1_common.utils.filesystem*), [191](#)
 create_missing_directories_for_file() (in module *d1_common.utils.filesystem*), [191](#)
 create_mn_dn() (in module *d1_common.cert.x509*), [173](#)

[create_or_update\(\)](#) (in module [d1_gmn.app.resource_map](#)), 152
[create_or_update_chain\(\)](#) (in module [d1_gmn.app.revision](#)), 153
[create_or_update_db\(\)](#) (in module [d1_gmn.app.resource_map](#)), 152
[create_package\(\)](#) ([d1_cli.impl.command_processor.CommandProcessor](#) method), 80
[create_package\(\)](#) ([d1_cli.impl.operation_maker.OperationMaker](#) method), 82
[create_package_on_member_node\(\)](#) (in module [d1_util.create_data_packages](#)), 12
[create_regular_d1_exception\(\)](#) (in module [d1_test.mock_api.d1_exception](#)), 307
[create_replication_policy\(\)](#) ([d1_cli.impl.system_metadata.SystemMetadataCreator](#) method), 84
[create_replication_queue\(\)](#) ([d1_gmn.tests.gmn_test_client.GMNTTestClient](#) method), 157
[create_resource_map_for_pids\(\)](#) (in module [d1_util.create_data_packages](#)), 13
[create_root_cn_client\(\)](#) (in module [d1_gmn.app.node_registry](#)), 152
[create_science_object_on_member_node\(\)](#) (in module [d1_util.create_data_packages](#)), 12
[create_sciobj\(\)](#) ([d1_client.d1client.DataONEClient](#) method), 276
[create_session_extension\(\)](#) (in module [d1_test.utilities.generate_test_subject_certs](#)), 312
[create_simple_dn\(\)](#) (in module [d1_common.cert.x509](#)), 174
[create_store\(\)](#) (in module [d1_gmn.app.sciobj_store](#)), 156
[create_sysmeta\(\)](#) ([d1_client.d1client.DataONEClient](#) method), 277
[create_system_metadata\(\)](#) ([d1_cli.impl.system_metadata.SystemMetadataCreator](#) method), 84
[create_system_metadata_for_package\(\)](#) ([d1_cli.impl.system_metadata.SystemMetadataCreator](#) method), 84
[create_system_metadata_for_update\(\)](#) ([d1_cli.impl.system_metadata.SystemMetadataCreator](#) method), 84
[create_test_object_on_mn\(\)](#) (in module [d1_test.replication_tester.replication_tester](#)), 310
[create_utc_datetime\(\)](#) (in module [d1_common.date_time](#)), 213
[createGroup\(\)](#) ([d1_client.cnclient.CoordinatingNodeClient](#) method), 270
[createGroupResponse\(\)](#) ([d1_client.cnclient.CoordinatingNodeClient](#) method), 269
[createResourceMapFromStream\(\)](#) (in module [d1_common.resource_map](#)), 221
[createResponse\(\)](#) ([d1_client.mnclient.MemberNodeClient](#) method), 277
[createResponseForResourceMap\(\)](#) (in module [d1_common.resource_map](#)), 221
[CSR](#), 22
[CSRCreateError](#), 11
[CSRSignKeyError](#), 11
[cut_from_chain\(\)](#) (in module [d1_gmn.app.revision](#)), 153

D

[d1_cli](#) (module), 76
[d1_cli.dataone](#) (module), 85
[d1_cli.impl](#) (module), 76
[d1_cli.impl.access_control](#) (module), 76
[d1_cli.impl.check_dependencies](#) (module), 76
[d1_cli.impl.client](#) (module), 76
[d1_cli.impl.command_parser](#) (module), 77
[d1_cli.impl.command_processor](#) (module), 80
[d1_cli.impl.const](#) (module), 81
[d1_cli.impl.exceptions](#) (module), 81
[d1_cli.impl.format_ids](#) (module), 81
[d1_cli.impl.nodes](#) (module), 81
[d1_cli.impl.operation_executer](#) (module), 81
[d1_cli.impl.operation_formatter](#) (module), 81
[d1_cli.impl.operation_maker](#) (module), 82
[d1_cli.impl.operation_queue](#) (module), 82
[d1_cli.impl.operation_validator](#) (module), 82
[d1_cli.impl.replication_policy](#) (module), 83
[d1_cli.impl.session](#) (module), 83
[d1_cli.impl.system_metadata](#) (module), 84
[d1_cli.impl.util](#) (module), 84
[d1_cli.tests](#) (module), 84
[d1_cli.version](#) (module), 85
[d1_client](#) (module), 247
[d1_client.aio](#) (module), 247
[d1_client.aio.async_client](#) (module), 247
[d1_client.baseclient](#) (module), 255
[d1_client.baseclient_1_1](#) (module), 258
[d1_client.baseclient_1_2](#) (module), 260
[d1_client.baseclient_2_0](#) (module), 260
[d1_client.cnclient](#) (module), 260
[d1_client.cnclient_1_1](#) (module), 273
[d1_client.cnclient_1_2](#) (module), 274

`dl_client.cnclient_2_0 (module)`, 274
`dl_client.command_line (module)`, 275
`dl_client.dlclient (module)`, 276
`dl_client.iter (module)`, 248
`dl_client.iter.base_multi (module)`, 248
`dl_client.iter.logrecord (module)`, 249
`dl_client.iter.logrecord_multi (module)`, 250
`dl_client.iter.node (module)`, 251
`dl_client.iter.objectlist (module)`, 251
`dl_client.iter.objectlist_multi (module)`, 253
`dl_client.iter.sysmeta_multi (module)`, 253
`dl_client.mnclient (module)`, 277
`dl_client.mnclient_1_1 (module)`, 278
`dl_client.mnclient_1_2 (module)`, 278
`dl_client.mnclient_2_0 (module)`, 279
`dl_client.session (module)`, 279
`dl_client.solr_client (module)`, 281
`dl_client.tests (module)`, 254
`dl_client.tests.shared_context (module)`, 254
`dl_client.util (module)`, 284
`dl_common (module)`, 163
`dl_common.bagit (module)`, 205
`dl_common.cert (module)`, 163
`dl_common.cert.jwt (module)`, 164
`dl_common.cert.subject_info (module)`, 167
`dl_common.cert.subjects (module)`, 172
`dl_common.cert.x509 (module)`, 173
`dl_common.checksum (module)`, 205
`dl_common.const (module)`, 208
`dl_common.date_time (module)`, 208
`dl_common.env (module)`, 214
`dl_common.ext (module)`, 178
`dl_common.ext.mimeparser (module)`, 178
`dl_common.iter (module)`, 179
`dl_common.iter.bytes (module)`, 179
`dl_common.iter.path (module)`, 180
`dl_common.iter.stream (module)`, 183
`dl_common.iter.string (module)`, 183
`dl_common.logging_context (module)`, 214
`dl_common.multipart (module)`, 214
`dl_common.node (module)`, 215
`dl_common.object_format_cache (module)`, 216
`dl_common.replication_policy (module)`, 217
`dl_common.resource_map (module)`, 220
`dl_common.revision (module)`, 226
`dl_common.system_metadata (module)`, 227
`dl_common.tests (module)`, 183
`dl_common.type_conversions (module)`, 231
`dl_common.types (module)`, 184
`dl_common.types.dataoneErrors (module)`, 185
`dl_common.types.dataoneTypes (module)`, 185
`dl_common.types.dataoneTypes_v1 (module)`, 185
`dl_common.types.dataoneTypes_v1_1 (module)`, 185
`dl_common.types.dataoneTypes_v1_2 (module)`, 185
`dl_common.types.dataoneTypes_v2_0 (module)`, 186
`dl_common.types.exceptions (module)`, 186
`dl_common.url (module)`, 237
`dl_common.util (module)`, 238
`dl_common.utils (module)`, 190
`dl_common.utils.filesystem (module)`, 190
`dl_common.utils.progress_logger (module)`, 192
`dl_common.wrap (module)`, 193
`dl_common.wrap.access_policy (module)`, 194
`dl_common.wrap.simple_xml (module)`, 200
`dl_common.xml (module)`, 241
`dl_gmn (module)`, 140
`dl_gmn.app (module)`, 140
`dl_gmn.app.context_processors (module)`, 149
`dl_gmn.app.did (module)`, 149
`dl_gmn.app.gmn (module)`, 150
`dl_gmn.app.management (module)`, 140
`dl_gmn.app.management.commands (module)`, 140
`dl_gmn.app.management.commands.event (module)`, 141
`dl_gmn.app.management.commands.logininfo_async (module)`, 141
`dl_gmn.app.management.commands.objectlist_async (module)`, 142
`dl_gmn.app.management.commands.util (module)`, 140
`dl_gmn.app.management.commands.util.standard_args (module)`, 140
`dl_gmn.app.middleware (module)`, 142
`dl_gmn.app.middleware.detail_codes (module)`, 142
`dl_gmn.app.middleware.exception_handler (module)`, 142
`dl_gmn.app.middleware.profilng_handler (module)`, 143
`dl_gmn.app.middleware.session_cert (module)`, 143
`dl_gmn.app.middleware.session_jwt (module)`, 143
`dl_gmn.app.migrations (module)`, 144
`dl_gmn.app.migrations.0001_initial (mod-`

[ule](#), 144
[dl_gmn.app.migrations.0002_scienceobject_filename](#) ([module](#)), 144
[dl_gmn.app.migrations.0003_mediatype_mediatypeproperties](#) ([module](#)), 144
[dl_gmn.app.migrations.0004_auto_20170523_0137](#) ([module](#)), 144
[dl_gmn.app.migrations.0005_auto_20170527_1554](#) ([module](#)), 144
[dl_gmn.app.migrations.0006_chainidtopersistentid](#) ([module](#)), 145
[dl_gmn.app.migrations.0007_delete_chainidtopersistentid](#) ([module](#)), 145
[dl_gmn.app.migrations.0008_chainidtopersistentid](#) ([module](#)), 145
[dl_gmn.app.migrations.0009_auto_20170603_0546](#) ([module](#)), 145
[dl_gmn.app.migrations.0010_auto_20170805_0107](#) ([module](#)), 145
[dl_gmn.app.migrations.0011_resourcemap_resourceidmember](#) ([module](#)), 145
[dl_gmn.app.migrations.0012_auto_20170821_2129](#) ([module](#)), 146
[dl_gmn.app.migrations.0013_auto_20171017_0724](#) ([module](#)), 146
[dl_gmn.app.migrations.0014_auto_20171017_0725](#) ([module](#)), 146
[dl_gmn.app.migrations.0015_auto_20171029_0246](#) ([module](#)), 146
[dl_gmn.app.migrations.0016_auto_20180518_1539](#) ([module](#)), 146
[dl_gmn.app.migrations.0018_auto_20180901_0105](#) ([module](#)), 146
[dl_gmn.app.migrations.0019_auto_20190418_1502](#) ([module](#)), 147
[dl_gmn.app.model_util](#) ([module](#)), 151
[dl_gmn.app.node](#) ([module](#)), 151
[dl_gmn.app.node_registry](#) ([module](#)), 151
[dl_gmn.app.proxy](#) ([module](#)), 152
[dl_gmn.app.psycpg_adapter](#) ([module](#)), 152
[dl_gmn.app.resource_map](#) ([module](#)), 152
[dl_gmn.app.revision](#) ([module](#)), 153
[dl_gmn.app.scimeta](#) ([module](#)), 154
[dl_gmn.app.sciobj_store](#) ([module](#)), 154
[dl_gmn.app.settings_default](#) ([module](#)), 156
[dl_gmn.app.util](#) ([module](#)), 156
[dl_gmn.app.views](#) ([module](#)), 147
[dl_gmn.app.views.assert_db](#) ([module](#)), 147
[dl_gmn.app.views.assert_sysmeta](#) ([module](#)), 148
[dl_gmn.app.views.slice](#) ([module](#)), 148
[dl_gmn.manage](#) ([module](#)), 161
[dl_gmn.settings_template](#) ([module](#)), 161
[dl_gmn.settings_test](#) ([module](#)), 161
[dl_gmn.tests](#) ([module](#)), 157
[dl_gmn.tests.conftest](#) ([module](#)), 157
[dl_gmn.tests.gmn_test_client](#) ([module](#)), 157
[dl_gmn.tests.gmn_wsgi](#) ([module](#)), 158
[dl_gmn.version](#) ([module](#)), 161
[dl_onedrive](#) ([module](#)), 48
[dl_onedrive.impl](#) ([module](#)), 48
[dl_onedrive.impl.attributes](#) ([module](#)), 54
[dl_onedrive.impl.cache_memory](#) ([module](#)), 54
[dl_onedrive.impl.clients](#) ([module](#)), 48
[dl_onedrive.impl.clients.onedrive_dl_client](#) ([module](#)), 48
[dl_onedrive.impl.clients.onedrive_solr_client](#) ([module](#)), 48
[dl_onedrive.impl.clients.onedrive_zotero_client](#) ([module](#)), 49
[dl_onedrive.impl.clients.query_engine_description](#) ([module](#)), 49
[dl_onedrive.impl.directory](#) ([module](#)), 54
[dl_onedrive.impl.disk_cache](#) ([module](#)), 55
[dl_onedrive.impl.drivers](#) ([module](#)), 49
[dl_onedrive.impl.drivers.dokan](#) ([module](#)), 49
[dl_onedrive.impl.drivers.dokan.const](#) ([module](#)), 49
[dl_onedrive.impl.drivers.fuse](#) ([module](#)), 50
[dl_onedrive.impl.log_decorator](#) ([module](#)), 55
[dl_onedrive.impl.object_tree](#) ([module](#)), 55
[dl_onedrive.impl.onedrive_exceptions](#) ([module](#)), 56
[dl_onedrive.impl.os_escape](#) ([module](#)), 56
[dl_onedrive.impl.resolver](#) ([module](#)), 50
[dl_onedrive.impl.resolver.author](#) ([module](#)), 50
[dl_onedrive.impl.resolver.dl_object](#) ([module](#)), 50
[dl_onedrive.impl.resolver.flat_space](#) ([module](#)), 51
[dl_onedrive.impl.resolver.object_tree_resolver](#) ([module](#)), 51
[dl_onedrive.impl.resolver.region](#) ([module](#)), 51
[dl_onedrive.impl.resolver.resolver_base](#) ([module](#)), 51
[dl_onedrive.impl.resolver.resource_map](#) ([module](#)), 52
[dl_onedrive.impl.resolver.root](#) ([module](#)), 52
[dl_onedrive.impl.resolver.single](#) ([module](#)), 52
[dl_onedrive.impl.resolver.taxa](#) ([module](#)), 52
[dl_onedrive.impl.resolver.time_period](#)

[\(module\)](#), 53

[dl_onedrive.impl.tests \(module\)](#), 53

[dl_onedrive.impl.tests.command_echoer \(module\)](#), 53

[dl_onedrive.impl.tests.object_tree_test_sample \(module\)](#), 53

[dl_onedrive.impl.tests.test_zotero_client \(module\)](#), 54

[dl_onedrive.impl.util \(module\)](#), 57

[dl_onedrive.onedrive \(module\)](#), 57

[dl_scimeta \(module\)](#), 286

[dl_scimeta.gen_root_xsd \(module\)](#), 286

[dl_scimeta.lxml_validate \(module\)](#), 286

[dl_scimeta.schema_resolve \(module\)](#), 287

[dl_scimeta.tests \(module\)](#), 286

[dl_scimeta.tests.validate_test_docs \(module\)](#), 286

[dl_scimeta.util \(module\)](#), 287

[dl_scimeta.validate \(module\)](#), 291

[dl_test \(module\)](#), 300

[dl_test.instance_generator \(module\)](#), 300

[dl_test.instance_generator.checksum \(module\)](#), 301

[dl_test.instance_generator.format_id \(module\)](#), 301

[dl_test.instance_generator.identifier \(module\)](#), 302

[dl_test.instance_generator.media_type \(module\)](#), 302

[dl_test.instance_generator.names \(module\)](#), 302

[dl_test.instance_generator.node_ref \(module\)](#), 302

[dl_test.instance_generator.person \(module\)](#), 302

[dl_test.instance_generator.random_data \(module\)](#), 303

[dl_test.instance_generator.replication_policy \(module\)](#), 305

[dl_test.instance_generator.replication_status \(module\)](#), 305

[dl_test.instance_generator.subject \(module\)](#), 305

[dl_test.instance_generator.tests \(module\)](#), 300

[dl_test.instance_generator.unicode_names \(module\)](#), 305

[dl_test.instance_generator.unicode_test_strings \(module\)](#), 305

[dl_test.instance_generator.user_agent \(module\)](#), 305

[dl_test.instance_generator.words \(module\)](#), 305

[dl_test.mock_api \(module\)](#), 306

[dl_test.mock_api.create \(module\)](#), 307

[dl_test.mock_api.dl_exception \(module\)](#), 307

[dl_test.mock_api.django_client \(module\)](#), 307

[dl_test.mock_api.generate_identifier \(module\)](#), 308

[dl_test.mock_api.ping \(module\)](#), 309

[dl_test.mock_api.post \(module\)](#), 309

[dl_test.mock_api.tests \(module\)](#), 306

[dl_test.replication_tester \(module\)](#), 309

[dl_test.replication_tester.replication_error \(module\)](#), 309

[dl_test.replication_tester.replication_server \(module\)](#), 309

[dl_test.replication_tester.replication_tester \(module\)](#), 310

[dl_test.replication_tester.test_object_generator \(module\)](#), 310

[dl_test.slender_node_test_client \(module\)](#), 314

[dl_test.test_files \(module\)](#), 315

[dl_test.tests \(module\)](#), 310

[dl_test.utilities \(module\)](#), 311

[dl_test.utilities.create_from_file \(module\)](#), 311

[dl_test.utilities.generate_test_subject_certs \(module\)](#), 311

[dl_test.utilities.my_subject \(module\)](#), 312

[dl_test.utilities.pem_in_http_header \(module\)](#), 313

[dl_test.utilities.populate_mn \(module\)](#), 313

[dl_test.utilities.test_object_generator \(module\)](#), 313

[dl_test.utilities.urlencode \(module\)](#), 313

[dl_test.xml_normalize \(module\)](#), 315

[dl_util \(module\)](#), 9

[dl_util.cert_check_cn \(module\)](#), 9

[dl_util.cert_check_local \(module\)](#), 10

[dl_util.cert_create_ca \(module\)](#), 10

[dl_util.cert_create_csr \(module\)](#), 11

[dl_util.cert_sign_csr \(module\)](#), 11

[dl_util.check_object_checksums \(module\)](#), 11

[dl_util.check_x509_certificate_cn \(module\)](#), 11

[dl_util.check_x509_certificate_local \(module\)](#), 12

[dl_util.create_data_packages \(module\)](#), 12

[dl_util.create_object_on_member_node \(module\)](#), 13

[dl_util.delete_all_objects_of_type \(module\)](#), 13

`d1_util.display_node_status (module)`, 14
`d1_util.download_all_objects (module)`, 15
`d1_util.download_and_display_data_package (module)`, 15
`d1_util.download_mn_objects (module)`, 16
`d1_util.download_sciobj (module)`, 16
`d1_util.download_server_certs (module)`, 16
`d1_util.download_sysmeta (module)`, 17
`d1_util.download_sysmeta_multiproc (module)`, 17
`d1_util.download_test_docs (module)`, 17
`d1_util.generate_object_stats (module)`, 18
`d1_util.jwt_token_tasks (module)`, 18
`d1_util.parse_format_id_list (module)`, 19
`d1_util.resolve_package_identifier (module)`, 19
`d1_util.solr_query (module)`, 19
`d1_util.tests (module)`, 9
`d1_util.xml_apply_xslt (module)`, 19
`d1_util.xml_remove_empty_elements (module)`, 19
`d1_util.xml_strip_whitespace (module)`, 20
DataONE, 20
DataONE Client Library for Python, 20
DataONE Common Library for Python, 20
DataONE Test Utilities for Python, 20
DataONEBaseClient (class in `d1_client.baseclient`), 255
DataONEBaseClient_1_1 (class in `d1_client.baseclient_1_1`), 258
DataONEBaseClient_1_2 (class in `d1_client.baseclient_1_2`), 260
DataONEBaseClient_2_0 (class in `d1_client.baseclient_2_0`), 260
DataONEClient (class in `d1_client.d1client`), 276
DataONEClient (class in `d1_onedrive.impl.clients.onedrive_d1_client`), 48
DataONEException, 187
DataoneExceptionToDetailCode (class in `d1_gmn.app.middleware.detail_codes`), 142
date () (`d1_onedrive.impl.attributes.Attributes` method), 54
date_is_utc () (in `d1_gmn.app.views.assert_db`), 147
date_utc_now_iso () (in `d1_common.date_time`), 213
decode_bu64 () (in `module d1_common.cert.jwt`), 166
decode_der () (in `module d1_common.cert.x509`), 176
decodePathElement () (in `module d1_common.url`), 237
decodeQueryElement () (in `module d1_common.url`), 237
default () (`d1_cli.impl.command_parser.CLI` method), 77
default () (`d1_common.util.ToJsonCompatibleTypes` method), 241
delete () (`d1_client.cnclient_2_0.CoordinatingNodeClient_2_0` method), 274
delete () (`d1_client.mnclient.MemberNodeClient` method), 278
DELETE () (`d1_client.session.Session` method), 281
delete () (`d1_client.solr_client.SolrClient` method), 283
delete_all_access_policies () (`d1_gmn.tests.gmn_test_client.GMNTTestClient` method), 157
delete_all_objects () (`d1_gmn.tests.gmn_test_client.GMNTTestClient` method), 158
delete_by_query () (`d1_client.solr_client.SolrClient` method), 283
delete_chain () (in `module d1_gmn.app.revision`), 153
delete_event_log () (`d1_gmn.tests.gmn_test_client.GMNTTestClient` method), 158
delete_objects_from_member_node () (`d1_util.delete_all_objects_of_type.MemberNodeObjectDeleter` method), 14
delete_sciobj () (in `module d1_gmn.app.sciobj_store`), 156
delete_unused_subjects () (in `module d1_gmn.app.model_util`), 151
deleteReplicationMetadata () (`d1_client.cnclient.CoordinatingNodeClient` method), 272
deleteReplicationMetadataResponse () (`d1_client.cnclient.CoordinatingNodeClient` method), 272
deleteResponse () (`d1_client.cnclient_2_0.CoordinatingNodeClient_2_0` method), 274
deleteResponse () (`d1_client.mnclient.MemberNodeClient` method), 278
denyMapIdentity () (`d1_client.cnclient.CoordinatingNodeClient` method), 269
denyMapIdentityResponse () (`d1_client.cnclient.CoordinatingNodeClient` method), 268
dependencies (`d1_gmn.app.migrations.0001_initial.Migration` attribute), 144
dependencies (`d1_gmn.app.migrations.0002_scienceobject_filename.M` attribute), 144
dependencies (`d1_gmn.app.migrations.0003_mediatype_mediatypeprop` attribute), 144

dependencies (*d1_gmn.app.migrations.0004_auto_20170523_10137_Migration* (*d1_gmn.app.middleware.detail_codes.DataoneException* attribute), 144 method), 142

dependencies (*d1_gmn.app.migrations.0005_auto_20170527_1554_Migration* (in module *d1_common.replication_policy*), 220 attribute), 144

dependencies (*d1_gmn.app.migrations.0006_chainidtopersistent_Migration* (*d1_onedrive.impl.directory*), 54 attribute), 145 *disable_cert_validation()* (in module *d1_common.replication_policy*), 220

dependencies (*d1_gmn.app.migrations.0007_delete_chainidtopersistent_Migration* (*d1_onedrive.impl.disk_cache*), 55 attribute), 145 *DiskCache* (class in *d1_onedrive.impl.disk_cache*), 55

dependencies (*d1_gmn.app.migrations.0008_chainidtopersistent_Migration* (*d1_onedrive.impl.operation_queue.OperationQueue* attribute), 145 method), 82

dependencies (*d1_gmn.app.migrations.0009_auto_20170603_0543_Migration* (*d1_onedrive.impl.operation_queue.OperationQueue* attribute), 145 DN, 22

dependencies (*d1_gmn.app.migrations.0010_auto_20170805_10107_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 145 method), 78

dependencies (*d1_gmn.app.migrations.0011_resource_map_resource_map_member_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 145 method), 78

dependencies (*d1_gmn.app.migrations.0012_auto_20170824_2129_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 79

dependencies (*d1_gmn.app.migrations.0013_auto_20171017_10724_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 78

dependencies (*d1_gmn.app.migrations.0014_auto_20171017_10725_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 78

dependencies (*d1_gmn.app.migrations.0015_auto_20171029_10246_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 80

dependencies (*d1_gmn.app.migrations.0016_auto_20180518_1539_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 78

dependencies (*d1_gmn.app.migrations.0018_auto_20180901_09154_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 146 method), 79

dependencies (*d1_gmn.app.migrations.0019_auto_20190418_1512_Migration* (*d1_cli.impl.command_parser.CLI* attribute), 147 method), 78

describe() (*d1_client.aio.async_client.AsyncDataONEClient* *denyrep()* (*d1_cli.impl.command_parser.CLI* method), 248 method), 78

describe() (*d1_client.baseclient.DataONEBaseClient* *do_edit()* (*d1_cli.impl.command_parser.CLI* method), 258 method), 80

describe() (*d1_onedrive.impl.clients.onedrive_d1_client.DataONEClient* (*d1_cli.impl.command_parser.CLI* method), 48 method), 77

describeResponse() (*d1_client.baseclient.DataONEBaseClient* *do_exit()* (*d1_cli.impl.command_parser.CLI* method), 258 method), 77

deserialize() (*d1_common.resource_map.ResourceMap* *do_get()* (*d1_cli.impl.command_parser.CLI* method), 223 method), 78

deserialize() (in module *d1_common.types.exceptions*), 186 *do_GET()* (*d1_test.replication_tester.replication_server.Handler* method), 310

deserialize() (in module *d1_common.xml*), 241 *do_help()* (*d1_cli.impl.command_parser.CLI* method), 77

deserialize_d1_exception() (in module *d1_common.xml*), 241 *do_history()* (*d1_cli.impl.command_parser.CLI* method), 77

deserialize_from_headers() (in module *d1_common.types.exceptions*), 186 *do_list()* (*d1_cli.impl.command_parser.CLI* method), 79

deserialize_pem() (in module *d1_common.cert.x509*), 174 *do_listformats()* (*d1_cli.impl.command_parser.CLI* method), 79

deserialize_pem_file() (in module *d1_common.cert.x509*), 174 *do_listnodes()* (*d1_cli.impl.command_parser.CLI* method), 79

deserialize_subject_info() (in module *d1_common.cert.subject_info*), 170 *do_load()* (*d1_cli.impl.command_parser.CLI* method), 77

`do_log()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_meta()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_numberrep()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_package()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_ping()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_preferrep()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_PUT()` (*d1_test.replication_tester.replication_server.Handler method*), 310
`do_queue()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_quit()` (*d1_cli.impl.command_parser.CLI method*), 77
`do_removeverep()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_reset()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_resolve()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_run()` (*d1_cli.impl.command_parser.CLI method*), 80
`do_save()` (*d1_cli.impl.command_parser.CLI method*), 78
`do_search()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_set()` (*d1_cli.impl.command_parser.CLI method*), 77
`do_update()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_updateaccess()` (*d1_cli.impl.command_parser.CLI method*), 79
`do_updatereplication()` (*d1_cli.impl.command_parser.CLI method*), 79
Dokan, 23
`download()` (in module *d1_util.download_test_docs*), 17
`download_all()` (*d1_util.download_mn_objects.MemberNodeObjectDownloader method*), 16
`download_as_der()` (in module *d1_common.cert.x509*), 175
`download_as_obj()` (in module *d1_common.cert.x509*), 175
`download_as_pem()` (in module *d1_common.cert.x509*), 175
`download_cn_certs()` (in module *d1_util.jwt_token_tasks*), 18
`download_d1_object()` (*d1_util.download_all_objects.MemberNodeObjectDownloader method*), 15
`download_node_registry()` (in module *d1_gmn.app.node_registry*), 152
`download_objects_from_member_node()` (*d1_util.download_all_objects.MemberNodeObjectDownloader method*), 15
`download_schema()` (in module *d1_scimeta.schema_resolve*), 287
`download_server_cert()` (in module *d1_util.download_server_certs*), 17
DownloadError, 16
`dst()` (*d1_common.date_time.FixedOffset method*), 209
`dst()` (*d1_common.date_time.UTC method*), 208
`dt_from_http_datetime_str()` (in module *d1_common.date_time*), 211
`dt_from_iso8601_str()` (in module *d1_common.date_time*), 211
`dt_from_ts()` (in module *d1_common.date_time*), 210
`dump()` (*d1_common.wrap.access_policy.AccessPolicyWrapper method*), 197
`dump()` (in module *d1_common.wrap.access_policy*), 199
`dump()` (in module *d1_scimeta.util*), 291
`dump_headers()` (*d1_client.aio.async_client.AsyncDataONEClient method*), 248
`dump_pretty_tree()` (in module *d1_scimeta.util*), 291
`dump_request_and_response()` (*d1_client.session.Session method*), 281
`dump_stack()` (in module *d1_gmn.app.util*), 156
`dump_to_log()` (*d1_common.util.EventCounter method*), 239
`dumpExtensions()` (in module *d1_test.utilities.my_subject*), 313

E

`echo_raw_post_data()` (*d1_gmn.tests.gmn_test_client.GMNTTestClient method*), 158
`echo_request_object()` (*d1_gmn.tests.gmn_test_client.GMNTTestClient method*), 158
`echo_session()` (*d1_gmn.tests.gmn_test_client.GMNTTestClient method*), 157
`echoCredentials()` (*d1_client.cnclient_2_0.CoordinatingNodeClient_2_0 method*), 275
`echoCredentialsResponse()` (*d1_client.cnclient_2_0.CoordinatingNodeClient_2_0 method*), 275
`echoIndexedObject()` (*d1_client.cnclient_2_0.CoordinatingNodeClient_2_0 method*), 275

[echoIndexedObjectResponse\(\)](#) ([d1_client.cnclient_2_0.CoordinatingNodeClient_2_0](#) [extract_subject_from_dn\(\)](#) (in module [d1_common.cert.x509](#)), 176
[method](#)), 275
[echoSystemMetadata\(\)](#) ([d1_client.cnclient_2_0.CoordinatingNodeClient_2_0](#) [extract_subject_info_extension\(\)](#) (in module [d1_common.cert.x509](#)), 175
[method](#)), 275
[echoSystemMetadataResponse\(\)](#) ([d1_client.cnclient_2_0.CoordinatingNodeClient_2_0](#) [extract_subjects\(\)](#) (in module [d1_common.cert.subject_info](#)), 169
[method](#)), 275
[edit\(\)](#) ([d1_cli.impl.operation_queue.OperationQueue](#) [extract_subjects\(\)](#) (in module [d1_common.cert.subjects](#)), 172
[method](#)), 82
[emptyline\(\)](#) ([d1_cli.impl.command_parser.CLI](#) [extract_version_tag_from_url\(\)](#) (in module [d1_common.type_conversions](#)), 232
[method](#)), 77
[encode\(\)](#) ([d1_common.types.exceptions.DataONEException](#) [method](#)), 188
F
[encode_bu64\(\)](#) (in module [d1_common.cert.jwt](#)), 166
[encodeAndJoinPathElements\(\)](#) (in module [d1_common.url](#)), 237
[encodePathElement\(\)](#) (in module [d1_common.url](#)), 237
[encodeQueryElement\(\)](#) (in module [d1_common.url](#)), 237
[end_task_type\(\)](#) ([d1_common.utils.progress_logger.ProgressLogger](#) [find_files_in_group\(\)](#) (in module [d1_util.create_data_packages](#)), 13
[method](#)), 193
[ensure_dir_exists\(\)](#) (in module [d1_onedrive.impl.util](#)), 57
[etree_replace_namespace\(\)](#) (in module [d1_common.type_conversions](#)), 236
[etree_to_pretty_xml\(\)](#) (in module [d1_common.type_conversions](#)), 235
[etree_to_pyxb\(\)](#) (in module [d1_common.type_conversions](#)), 235
[etree_to_stable_tree\(\)](#) (in module [d1_test.xml_normalize](#)), 316
[etree_to_str\(\)](#) (in module [d1_common.type_conversions](#)), 235
[event\(\)](#) ([d1_common.utils.progress_logger.ProgressLogger](#) [find_object_size_stats_node\(\)](#) (in module [d1_util.generate_object_stats](#)), 18
[method](#)), 193
[event_dict\(\)](#) ([d1_common.util.EventCounter](#) [find_object_size_stats_node_all\(\)](#) (in module [d1_util.generate_object_stats](#)), 18
[property](#)), 239
[EventCounter](#) (class in [d1_common.util](#)), 238
[EventLogIteratorAsync](#) (class in [d1_gmn.app.management.commands.loginfo_async](#)), 141
[exception\(\)](#) ([d1_gmn.tests.gmn_test_client.GMNTestClient](#) [find_url_mismatches\(\)](#) (in module [d1_common.url](#)), 238
[method](#)), 158
[ExceptionHandler](#) (class in [d1_gmn.app.middleware.exception_handler](#)), 143
[execute\(\)](#) ([d1_cli.impl.operation_executer.OperationExecuter](#) [find_valid_combinations\(\)](#) (in module [d1_util.jwt_token_tasks](#)), 18
[method](#)), 81
[execute\(\)](#) ([d1_cli.impl.operation_queue.OperationQueue](#) [fitness_and_quality_parsed\(\)](#) (in module [d1_common.ext.mimeparser](#)), 179
[method](#)), 82
[extract_issuer_ca_cert_url\(\)](#) (in module [d1_common.date_time](#)), 208
[extract_subject_from_dn\(\)](#) (in module [d1_common.cert.x509](#)), 176
[extract_subject_info_extension\(\)](#) (in module [d1_common.cert.x509](#)), 175
[extract_subjects\(\)](#) (in module [d1_common.cert.subject_info](#)), 169
[extract_version_tag_from_url\(\)](#) (in module [d1_common.type_conversions](#)), 232
F
[field_alpha_histogram\(\)](#) ([d1_client.solr_client.SolrClient](#) [force_refresh\(\)](#) ([d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient](#) [method](#)), 49
[method](#)), 283
[filename_from_cert_obj\(\)](#) (in module [d1_util.jwt_token_tasks](#)), 18
[filename_from_identifier\(\)](#) (in module [d1_onedrive.impl.os_escape](#)), 56
[find_files_in_group\(\)](#) (in module [d1_util.create_data_packages](#)), 13
[find_object_size_stats_node\(\)](#) (in module [d1_util.generate_object_stats](#)), 18
[find_object_size_stats_node_all\(\)](#) (in module [d1_util.generate_object_stats](#)), 18
[find_url_mismatches\(\)](#) (in module [d1_common.url](#)), 238
[find_valid_combinations\(\)](#) (in module [d1_util.jwt_token_tasks](#)), 18
[fitness_and_quality_parsed\(\)](#) (in module [d1_common.ext.mimeparser](#)), 179
[fixed_offset\(\)](#) (class in [d1_common.date_time](#)), 208
[force_refresh\(\)](#) ([d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient](#) [method](#)), 49
[format\(\)](#) ([d1_cli.impl.format_ids.FormatIDs](#) [method](#)), 81
[format\(\)](#) ([d1_cli.impl.nodes.Nodes](#) [method](#)), 81
[format_checksum\(\)](#) (in module [d1_common.checksum](#)), 207
[format_diff_pyxb\(\)](#) (in module [d1_common.xml](#)), 244
[format_diff_xml\(\)](#) (in module [d1_common.xml](#)), 244
[format_json_to_normalized_pretty_json\(\)](#) (in module [d1_common.util](#)), 240
[format_sec_to_dhm\(\)](#) (in module [d1_common.util](#)), 241
[FormatIDs](#) (class in [d1_cli.impl.format_ids](#)), 81

friendly_format() (in module *d1_common.types.exceptions.DataONEException*), 187

FUSE, 23

fusepy, 23

G

gen_abs_uri() (in module *d1_scimeta.util*), 290

gen_abs_xsd_path_list() (in module *d1_scimeta.util*), 289

gen_access_policy() (in module *d1_common.system_metadata*), 231

gen_checksum_and_size() (in module *d1_common.system_metadata*), 231

gen_media_type() (in module *d1_common.system_metadata*), 231

gen_rel_xsd_path() (in module *d1_scimeta.util*), 289

gen_replication_policy() (in module *d1_common.system_metadata*), 231

gen_safe_path() (in module *d1_common.utils.filesystem*), 190

gen_safe_path_element() (in module *d1_common.utils.filesystem*), 191

gen_subject_info_tree() (in module *d1_common.cert.subject_info*), 170

gen_sysmeta() (in module *d1_test.utilities.create_from_file*), 311

gen_target_ns_to_rel_xsd_path_tuple() (in module *d1_scimeta.gen_root_xsd*), 286

gen_xsd_name_dict() (in module *d1_scimeta.util*), 289

gen_xsd_tree() (in module *d1_scimeta.gen_root_xsd*), 286

Generate (class in *d1_test.instance_generator.format_id*), 301

Generate (class in *d1_test.instance_generator.user_agent*), 305

generate() (in module *d1_test.instance_generator.checksum*), 301

generate() (in module *d1_test.instance_generator.identifier*), 302

generate() (in module *d1_test.instance_generator.media_type*), 302

generate() (in module *d1_test.instance_generator.node_ref*), 302

generate() (in module *d1_test.instance_generator.person*), 302

generate() (in module *d1_test.instance_generator.replication_policy*), 305

generate() (in module *d1_test.instance_generator.replication_status*), 305

generate() (in module *d1_test.instance_generator.subject*), 305

generate_bare() (in module *d1_test.instance_generator.identifier*), 302

generate_bare() (in module *d1_test.instance_generator.node_ref*), 302

generate_bare() (in module *d1_test.instance_generator.subject*), 305

generate_ca_cert() (in module *d1_common.cert.x509*), 177

generate_cert() (in module *d1_common.cert.x509*), 177

generate_csr() (in module *d1_common.cert.x509*), 174

generate_pid() (in module *d1_test.instance_generator.identifier*), 302

generate_private_key() (in module *d1_common.cert.x509*), 176

generate_public_access_policy() (in module *d1_util.create_data_packages*), 13

generate_random_ascii() (in module *d1_test.replication_tester.test_object_generator*), 310

generate_random_ascii() (in module *d1_test.utilities.test_object_generator*), 313

generate_science_object_with_sysmeta() (in module *d1_test.replication_tester.test_object_generator*), 310

generate_science_object_with_sysmeta() (in module *d1_test.utilities.test_object_generator*), 313

generate_sid() (in module *d1_test.instance_generator.identifier*), 302

generate_sys_meta() (in module *d1_util.create_data_packages*), 13

generate_system_metadata_for_science_object() (in module *d1_util.create_data_packages*), 13

generate_system_metadata_pyxb() (in module *d1_common.system_metadata*), 229

generateIdentifier() (*d1_client.baseclient.DataONEBaseClient* method), 258

generateIdentifierResponse() (*d1_client.baseclient.DataONEBaseClient* method), 258

get() (*d1_cli.impl.format_ids.FormatIDs* method), 81

get() (*d1_cli.impl.nodes.Nodes* method), 81

get() (*d1_cli.impl.session.Session* method), 83

get() (*d1_client.aio.async_client.AsyncDataONEClient* method), 248

get() (*d1_client.baseclient.DataONEBaseClient* method), 257

GET() (*d1_client.session.Session* method), 280

`get()` (*d1_client.solr_client.SolrClient* method), 282
`get()` (*d1_test.slender_node_test_client.SlenderNodeTestClient* method), 314
`get_abs_root_xsd_path()` (in module *d1_scimeta.util*), 288
`get_abs_schema_branch_path()` (in module *d1_scimeta.util*), 289
`get_abs_sciobj_file_path_by_pid()` (in module *d1_gmn.app.sciobj_store*), 155
`get_abs_sciobj_file_path_by_rel_path()` (in module *d1_gmn.app.sciobj_store*), 155
`get_abs_sciobj_file_path_by_url()` (in module *d1_gmn.app.sciobj_store*), 155
`get_abs_sciobj_file_url()` (in module *d1_gmn.app.sciobj_store*), 155
`get_abs_sciobj_store_path()` (in module *d1_gmn.app.sciobj_store*), 155
`get_abs_test_file_path()` (in module *d1_test.test_files*), 315
`get_access_control()` (*d1_cli.impl.session.Session* method), 83
`get_access_policy()` (*d1_gmn.tests.gmn_test_client.GMNTestClient* method), 157
`get_all_pid_by_sid()` (in module *d1_gmn.app.revision*), 153
`get_and_save()` (*d1_client.baseclient.DataONEBaseClient* method), 258
`get_api_major_by_base_url()` (in module *d1_client.d1client*), 277
`get_attr_value()` (*d1_common.wrap.simple_xml.SimpleXMLWrapperClient* method), 203
`get_attributes()` (*d1_onedrive.impl.resolver.author.Resolver* method), 50
`get_attributes()` (*d1_onedrive.impl.resolver.d1_object.Resolver* method), 50
`get_attributes()` (*d1_onedrive.impl.resolver.flat_space.Resolver* method), 51
`get_attributes()` (*d1_onedrive.impl.resolver.object_tree_resolver.Resolver* method), 51
`get_attributes()` (*d1_onedrive.impl.resolver.region.Resolver* method), 51
`get_attributes()` (*d1_onedrive.impl.resolver.resource_map.Resolver* method), 52
`get_attributes()` (*d1_onedrive.impl.resolver.root.RootResolver* method), 52
`get_attributes()` (*d1_onedrive.impl.resolver.single.Resolver* method), 52
`get_attributes()` (*d1_onedrive.impl.resolver.taxa.Resolver* method), 52
`get_attributes()` (*d1_onedrive.impl.resolver.time_period.Resolver* method), 53
`get_authenticated_subjects()` (in module *d1_gmn.app.middleware.session_cert*), 143
`get_auto()` (in module *d1_common.xml*), 244
`get_blocked()` (*d1_cli.impl.replication_policy.ReplicationPolicy* method), 83
`get_bu64_tup()` (in module *d1_common.cert.jwt*), 165
`get_capabilities()` (*d1_client.aio.async_client.AsyncDataONEClient* method), 248
`get_cert_info_list()` (in module *d1_common.cert.x509*), 178
`get_checksum_calculator_by_dataone_designator()` (in module *d1_common.checksum*), 207
`get_client_class_by_version_tag()` (in module *d1_client.d1client*), 277
`get_client_type()` (in module *d1_client.d1client*), 277
`get_cn_metrics()` (in module *d1_util.display_node_status*), 14
`get_cn_subjects()` (in module *d1_gmn.app.node_registry*), 151
`get_cn_subjects_from_dataone_root()` (in module *d1_gmn.app.node_registry*), 152
`get_cn_subjects_string()` (in module *d1_gmn.app.node_registry*), 152
`get_content_type()` (*d1_common.object_format_cache.ObjectFormatListCache* method), 217
`get_content_type()` (in module *d1_common.util*), 238
`get_curl_command_line()` (*d1_client.session.Session* method), 281
`get_d1_env()` (in module *d1_common.env*), 214
`get_d1_env_by_base_url()` (in module *d1_common.env*), 214
`get_d1_env_keys()` (in module *d1_common.env*), 214
`get_default_checksum_algorithm()` (in module *d1_common.checksum*), 207
`get_pickle_file_path()` (*d1_cli.impl.session.Session* method), 84
`get_pid_by_foreign_key()` (in module *d1_gmn.app.did*), 150
`get_resolver()` (*d1_onedrive.impl.resolver.author.Resolver* method), 50
`get_resolver()` (*d1_onedrive.impl.resolver.d1_object.Resolver* method), 50
`get_resolver()` (*d1_onedrive.impl.resolver.flat_space.Resolver* method), 51
`get_resolver()` (*d1_onedrive.impl.resolver.object_tree_resolver.Resolver* method), 51
`get_resolver()` (*d1_onedrive.impl.resolver.region.Resolver* method), 51
`get_resolver()` (*d1_onedrive.impl.resolver.resource_map.Resolver* method), 52

[get_directory\(\) \(d1_onedrive.impl.resolver.root.RootResolver method\), 52](#)
[get_directory\(\) \(d1_onedrive.impl.resolver.single.Resolver method\), 52](#)
[get_directory\(\) \(d1_onedrive.impl.resolver.taxa.Resolver method\), 52](#)
[get_directory\(\) \(d1_onedrive.impl.resolver.time_period.Resolver method\), 53](#)
[get_effective_perm_list\(\) \(d1_common.wrap.access_policy.AccessPolicyWrapper method\), 197](#)
[get_effective_perm_list\(\) \(in module d1_common.wrap.access_policy\), 199](#)
[get_element_by_attr_key\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 202](#)
[get_element_by_name\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 202](#)
[get_element_by_xpath\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 201](#)
[get_element_dt\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 203](#)
[get_element_list_by_attr_key\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 201](#)
[get_element_list_by_name\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 201](#)
[get_element_text\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 202](#)
[get_element_text_by_attr_key\(\) \(d1_common.wrap.simple_xml.SimpleXMLWrapper method\), 202](#)
[get_error_log_as_str\(\) \(d1_scimeta.util\), 290](#)
[get_extension_by_name\(\) \(d1_common.cert.x509\), 178](#)
[get_field_min_max\(\) \(d1_client.solr_client.SolrClient method\), 283](#)
[get_field_values\(\) \(d1_client.solr_client.SolrClient method\), 283](#)
[get_filename_extension\(\) \(d1_common.object_format_cache.ObjectFormatListCache method\), 217](#)
[get_filtered_sub_tree\(\) \(d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient method\), 49](#)
[get_folder\(\) \(d1_onedrive.impl.object_tree.ObjectTree method\), 55](#)
[get_format_ids\(\) \(d1_cli.impl.command_processor.CommandProcessor method\), 80](#)
[get_gen_metrics\(\) \(in module d1_util.display_node_status\), 14](#)
[get_gmn_version\(\) \(in module d1_gmn.app.sciobj_store\), 156](#)
[get_highest_perm_str\(\) \(d1_common.wrap.access_policy.AccessPolicyWrapper method\), 197](#)
[get_highest_perm_str\(\) \(in module d1_common.wrap.access_policy\), 199](#)
[get_identifiers\(\) \(in module d1_common.revision\), 226](#)
[get_ids\(\) \(d1_client.solr_client.SolrClient method\), 282](#)
[get_jwt_bu64\(\) \(in module d1_common.cert.jwt\), 165](#)
[get_jwt_dict\(\) \(in module d1_common.cert.jwt\), 165](#)
[get_jwt_tup\(\) \(in module d1_common.cert.jwt\), 165](#)
[get_label_set\(\) \(d1_common.cert.subject_info.SubjectInfoNode method\), 172](#)
[get_node_path_list\(\) \(d1_common.cert.subject_info.SubjectInfoNode method\), 171](#)
[get_list\(\) \(d1_cli.impl.access_control.AccessControl method\), 76](#)
[get_log_records\(\) \(d1_client.aio.async_client.AsyncDataONEClient method\), 248](#)
[get_mn_metrics\(\) \(in module d1_util.display_node_status\), 14](#)
[get_node_id\(\) \(d1_client.d1client.DataONEClient method\), 277](#)
[get_node_list_from_coordinating_node\(\) \(in module d1_util.display_node_status\), 14](#)
[get_node_list_from_coordinating_node\(\) \(in module d1_util.download_all_objects\), 15](#)
[get_nodes\(\) \(d1_cli.impl.command_processor.CommandProcessor method\), 80](#)
[get_normalized_perm_list\(\) \(d1_common.wrap.access_policy.AccessPolicyWrapper method\), 197](#)
[get_normalized_perm_list\(\) \(in module d1_common.wrap.access_policy\), 199](#)
[get_normalized_pyxb\(\) \(d1_common.wrap.access_policy.AccessPolicyWrapper method\), 197](#)
[get_normalized_pyxb\(\) \(in module d1_common.wrap.access_policy\), 199](#)
[get_normalized_xml_representation\(\) \(in module d1_test.xml_normalize\), 316](#)
[get_number_of_log_records\(\) \(in module d1_util.display_node_status\), 14](#)

<code>get_number_of_objects()</code> (in module <code>d1_util.display_node_status</code>), 14	<code>(d1_cli.impl.replication_policy.ReplicationPolicy method)</code> , 83
<code>get_number_of_replicas()</code> (in module <code>d1_cli.impl.replication_policy.ReplicationPolicy method</code>), 83	<code>get_replication_policy()</code> (<code>d1_cli.impl.session.Session method</code>), 83
<code>get_object_record()</code> (<code>d1_onedrive.impl.object_tree.ObjectTree method</code>), 55	<code>get_replication_queue()</code> (<code>d1_gmn.tests.gmn_test_client.GMNTestClient method</code>), 157
<code>get_object_record_with_sync()</code> (<code>d1_onedrive.impl.object_tree.ObjectTree method</code>), 55	<code>get_req_val()</code> (in module <code>d1_common.xml</code>), 245
<code>get_object_tree_folder_name()</code> (<code>d1_onedrive.impl.object_tree.ObjectTree method</code>), 55	<code>get_resource_map_from_sciobj()</code> (in module <code>d1_gmn.app.resource_map</code>), 152
<code>get_operation_queue()</code> (<code>d1_cli.impl.command_processor.CommandProcessor method</code>), 80	<code>get_resource_map_members()</code> (in module <code>d1_gmn.app.resource_map</code>), 152
<code>get_opt_attr()</code> (in module <code>d1_common.xml</code>), 244	<code>get_resource_map_members_by_map()</code> (in module <code>d1_gmn.app.resource_map</code>), 152
<code>get_opt_val()</code> (in module <code>d1_common.xml</code>), 245	<code>get_resource_map_members_by_member()</code> (in module <code>d1_gmn.app.resource_map</code>), 152
<code>get_or_create_did()</code> (in module <code>d1_gmn.app.did</code>), 150	<code>get_resource_path()</code> (<code>d1_gmn.tests.gmn_test_client.GMNTestClient method</code>), 157
<code>get_path_list()</code> (<code>d1_common.cert.subject_info.SubjectInfoNode method</code>), 172	<code>get_root_ns()</code> (in module <code>d1_scimeta.util</code>), 287
<code>get_path_str()</code> (<code>d1_common.cert.subject_info.SubjectInfoNode method</code>), 171	<code>get_schema_name()</code> (in module <code>d1_scimeta.util</code>), 287
<code>get_pids_for_all_locally_stored_objects()</code> (in module <code>d1_gmn.app.model_util</code>), 151	<code>get_sci_model()</code> (in module <code>d1_gmn.app.model_util</code>), 151
<code>get_pids_in_revision_chain()</code> (in module <code>d1_common.revision</code>), 226	<code>get_science_object()</code> (<code>d1_onedrive.impl.clients.onedrive_d1_client.DataONEClient method</code>), 48
<code>get_ping()</code> (in module <code>d1_util.display_node_status</code>), 14	<code>get_science_object()</code> (<code>d1_onedrive.impl.object_tree.ObjectTree method</code>), 55
<code>get_preferred()</code> (<code>d1_cli.impl.replication_policy.ReplicationPolicy method</code>), 83	<code>get_science_object_through_cache()</code> (<code>d1_onedrive.impl.tests.command_echoer.CommandEchoer method</code>), 53
<code>get_pretty_xml()</code> (<code>d1_common.wrap.simple_xml.SimpleXMLWrapper method</code>), 201	<code>get_sciobj_format_id_list()</code> (in module <code>d1_util.parse_format_id_list</code>), 19
<code>get_pretty_xml()</code> (in module <code>d1_gmn.app.node</code>), 151	<code>get_sciobj_byte_iterator_by_url()</code> (in module <code>d1_gmn.app.sciobj_store</code>), 154
<code>get_public_key_pem()</code> (in module <code>d1_common.cert.x509</code>), 176	<code>get_sciobj_iter_by_pid()</code> (in module <code>d1_gmn.app.sciobj_store</code>), 154
<code>get_pyxb()</code> (<code>d1_common.types.exceptions.DataONEException method</code>), 188	<code>get_sciobj_iter_by_url()</code> (in module <code>d1_gmn.app.sciobj_store</code>), 154
<code>get_pyxb()</code> (in module <code>d1_gmn.app.node</code>), 151	<code>get_sciobj_iter_remote()</code> (in module <code>d1_gmn.app.proxy</code>), 152
<code>get_pyxb_binding_by_api_version()</code> (in module <code>d1_common.type_conversions</code>), 231	<code>get_session()</code> (<code>d1_cli.impl.command_processor.CommandProcessor method</code>), 80
<code>get_pyxb_namespaces()</code> (in module <code>d1_common.type_conversions</code>), 232	<code>get_setting()</code> (<code>d1_gmn.tests.gmn_test_client.GMNTestClient method</code>), 158
<code>get_random_pid_list()</code> (in module <code>d1_util.download_test_docs</code>), 17	<code>get_sid_by_pid()</code> (in module <code>d1_gmn.app.revision</code>), 153
<code>get_rel_path()</code> (in module <code>d1_scimeta.util</code>), 289	<code>get_solr_record()</code> (<code>d1_onedrive.impl.clients.onedrive_solr_client.OneDriveSolrClient method</code>), 49
<code>get_rel_sciobj_file_path()</code> (in module <code>d1_gmn.app.sciobj_store</code>), 155	<code>get_sort_key()</code> (<code>d1_test.xml_normalize.StableNode method</code>), 316
<code>get_rel_sciobj_file_url_by_pid()</code> (in module <code>d1_gmn.app.sciobj_store</code>), 155	
<code>get_replication_allowed()</code>	

`get_source_tree_folder()` (*d1_onedrive.impl.object_tree.ObjectTree* method), 56
`get_standard_arg_parser()` (in module *d1_client.command_line*), 275
`get_static_path()` (in module *d1_gmn.app.util*), 156
`get_store_version()` (in module *d1_gmn.app.sciobj_store*), 156
`get_store_version_path()` (in module *d1_gmn.app.sciobj_store*), 156
`get_str()` (*d1_test.xml_normalize.StableNode* method), 316
`get_subject_set()` (*d1_common.cert.subject_info.SubjectInfoNode* method), 172
`get_subject_with_file_validation()` (in module *d1_common.cert.jwt*), 164
`get_subject_with_local_validation()` (in module *d1_common.cert.jwt*), 164
`get_subject_with_remote_validation()` (in module *d1_common.cert.jwt*), 164
`get_subject_without_validation()` (in module *d1_common.cert.jwt*), 164
`get_subjects()` (in module *d1_gmn.app.middleware.session_cert*), 143
`get_subjects_with_equal_or_higher_perm()` (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 197
`get_subjects_with_equal_or_higher_perm()` (in module *d1_common.wrap.access_policy*), 199
`get_supported_algorithms()` (in module *d1_common.checksum*), 207
`get_supported_format_id_list()` (in module *d1_scimeta.util*), 288
`get_system_metadata()` (*d1_client.aio.async_client.AsyncDataONEClient* method), 248
`get_system_metadata()` (*d1_onedrive.impl.clients.onedrive_d1_client.DataONEClient* method), 48
`get_system_metadata()` (*d1_onedrive.impl.object_tree.ObjectTree* method), 56
`get_system_metadata_as_string()` (*d1_onedrive.impl.clients.onedrive_d1_client.DataONEClient* method), 48
`get_system_metadata_through_cache()` (*d1_onedrive.impl.tests.command_echoer.CommandEchoer* method), 53
`get_target_ns()` (in module *d1_scimeta.util*), 288
`get_test_xml_path_list()` (in module *d1_scimeta.tests.validate_test_docs*), 286
`get_version_tag()` (in module *d1_common.type_conversions*), 232
`get_version_tag_by_d1_client()` (in module *d1_client.d1client*), 277
`get_version_tag_by_pyxb_binding()` (in module *d1_common.type_conversions*), 231
`get_xml()` (*d1_common.wrap.simple_xml.SimpleXMLWrapper* method), 201
`get_xml()` (in module *d1_gmn.app.node*), 151
`get_xml_below_element()` (*d1_common.wrap.simple_xml.SimpleXMLWrapper* method), 201
`get_xs_include_xs_import_schema_location_tup()` (in module *d1_scimeta.util*), 287
`get_xsd_path()` (in module *d1_scimeta.util*), 290
`get_xsi_schema_location_tup()` (in module *d1_scimeta.util*), 287
`getAggregatedPids()` (*d1_common.resource_map.ResourceMap* method), 225
`getAggregatedScienceDataPids()` (*d1_common.resource_map.ResourceMap* method), 225
`getAggregatedScienceMetadataPids()` (*d1_common.resource_map.ResourceMap* method), 225
`getAggregation()` (*d1_common.resource_map.ResourceMap* method), 223
`getAllPredicates()` (*d1_common.resource_map.ResourceMap* method), 224
`getAllTriples()` (*d1_common.resource_map.ResourceMap* method), 224
`getCapabilities()` (*d1_client.mnclient.MemberNodeClient* method), 277
`getCapabilitiesResponse()` (*d1_client.mnclient.MemberNodeClient* method), 277
`getChecksum()` (*d1_client.cnclient.CoordinatingNodeClient* method), 264
`getChecksum()` (*d1_client.mnclient.MemberNodeClient* method), 277
`getChecksumResponse()` (*d1_client.cnclient.CoordinatingNodeClient* method), 263
`getChecksumResponse()` (*d1_client.mnclient.MemberNodeClient* method), 277
`getFormat()` (*d1_client.cnclient.CoordinatingNodeClient* method), 261
`getFormatResponse()` (*d1_client.cnclient.CoordinatingNodeClient* method), 261

[getLogRecords\(\)](#) (*d1_client.baseclient.DataONEBaseClient* [method](#)), 257
[getLogRecordsResponse\(\)](#) (*d1_client.baseclient.DataONEBaseClient* [method](#)), 258
[global_settings\(\)](#) (in module *d1_gmn.app.context_processors*), 149
[getObjectByPid\(\)](#) (*d1_common.resource_map.ResourceMap* [method](#)), 223
[getPackage\(\)](#) (*d1_client.mnclient_1_2.MemberNodeClient_1_2* [method](#)), 278
[getPackageResponse\(\)](#) (*d1_client.mnclient_1_2.MemberNodeClient_1_2* [method](#)), 278
[getQueryEngineDescription\(\)](#) (*d1_client.baseclient_1_1.DataONEBaseClient_1_1* [method](#)), 259
[getQueryEngineDescription\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 265
[getQueryEngineDescriptionResponse\(\)](#) (*d1_client.baseclient_1_1.DataONEBaseClient_1_1* [method](#)), 259
[getQueryEngineDescriptionResponse\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 265
[getReplica\(\)](#) (*d1_client.mnclient.MemberNodeClient* [method](#)), 278
[getReplicaResponse\(\)](#) (*d1_client.mnclient.MemberNodeClient* [method](#)), 278
[getResourceMapPid\(\)](#) (*d1_common.resource_map.ResourceMap* [method](#)), 224
[getResponse\(\)](#) (*d1_client.baseclient.DataONEBaseClient* [method](#)), 257
[getSubjectFromCertFile\(\)](#) (in module *d1_test.utilities.my_subject*), 313
[getSubjectFromName\(\)](#) (in module *d1_test.utilities.my_subject*), 312
[getSubjectInfo\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 267
[getSubjectInfoFromCert\(\)](#) (in module *d1_test.utilities.my_subject*), 313
[getSubjectInfoResponse\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 267
[getSubjectObjectsByPredicate\(\)](#) (*d1_common.resource_map.ResourceMap* [method](#)), 224
[getSystemMetadata\(\)](#) (*d1_client.baseclient.DataONEBaseClient* [method](#)), 258
[getSystemMetadata\(\)](#) (*d1_test.slender_node_test_client.SlenderNodeTestClient* [method](#)), 314
[getSystemMetadataResponse\(\)](#) (*d1_client.baseclient.DataONEBaseClient* [method](#)), 258
[gm_client_v1\(\)](#) (in module *d1_gmn.tests.conftest*), 157
[gm_client_v1_v2\(\)](#) (in module *d1_gmn.tests.conftest*), 157
[gm_client_v2\(\)](#) (in module *d1_gmn.tests.conftest*), 157
[gm_vse_enable_python_profiling\(\)](#) (*d1_gmn.tests.gmn_test_client.GMNTTestClient* [method](#)), 157
[gm_vse_enable_sql_profiling\(\)](#) (*d1_gmn.tests.gmn_test_client.GMNTTestClient* [method](#)), 157
[gm_vse_provide_subject\(\)](#) (*d1_gmn.tests.gmn_test_client.GMNTTestClient* [method](#)), 157
[GMNTTestClient](#) (class in *d1_gmn.tests.gmn_test_client*), 157
[group_name\(\)](#) (in module *d1_util.create_data_packages*), 13
H
[handle\(\)](#) (*d1_gmn.app.management.commands.event.Command* [method](#)), 141
[handle_options\(\)](#) (in module *d1_cli.dataone*), 85
[handle_unexpected_exception\(\)](#) (in module *d1_cli.dataone*), 85
[Handler](#) (class in *d1_test.replication_tester.replication_server*), 310
[has_matching_modified_timestamp\(\)](#) (in module *d1_gmn.app.views.assert_sysmeta*), 148
[has_replication_policy\(\)](#) (in module *d1_common.replication_policy*), 217
[has_tz\(\)](#) (in module *d1_common.date_time*), 209
[hasReservation\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 263
[hasReservationResponse\(\)](#) (*d1_client.cnclient.CoordinatingNodeClient* [method](#)), 263
[HEAD\(\)](#) (*d1_client.session.Session* [method](#)), 280
[http_datetime_str_from_dt\(\)](#) (in module *d1_common.date_time*), 210
I
[identifier_from_filename\(\)](#) (in module *d1_onedrive.impl.os_escape*), 56
[IdentifierNotUnique](#), 188
[Identity Provider](#), 22

initial (*d1_gmn.app.migrations.0001_initial.Migration* is_matching_version() (in module attribute), 144 *d1_gmn.app.sciobj_store*), 156
 initialize() (*d1_common.resource_map.ResourceMap* is_member_node() (in module method), 222 *d1_util.display_node_status*), 14
 inject_fictional_event_log() is_member_node() (in module (*d1_gmn.tests.gmn_test_client.GMNTTestClient* is_multipart() (in module *d1_common.multipart*), 215 method), 158 *d1_util.download_all_objects*), 15
 input_key_passphrase() (in module is_not_archived() (in module *d1_common.cert.x509*), 177 *d1_gmn.app.views.assert_db*), 147
 insert() (*d1_onedrive.impl.directory.Directory* is_not_obsoleted() (in module method), 54 *d1_gmn.app.views.assert_db*), 147
 InsufficientResources, 188 is_not_replica() (in module *d1_gmn.app.views.assert_db*), 147
 InvalidArguments, 81 is_obsoleted() (in module *d1_gmn.app.did*), 150
 InvalidCredentials, 188 is_obsoleted_by_pid() (in module *d1_gmn.app.revision*), 153
 InvalidRequest, 189 is_obsoletes_pid() (in module *d1_gmn.app.revision*), 153
 InvalidSystemMetadata, 189 is_preferred() (in module *d1_common.replication_policy*), 218
 InvalidToken, 189 is_private() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198
 Investigator Toolkit (ITK), 21 is_private() (in module *d1_common.wrap.access_policy*), 199
 is_archived() (in module *d1_gmn.app.did*), 150 is_proxy_url() (in module *d1_gmn.app.proxy*), 152
 is_blocked() (in module is_public() (*d1_common.wrap.access_policy.AccessPolicyWrapper* method), 198 *d1_common.replication_policy*), 218
 is_bool_param() (in module is_public() (in module *d1_common.wrap.access_policy*), 199 *d1_gmn.app.views.assert_db*), 147
 is_coordinating_node() (in module is_public() (in module *d1_common.wrap.access_policy*), 199 *d1_util.display_node_status*), 14
 is_did() (in module *d1_gmn.app.views.assert_db*), 147 is_pyxb() (in module *d1_common.type_conversions*), 233
 is_dir() (*d1_onedrive.impl.attributes.Attributes* is_pyxb_d1_type() (in module method), 54 *d1_common.type_conversions*), 233
 is_empty() (*d1_common.wrap.access_policy.AccessPolicyWrapper* is_pyxb_d1_type_name() (in module method), 198 *d1_common.wrap.access_policy*), 199
 is_empty() (in module is_resource_map_db() (in module *d1_common.wrap.access_policy*), 199 *d1_gmn.app.did*), 150
 is_empty() (in module *d1_gmn.app.sciobj_store*), 156 is_resource_map_member() (in module *d1_gmn.app.sciobj_store*), 155
 is_existing_object() (in module is_resource_map_sysmeta_pyxb() (in module *d1_gmn.app.did*), 150 *d1_gmn.app.views.assert_db*), 147
 is_existing_object() (in module is_revision() (in module *d1_gmn.app.views.assert_db*), 147 *d1_gmn.app.did*), 150
 is_existing_sciobj_file() (in module is_revision_chain_placeholder() (in module *d1_gmn.app.sciobj_store*), 155 *d1_gmn.app.did*), 150
 is_existing_store() (in module is_root() (*d1_onedrive.impl.resolver.resolver_base.Resolver* method), 51 *d1_gmn.app.sciobj_store*), 155
 is_in_revision_chain() (in module is_sciobj_valid_for_create() (in module *d1_gmn.app.did*), 150 *d1_gmn.app.views.assert_db*), 147
 is_in_revision_chain() (in module is_sid() (in module *d1_gmn.app.views.assert_db*), 147 *d1_gmn.app.views.assert_db*), 147
 is_installed_scimeta_format_id() (in module is_sid() (in module *d1_gmn.app.views.assert_db*), 147 *d1_scimeta.util*), 289
 is_leaf() (*d1_common.cert.subject_info.SubjectInfoNodes* is_root() (in module *d1_onedrive.impl.util*), 57 property), 172
 is_local_replica() (in module *d1_gmn.app.did*), 150 is_sciobj_valid_for_create() (in module *d1_gmn.app.resource_map*), 152
 is_lower_version() (in module is_sid() (in module *d1_gmn.app.views.assert_db*), 147 *d1_gmn.app.sciobj_store*), 156

`is_store_subdir()` (in module `d1_gmn.app.sciobj_store`), 156
`is_supported_algorithm()` (in module `d1_common.checksum`), 207
`is_sysmeta_pyxb()` (in module `d1_common.system_metadata`), 228
`is_tmp()` (in module `d1_gmn.app.sciobj_store`), 156
`is_unprocessed_local_replica()` (in module `d1_gmn.app.did`), 150
`is_url()` (in module `d1_scimeta.util`), 291
`is_urls_equivalent()` (in module `d1_common.url`), 238
`is_utc()` (in module `d1_common.date_time`), 209
`is_valid_format_id()` (`d1_common.object_format_cache.ObjectFormatListCache` method), 217
`is_valid_iso8601()` (in module `d1_common.date_time`), 209
`is_valid_pid_for_create()` (in module `d1_gmn.app.did`), 149
`is_valid_pid_for_create()` (in module `d1_gmn.app.views.assert_db`), 147
`is_valid_pid_to_be_updated()` (in module `d1_gmn.app.did`), 149
`is_valid_pid_to_be_updated()` (in module `d1_gmn.app.views.assert_db`), 147
`is_valid_sid_for_chain()` (in module `d1_gmn.app.did`), 149
`is_valid_sid_for_chain()` (in module `d1_gmn.app.views.assert_sysmeta`), 148
`is_valid_sid_for_new_standalone()` (in module `d1_gmn.app.did`), 149
`is_valid_sid_for_new_standalone()` (in module `d1_gmn.app.views.assert_sysmeta`), 148
`is_valid_utf8()` (in module `d1_common.xml`), 244
`is_valid_xml_file()` (in module `d1_scimeta.util`), 291
`is_valid_xsd_file()` (in module `d1_scimeta.util`), 291
`is_verbose()` (`d1_cli.impl.session.Session` method), 83
`isAuthorized()` (`d1_client.baseclient.DataONEBaseClient` method), 258
`isAuthorizedResponse()` (`d1_client.baseclient.DataONEBaseClient` method), 258
`isHttpOrHttps()` (in module `d1_common.url`), 237
`isNodeAuthorized()` (`d1_client.cnclient.CoordinatingNodeClient` method), 272
`isNodeAuthorizedResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 272
`ISO8601`, 24
`iterate_collection_tree()` (`d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient` method), 49
`iterate_collection_trees()` (`d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient` method), 49
`iterate_filtered_tree()` (`d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient` method), 49
`itr()` (`d1_gmn.app.management.commands.logininfo_async.EventLogItera` method), 141
`itr()` (`d1_gmn.app.management.commands.objectlist_async.ObjectListIt` method), 142
J
`joinPathElements()` (in module `d1_common.url`), 237
`jwt_cleanup()` (in module `d1_util.jwt_token_tasks`), 18
`JwtException`, 167
K
`keys()` (`d1_onedrive.impl.cache_memory.Cache` method), 54
L
`leaf_node_gen()` (`d1_common.cert.subject_info.SubjectInfoNode` property), 171
`list_format_ids()` (`d1_cli.impl.command_processor.CommandProcessor` method), 80
`list_nodes()` (`d1_cli.impl.command_processor.CommandProcessor` method), 80
`list_nodes()` (`d1_client.aio.async_client.AsyncDataONEClient` method), 248
`list_objects()` (`d1_cli.impl.command_processor.CommandProcessor` method), 80
`list_objects()` (`d1_client.aio.async_client.AsyncDataONEClient` method), 248
`listChecksumAlgorithms()` (`d1_client.cnclient.CoordinatingNodeClient` method), 262
`listChecksumAlgorithmsResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 262
`listFormats()` (`d1_client.cnclient.CoordinatingNodeClient` method), 261
`listFormatsResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 261
`listNodes()` (`d1_client.cnclient.CoordinatingNodeClient` method), 262

[listNodesResponse\(\)](#) ([d1_client.cnclient.CoordinatingNodeClient](#) [method](#)), 239
[listObjects\(\)](#) ([d1_client.baseclient.DataONEBaseClient](#) [method](#)), 258
[listObjectsResponse\(\)](#) ([d1_client.baseclient.DataONEBaseClient](#) [method](#)), 258
[listSubjects\(\)](#) ([d1_client.cnclient.CoordinatingNodeClient](#) [method](#)), 268
[listSubjectsResponse\(\)](#) ([d1_client.cnclient.CoordinatingNodeClient](#) [method](#)), 267
[listViews\(\)](#) ([d1_client.cnclient_2_0.CoordinatingNodeClient_2_0](#) [method](#)), 275
[listViews\(\)](#) ([d1_client.mnclient_1_2.MemberNodeClient_1_2](#) [method](#)), 278
[listViewsResponse\(\)](#) ([d1_client.cnclient_2_0.CoordinatingNodeClient_2_0](#) [method](#)), 275
[listViewsResponse\(\)](#) ([d1_client.mnclient_1_2.MemberNodeClient_1_2](#) [method](#)), 278
[LOA](#), 24
[load\(\)](#) ([d1_cli.impl.session.Session](#) [method](#)), 83
[load_bin\(\)](#) (in module [d1_test.test_files](#)), 315
[load_bytes_from_file\(\)](#) (in module [d1_scimeta.util](#)), 290
[load_cert\(\)](#) (in module [d1_test.test_files](#)), 315
[load_csr\(\)](#) (in module [d1_common.cert.x509](#)), 177
[load_json\(\)](#) (in module [d1_common.util](#)), 240
[load_json\(\)](#) (in module [d1_test.test_files](#)), 315
[load_jwt\(\)](#) (in module [d1_test.test_files](#)), 315
[load_private_key\(\)](#) (in module [d1_common.cert.x509](#)), 177
[load_schema\(\)](#) (in module [d1_scimeta.schema_resolve](#)), 287
[load_utf8_to_str\(\)](#) (in module [d1_test.test_files](#)), 315
[load_xml_file_to_tree\(\)](#) (in module [d1_scimeta.util](#)), 290
[load_xml_to_bytes\(\)](#) (in module [d1_test.test_files](#)), 315
[load_xml_to_pyxb\(\)](#) (in module [d1_test.test_files](#)), 315
[load_xml_to_str\(\)](#) (in module [d1_test.test_files](#)), 315
[local_now\(\)](#) (in module [d1_common.date_time](#)), 213
[local_now_iso\(\)](#) (in module [d1_common.date_time](#)), 213
[log](#) (in module [d1_scimeta.tests.validate_test_docs](#)), 286
[log\(\)](#) ([d1_cli.impl.command_processor.CommandProcessor](#) [method](#)), 80
[log_and_count\(\)](#) ([d1_common.util.EventCounter](#) [method](#)), 239
[log_cert_info\(\)](#) (in module [d1_common.cert.x509](#)), 177
[log_dict\(\)](#) (in module [d1_util.check_object_checksums](#)), 11
[log_dict\(\)](#) (in module [d1_util.generate_object_stats](#)), 18
[log_dump\(\)](#) ([d1_onedrive.impl.cache_memory.Cache](#) [method](#)), 54
[log_dump\(\)](#) (in module [d1_onedrive.impl.util](#)), 57
[log_func](#) (class in [d1_onedrive.impl.log_decorator](#)), 55
[log_jwt_bu64_info\(\)](#) (in module [d1_common.cert.jwt](#)), 166
[log_jwt_dict_info\(\)](#) (in module [d1_common.cert.jwt](#)), 166
[log_setup\(\)](#) (in module [d1_cli.dataone](#)), 85
[log_setup\(\)](#) (in module [d1_client.command_line](#)), 276
[log_setup\(\)](#) (in module [d1_common.util](#)), 238
[log_setup\(\)](#) (in module [d1_onedrive.onedrive](#)), 57
[log_startup_parameters\(\)](#) (in module [d1_onedrive.onedrive](#)), 57
[log_version\(\)](#) (in module [d1_onedrive.onedrive](#)), 57
[LoggingContext](#) (class in module [d1_common.logging_context](#)), 214
[LogRecordIterator](#) (class in module [d1_client.iter.logrecord](#)), 250
[LogRecordIteratorMulti](#) (class in module [d1_client.iter.logrecord_multi](#)), 250

M

[macfuse](#), 23
[main\(\)](#) (in module [d1_cli.dataone](#)), 85
[main\(\)](#) (in module [d1_onedrive.onedrive](#)), 57
[main\(\)](#) (in module [d1_scimeta.gen_root_xsd](#)), 286
[main\(\)](#) (in module [d1_scimeta.xml_validate](#)), 286
[main\(\)](#) (in module [d1_scimeta.schema_resolve](#)), 287
[main\(\)](#) (in module [d1_scimeta.tests.validate_test_docs](#)), 286
[main\(\)](#) (in module [d1_test.replication_tester.replication_tester](#)), 310
[main\(\)](#) (in module [d1_test.utilities.create_from_file](#)), 311
[main\(\)](#) (in module [d1_test.utilities.generate_test_subject_certs](#)), 312
[main\(\)](#) (in module [d1_test.utilities.populate_mn](#)), 313
[main\(\)](#) (in module [d1_util.cert_check_cn](#)), 10
[main\(\)](#) (in module [d1_util.cert_check_local](#)), 10
[main\(\)](#) (in module [d1_util.cert_create_ca](#)), 10
[main\(\)](#) (in module [d1_util.cert_create_csr](#)), 11
[main\(\)](#) (in module [d1_util.cert_sign_csr](#)), 11
[main\(\)](#) (in module [d1_util.check_object_checksums](#)), 11

`main()` (in module `d1_util.check_x509_certificate_cn`), `d1_util.download_mn_objects`), 16
 12 Metacat, 20
`main()` (in module `d1_util.check_x509_certificate_local`), `method_obj()` (in module
 12 `d1_common.wrap.access_policy`), 200
`main()` (in module `d1_util.create_data_packages`), 12 Migration (class in
`main()` (in module `d1_util.create_object_on_member_node`), `d1_gmn.app.migrations.0001_initial`), 144
 13 Migration (class in
`main()` (in module `d1_util.delete_all_objects_of_type`), `d1_gmn.app.migrations.0002_scienceobject_filename`),
 14 144
`main()` (in module `d1_util.display_node_status`), 14 Migration (class in
`main()` (in module `d1_util.download_all_objects`), 15 `d1_gmn.app.migrations.0003_mediatype_mediatypeproperty`),
`main()` (in module `d1_util.download_and_display_data_package`), 144
 16 Migration (class in
`main()` (in module `d1_util.download_mn_objects`), 16 `d1_gmn.app.migrations.0004_auto_20170523_0137`),
`main()` (in module `d1_util.download_sciobj`), 16 144
`main()` (in module `d1_util.download_server_certs`), 17 Migration (class in
`main()` (in module `d1_util.download_sysmeta`), 17 `d1_gmn.app.migrations.0005_auto_20170527_1554`),
`main()` (in module `d1_util.download_sysmeta_multiproc`), 144
 17 Migration (class in
`main()` (in module `d1_util.download_test_docs`), 17 `d1_gmn.app.migrations.0006_chainidtopersistentid`),
`main()` (in module `d1_util.generate_object_stats`), 18 145
`main()` (in module `d1_util.jwt_token_tasks`), 18 Migration (class in
`main()` (in module `d1_util.parse_format_id_list`), 19 `d1_gmn.app.migrations.0007_delete_chainidtopersistentid`),
`main()` (in module `d1_util.resolve_package_identifier`), 145
 19 Migration (class in
`main()` (in module `d1_util.solr_query`), 19 `d1_gmn.app.migrations.0008_chainidtopersistentid`),
`main()` (in module `d1_util.xml_apply_xslt`), 19 145
`main()` (in module `d1_util.xml_remove_empty_elements`), Migration (class in
 20 `d1_gmn.app.migrations.0009_auto_20170603_0546`),
`main()` (in module `d1_util.xml_strip_whitespace`), 20 145
`makeCNBaseURL()` (in module `d1_common.url`), 238 Migration (class in
`makeMNBaseURL()` (in module `d1_common.url`), 238 `d1_gmn.app.migrations.0010_auto_20170805_0107`),
`mapIdentity()` (`d1_client.cnclient.CoordinatingNodeClient` 145
 method), 268 Migration (class in
`mapIdentityResponse()` `d1_gmn.app.migrations.0011_resource_map_resource_map_member`
 (`d1_client.cnclient.CoordinatingNodeClient` 145
 method), 268 Migration (class in
`matches_url_pid()` (in module `d1_gmn.app.migrations.0012_auto_20170821_2129`),
`d1_gmn.app.views.assert_sysmeta`), 148 146
`max_size_sysmeta_list()` (in module Migration (class in
`d1_util.generate_object_stats`), 18 `d1_gmn.app.migrations.0013_auto_20171017_0724`),
`MemberNodeClient` (class in `d1_client.mnclient`), 146
 277 Migration (class in
`MemberNodeClient_1_1` (class in `d1_gmn.app.migrations.0014_auto_20171017_0725`),
`d1_client.mnclient_1_1`), 278 146
`MemberNodeClient_1_2` (class in Migration (class in
`d1_client.mnclient_1_2`), 278 `d1_gmn.app.migrations.0015_auto_20171029_0246`),
`MemberNodeClient_2_0` (class in 146
`d1_client.mnclient_2_0`), 279 Migration (class in
`MemberNodeObjectDeleter` (class in `d1_gmn.app.migrations.0016_auto_20180518_1539`),
`d1_util.delete_all_objects_of_type`), 14 146
`MemberNodeObjectDownloader` (class in Migration (class in
`d1_util.download_all_objects`), 15 `d1_gmn.app.migrations.0018_auto_20180901_0115`),
`MemberNodeObjectDownloader` (class in 146

Migration (class in *ObjectListIteratorMulti* (class in *d1_gmn.app.migrations.0019_auto_20190418_1512*), *d1_client.iter.objectlist_multi*), 253
 147
 minixsv, 23
 mk_func() (in module *obsoletes_matches_pid_if_specified()* (in module *d1_gmn.app.views.assert_sysmeta*), 148
d1_common.wrap.access_policy), 200
 MN, 21
 mod_ssl, 23
 mod_wsgi, 23
 MPM, 23
 MultiprocessedIteratorBase (class in *OneDriveSolrClient* (class in *d1_client.iter.base_multi*), 248
 MySQL, 23
N
 name (*d1_gmn.app.gmn.Startup* attribute), 151
 name() (*d1_common.types.exceptions.DataONEException* property), 188
 nested_update() (in module *d1_common.util*), 238
 Node, 21
 node_gen() (*d1_common.cert.subject_info.SubjectInfoNode* property), 171
 NodeListIterator (class in *d1_client.iter.node*), 251
 Nodes (class in *d1_cli.impl.nodes*), 81
 NoResultException, 56
 normalize() (in module *d1_common.multipart*), 215
 normalize() (in module *d1_common.replication_policy*), 218
 normalize_datetime_to_utc() (in module *d1_common.date_time*), 212
 normalize_in_place() (in module *d1_common.system_metadata*), 228
 normalize_request_response_dump() (in module *d1_client.util*), 284
 normalizeTarget() (in module *d1_common.url*), 237
 NotAuthorized, 189
 NotFound, 189
 NotImplemented, 189
O
 OAI-ORE Resource Map, 24
 object_format_dict() (in module *d1_common.object_format_cache.ObjectFormatListCache* property), 217
 ObjectFormatListCache (class in *operations* (*d1_gmn.app.migrations.0001_initial.Migration* attribute), 144
d1_common.object_format_cache), 216
 ObjectListIterator (class in *operations* (*d1_gmn.app.migrations.0002_scienceobject_filename.Migration* attribute), 144
d1_client.iter.objectlist), 252
 ObjectListIteratorAsync (class in *operations* (*d1_gmn.app.migrations.0003_mediatype_mediatypeproper* attribute), 144
d1_gmn.app.management.commands.objectlist_async), 142
operations (*d1_gmn.app.migrations.0004_auto_20170523_0137.Migration* attribute), 144
operations (*d1_gmn.app.migrations.0005_auto_20170527_1554.Migration* attribute), 144
operations (*d1_gmn.app.migrations.0006_chainidtopersistentid.Migration* attribute), 145
operations (*d1_gmn.app.migrations.0007_delete_chainidtopersistentid* attribute), 145
operations (*d1_gmn.app.migrations.0008_chainidtopersistentid.Migration* attribute), 145
operations (*d1_gmn.app.migrations.0009_auto_20170603_0546.Migration* attribute), 145
operations (*d1_gmn.app.migrations.0010_auto_20170805_0107.Migration* attribute), 145
operations (*d1_gmn.app.migrations.0011_resourcemap_resourcemapm* attribute), 145
operations (*d1_gmn.app.migrations.0012_auto_20170821_2129.Migration* attribute), 146

operations (*d1_gmn.app.migrations.0013_auto_20171017_0724*.Migration
 attribute), 146
 operations (*d1_gmn.app.migrations.0014_auto_20171017_0725*.Migration
 attribute), 146
 operations (*d1_gmn.app.migrations.0015_auto_20171029_0246*.Migration
 attribute), 146
 operations (*d1_gmn.app.migrations.0016_auto_20180518_1539*.Migration
 attribute), 146
 operations (*d1_gmn.app.migrations.0018_auto_20180901_0115*.Migration
 attribute), 147
 operations (*d1_gmn.app.migrations.0019_auto_20190418_1512*.Migration
 attribute), 147
 OperationValidator (class in *d1_cli.impl.operation_validator*), 82
 OPTIONS () (*d1_client.session.Session* method), 281
 Oracle, 24
 os_format () (in module *d1_onedrive.impl.util*), 57
 output () (in module *d1_cli.impl.util*), 84

P

pack_echo_header () (in module *d1_test.mock_api.create*), 307
 Param (class in *d1_client.solr_client*), 282
 parent_gen () (*d1_common.cert.subject_info.SubjectInfoNode*
 property), 171
 parse_cmd_line () (in module *d1_util.download_sysmeta_multiproc*), 17
 parse_media_range () (in module *d1_common.ext.mimeparser*), 178
 parse_mime_type () (in module *d1_common.ext.mimeparser*), 178
 parse_resource_map_from_str () (in module *d1_gmn.app.resource_map*), 152
 parse_response () (in module *d1_common.multipart*), 214
 parse_str () (in module *d1_common.multipart*), 214
 parse_xml () (*d1_common.wrap.simple_xml.SimpleXMLWrapper*
 method), 201
 parse_xml_bytes () (in module *d1_scimeta.util*), 290
 parseDoc () (*d1_common.resource_map.ResourceMap*
 method), 225
 parseUrl () (in module *d1_common.url*), 237
 path_arg_dict () (*d1_common.iter.path.ArgParser*
 property), 182
 path_generator () (in module *d1_common.iter.path*), 180
 PathException, 56
 pem_in_http_header_to_pem_in_string ()
 (in module *d1_test.utilities.pem_in_http_header*), 313
 pem_in_string_to_pem_in_http_header ()
 (in module *d1_test.utilities.pem_in_http_header*), 313

ping () (*d1_cli.impl.command_processor.CommandProcessor*
 method), 80
 ping () (*d1_client.baseclient.DataONEBaseClient*
 method), 257
 pingResponse () (*d1_client.baseclient.DataONEBaseClient*
 method), 257
 posix_filename_from_identifier () (in mod-
 ule *d1_onedrive.impl.os_escape*), 56
 posix_identifier_from_filename () (in mod-
 ule *d1_onedrive.impl.os_escape*), 56
 POST () (*d1_client.session.Session* method), 281
 post_has_mime_parts () (in module *d1_gmn.app.views.assert_db*), 147
 postcmd () (*d1_cli.impl.command_parser.CLI*
 method), 77
 PostgreSQL, 23
 postloop () (*d1_cli.impl.command_parser.CLI*
 method), 77
 precmd () (*d1_cli.impl.command_parser.CLI* method),
 77
 preloop () (*d1_cli.impl.command_parser.CLI*
 method), 77
 pretty_format_tree () (in module *d1_scimeta.util*), 291
 print_all_variables ()
 (*d1_cli.impl.session.Session* method), 83
 print_capabilities () (in module *d1_util.display_node_status*), 14
 print_debug () (in module *d1_cli.impl.util*), 84
 print_error () (in module *d1_cli.impl.util*), 84
 print_info () (in module *d1_cli.impl.util*), 84
 print_logging () (in module *d1_common.util*), 239
 print_operation ()
 (*d1_cli.impl.operation_formatter.OperationFormatter*
 method), 82
 print_replication_policy ()
 (*d1_cli.impl.replication_policy.ReplicationPolicy*
 method), 83
 print_single_variable ()
 (*d1_cli.impl.session.Session* method), 83
 print_variable () (*d1_cli.impl.session.Session*
 method), 83
 print_warn () (in module *d1_cli.impl.util*), 84
 process_exception ()
 (*d1_gmn.app.middleware.exception_handler.ExceptionHandler*
 method), 143
 process_input () (in module *d1_test.utilities.urlencode*), 314
 process_row () (*d1_client.solr_client.SolrSearchResponseIterator*
 method), 283
 ProgressLogger (class in *d1_common.utils.progress_logger*), 192
 Pycpg2, 24

PUT() (*d1_client.session.Session* method), 281
 Python, 23

python-dateutil, 23

python-iso8601, 24

python-setuptools, 24

PyXB, 23

pyxb_binding() (*d1_client.baseclient.DataONEBaseClient* property), 257

pyxb_get_namespace_name() (in module *d1_common.type_conversions*), 233

pyxb_get_type_name() (in module *d1_common.type_conversions*), 233

pyxb_is_dataone_exception() (in module *d1_common.types.exceptions*), 186

pyxb_is_v1() (in module *d1_common.type_conversions*), 234

pyxb_is_v2() (in module *d1_common.type_conversions*), 234

pyxb_to_dict() (in module *d1_common.node*), 215

pyxb_to_dict() (in module *d1_common.replication_policy*), 220

pyxb_to_etree() (in module *d1_common.type_conversions*), 235

pyxb_to_str() (in module *d1_common.type_conversions*), 235

pyxb_to_v1_str() (in module *d1_common.type_conversions*), 232

pyxb_to_v2_str() (in module *d1_common.type_conversions*), 233

Q

quality() (in module *d1_common.ext.mimeparser*), 179

quality_parsed() (in module *d1_common.ext.mimeparser*), 179

query() (*d1_client.baseclient_1_1.DataONEBaseClient_1_1* method), 259

query() (*d1_client.cnclient.CoordinatingNodeClient* method), 264

queryResponse() (*d1_client.baseclient_1_1.DataONEBaseClient_1_1* method), 259

queryResponse() (*d1_client.cnclient.CoordinatingNodeClient* method), 264

quote() (in module *d1_onedrive.impl.os_escape*), 56

R

raise_config_error() (*d1_gmn.app.gmn.Startup* method), 151

random_3_words() (in module *d1_test.instance_generator.random_data*), 303

random_bool() (in module *d1_test.instance_generator.random_data*), 304

random_bool_factor() (in module *d1_test.instance_generator.random_data*), 304

random_bytes() (in module *d1_test.instance_generator.random_data*), 303

random_bytes_file() (in module *d1_test.instance_generator.random_data*), 303

random_checksum_algorithm() (in module *d1_test.instance_generator.checksum*), 301

random_choice_pop() (in module *d1_test.instance_generator.random_data*), 304

random_cn() (in module *d1_test.instance_generator.random_data*), 303

random_email() (in module *d1_test.instance_generator.random_data*), 304

random_lower_ascii() (in module *d1_test.instance_generator.random_data*), 303

random_mn() (in module *d1_test.instance_generator.random_data*), 303

random_names() (in module *d1_test.instance_generator.names*), 302

random_regular_or_symbolic_subj() (in module *d1_test.instance_generator.random_data*), 303

random_sized_sample() (in module *d1_test.instance_generator.random_data*), 304

random_sized_sample_pop() (in module *d1_test.instance_generator.random_data*), 304

random_subj() (in module *d1_test.instance_generator.random_data*), 303

random_symbolic_subject() (in module *d1_test.instance_generator.random_data*), 303

random_unicode_char() (in module *d1_test.instance_generator.random_data*), 304

random_unicode_char_no_whitespace() (in module *d1_test.instance_generator.random_data*), 304

random_unicode_name() (in module *d1_test.instance_generator.random_data*), 303

random_unicode_name_list() (in module *d1_test.instance_generator.random_data*), 303

<code>random_unicode_name_unique_list()</code> (in module <code>d1_test.instance_generator.random_data</code>), 303	<code>(d1_client.cnclient.CoordinatingNodeClient method)</code> , 266
<code>random_unicode_str()</code> (in module <code>d1_test.instance_generator.random_data</code>), 304	<code>registerAccountResponse()</code> (<code>d1_client.cnclient.CoordinatingNodeClient method</code>), 266
<code>random_within_range()</code> (in module <code>d1_test.instance_generator.random_data</code>), 304	<code>registerResponse()</code> (<code>d1_client.cnclient.CoordinatingNodeClient method</code>), 273
<code>random_word()</code> (in module <code>d1_test.instance_generator.random_data</code>), 303	<code>remove_allowed_subject()</code> (<code>d1_cli.impl.access_control.AccessControl method</code>), 76
<code>random_word_list()</code> (in module <code>d1_test.instance_generator.random_data</code>), 303	<code>remove_children()</code> (<code>d1_common.wrap.simple_xml.SimpleXMLWrapper method</code>), 204
<code>random_word_unique_list()</code> (in module <code>d1_test.instance_generator.random_data</code>), 304	<code>remove_empty_elements()</code> (in module <code>d1_scimeta.util</code>), 291
<code>random_words()</code> (in module <code>d1_test.instance_generator.words</code>), 305	<code>remove_perm()</code> (<code>d1_common.wrap.access_policy.AccessPolicyWrapper method</code>), 199
<code>rdn_escape()</code> (in module <code>d1_common.cert.x509</code>), 177	<code>remove_perm()</code> (in module <code>d1_common.wrap.access_policy</code>), 200
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.author.Resolver</code> method), 50	<code>remove_subj()</code> (<code>d1_common.wrap.access_policy.AccessPolicyWrapper method</code>), 199
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.d1_object.Resolver</code> method), 50	<code>remove_subj()</code> (in module <code>d1_common.wrap.access_policy</code>), 200
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.flat_space.Resolver</code> method), 51	<code>removeMapIdentity()</code> (<code>d1_client.cnclient.CoordinatingNodeClient method</code>), 268
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.object_tree_resolver.Resolver</code> method), 51	<code>removeMapIdentityResponse()</code> (<code>d1_client.cnclient.CoordinatingNodeClient method</code>), 268
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.region.Resolver</code> method), 51	<code>replace_by_etree()</code> (<code>d1_common.wrap.simple_xml.SimpleXMLWrapper method</code>), 204
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.resource_map.Resolver</code> method), 52	<code>replace_by_xml()</code> (<code>d1_common.wrap.simple_xml.SimpleXMLWrapper method</code>), 204
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.root.RootResolver</code> method), 52	<code>replace_namespace_with_prefix()</code> (in module <code>d1_common.type_conversions</code>), 236
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.single.Resolver</code> method), 52	<code>replicate()</code> (<code>d1_client.mnclient.MemberNodeClient method</code>), 278
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.taxa.Resolver</code> method), 52	<code>replicateResponse()</code> (<code>d1_client.mnclient.MemberNodeClient method</code>), 278
<code>read_file()</code> (<code>d1_onedrive.impl.resolver.time_period.Resolver</code> method), 53	Replication target, 21
<code>ready()</code> (<code>d1_gmn.app.gmn.Startup</code> method), 151	ReplicationPolicy (class in <code>d1_cli.impl.replication_policy</code>), 83
<code>reformat_to_pretty_xml()</code> (in module <code>d1_common.xml</code>), 242	<code>RepZoteroClient</code> (class in <code>d1_test.replication_tester.replication_tester</code>), 310
<code>refresh()</code> (<code>d1_onedrive.impl.clients.onedrive_zotero_client.ZoteroClient</code> method), 49	<code>ReplicationTesterError</code> , 309
<code>refresh()</code> (<code>d1_onedrive.impl.object_tree.ObjectTree</code> method), 55	<code>requestMapIdentity()</code> (<code>d1_client.cnclient.CoordinatingNodeClient method</code>), 83
<code>refresh_cache()</code> (<code>d1_common.object_format_cache.ObjectFormatCache</code> method), 217	
<code>register()</code> (<code>d1_client.cnclient.CoordinatingNodeClient</code> method), 273	
<code>registerAccount()</code>	

requestMapIdentityResponse() (d1_client.cnclient.CoordinatingNodeClient method), 269
 reserveIdentifier() (d1_client.cnclient.CoordinatingNodeClient method), 261
 reserveIdentifierResponse() (d1_client.cnclient.CoordinatingNodeClient method), 261
 reset() (d1_cli.impl.session.Session method), 83
 resolve() (d1_cli.impl.command_processor.CommandProcessor method), 80
 resolve() (d1_client.cnclient.CoordinatingNodeClient method), 263
 resolve_schemas() (in module d1_scimeta.schema_resolve), 287
 resolve_sid() (in module d1_gmn.app.revision), 153
 ResolveError, 19, 20, 287
 Resolver (class in d1_onedrive.impl.resolver.author), 50
 Resolver (class in d1_onedrive.impl.resolver.d1_object), 50
 Resolver (class in d1_onedrive.impl.resolver.flat_space), 51
 Resolver (class in d1_onedrive.impl.resolver.object_tree_resolver), 51
 Resolver (class in d1_onedrive.impl.resolver.region), 51
 Resolver (class in d1_onedrive.impl.resolver.resolver_base), 51
 Resolver (class in d1_onedrive.impl.resolver.resource_map), 52
 Resolver (class in d1_onedrive.impl.resolver.single), 52
 Resolver (class in d1_onedrive.impl.resolver.taxa), 52
 Resolver (class in d1_onedrive.impl.resolver.time_period), 53
 resolveResponse() (d1_client.cnclient.CoordinatingNodeClient method), 263
 ResourceMap (class in d1_common.resource_map), 221
 REST, 24
 revision_list_to_obsoleted_by_dict() (in module d1_common.revision), 226
 revision_list_to_obsoletes_dict() (in module d1_common.revision), 226
 RootResolver (class in d1_onedrive.impl.resolver.root), 52
 round_to_nearest() (in module d1_common.date_time), 213
 run() (d1_test.replication_tester.replication_server.TestHTTPServer method), 309
 run_command_line_arguments() (d1_cli.impl.command_parser.CLI method), 77
 run_solr_query() (d1_onedrive.impl.clients.onedrive_solr_client.OnedriveSolrClient method), 48
S
 sanity() (in module d1_gmn.app.views.assert_sysmeta), 148
 save() (d1_cli.impl.session.Session method), 83
 save() (in module d1_test.test_files), 315
 save_bytes_to_file() (in module d1_scimeta.util), 290
 save_json() (in module d1_common.util), 240
 save_pem() (in module d1_common.cert.x509), 177
 save_store_version() (in module d1_gmn.app.sciobj_store), 156
 save_tree_to_file() (in module d1_scimeta.util), 290
 save_xml() (in module d1_util.download_test_docs), 17
 Science Data, 21
 Science Metadata, 21
 science_object_archive() (d1_cli.impl.command_processor.CommandProcessor method), 80
 science_object_create() (d1_cli.impl.command_processor.CommandProcessor method), 80
 science_object_get() (d1_cli.impl.command_processor.CommandProcessor method), 80
 science_object_update() (d1_cli.impl.command_processor.CommandProcessor method), 80
 SciMetaError, 291
 search() (d1_cli.impl.command_processor.CommandProcessor method), 80
 search() (d1_client.cnclient.CoordinatingNodeClient method), 264
 search() (d1_client.solr_client.SolrClient method), 282
 searchResponse() (d1_client.cnclient.CoordinatingNodeClient method), 264
 Self signed certificate, 22
 send_request() (d1_gmn.tests.gmn_wsgi.WSGIClient method), 158
 serialize_cert_to_der() (in module d1_common.cert.x509), 177
 serialize_cert_to_pem() (in module d1_common.cert.x509), 175
 serialize_for_transport() (in module d1_common.xml), 242
 serialize_gen() (in module d1_common.xml), 241

```

serialize_private_key_to_pem() (in module d1_common.cert.x509), 176
serialize_to_display() (in module d1_common.resource_map.ResourceMap method), 222
serialize_to_display() (in module d1_common.types.exceptions.DataONEExceptions method), 187
serialize_to_headers() (in module d1_common.types.exceptions.DataONEExceptions method), 188
serialize_to_normalized_compact_json() (in module d1_common.util), 240
serialize_to_normalized_pretty_json() (in module d1_common.util), 240
serialize_to_transport() (in module d1_common.resource_map.ResourceMap method), 222
serialize_to_transport() (in module d1_common.types.exceptions.DataONEExceptions method), 187
serialize_to_xml_str() (in module d1_common.xml), 242
Server key, 21
Server Side Authentication, 22
Server side certificate, 22
ServiceFailure, 189
Session (class in d1_cli.impl.session), 83
Session (class in d1_client.session), 279
session() (d1_client.aio.async_client.AsyncDataONEClient property), 248
SessionVariable (class in d1_cli.impl.const), 81
set() (d1_cli.impl.session.Session method), 83
set_attr_text() (d1_common.wrap.simple_xml.SimpleXMLWrapper method), 203
set_cn_subjects_for_environment() (in module d1_gmn.app.node_registry), 151
set_element_dt() (d1_common.wrap.simple_xml.SimpleXMLWrapper method), 203
set_element_text() (d1_common.wrap.simple_xml.SimpleXMLWrapper method), 202
set_element_text_by_attr_key() (d1_common.wrap.simple_xml.SimpleXMLWrapper method), 203
set_empty_cn_subjects_cache() (in module d1_gmn.app.node_registry), 151
set_number_of_replicas() (d1_cli.impl.replication_policy.ReplicationPolicy method), 83
set_replication_allowed() (d1_cli.impl.replication_policy.ReplicationPolicy method), 83
set_revision_links() (in module d1_gmn.app.revision), 153
set_with_conversion() (d1_cli.impl.session.Session method), 83
setAccessPolicy() (d1_client.cnclient.CoordinatingNodeClient method), 266
setAccessPolicyResponse() (d1_client.cnclient.CoordinatingNodeClient method), 265
setDocumentedBy() (d1_common.resource_map.ResourceMap method), 224
setDocuments() (d1_common.resource_map.ResourceMap method), 223
setObsoletedBy() (d1_client.cnclient.CoordinatingNodeClient method), 262
setObsoletedByResponse() (d1_client.cnclient.CoordinatingNodeClient method), 262
setReplicationPolicy() (d1_client.cnclient.CoordinatingNodeClient method), 271
setReplicationPolicyResponse() (d1_client.cnclient.CoordinatingNodeClient method), 271
setReplicationStatus() (d1_client.cnclient.CoordinatingNodeClient method), 270
setReplicationStatusResponse() (d1_client.cnclient.CoordinatingNodeClient method), 270
setRightsHolder() (d1_client.cnclient.CoordinatingNodeClient method), 265
setRightsHolderResponse() (d1_client.cnclient.CoordinatingNodeClient method), 265
sign_csr() (in module d1_util.cert_sign_csr), 11
SimpleXMLWrapper (class in d1_common.wrap.simple_xml), 201
SimpleXMLWrapperException, 204
Singleton (class in d1_common.object_format_cache), 216
size() (d1_common.iter.bytes.BytesIterator property), 179
size() (d1_common.iter.stream.StreamIterator property), 183
size() (d1_common.iter.string.StringIterator property), 183
size() (d1_onedrive.impl.attributes.Attributes method), 54
slash() (d1_gmn.tests.gmn_test_client.GMNTTestClient method), 158

```


SlenderNodeTestClient (class in *d1_common.type_conversions*), 234
d1_test.slender_node_test_client), 314
 Solr, 24
 solr_query() (*d1_onedrive.impl.tests.command_echoer.CommandEchoer* (in module *d1_common.type_conversions*), 232
 method), 53
 solr_query_raw() (*d1_onedrive.impl.tests.command_echoer.CommandEchoer* (in module *d1_common.type_conversions*), 232
 method), 53
 SolrArrayResponseIterator (class in *str_to_v2_str* (in module *d1_common.type_conversions*), 232
d1_client.solr_client), 284
 SolrArrayTransformer (class in *StreamIterator* (class in *d1_common.iter.stream*), 183
d1_client.solr_client), 283
 SolrClient (class in *d1_client.solr_client*), 282
 SolrRecordTransformerBase (class in *d1_onedrive.impl.util*), 57
d1_client.solr_client), 283
 SolrSearchResponseIterator (class in *StringIterator* (class in *d1_common.iter.string*), 183
d1_client.solr_client), 283
 SolrSubsampleResponseIterator (class in *strip_log* (in module *d1_common.type_conversions*), 236
d1_client.solr_client), 284
 SolrValuesResponseIterator (class in *strip_logEntry* (in module *d1_common.type_conversions*), 236
d1_client.solr_client), 284
 sort() (*d1_test.xml_normalize.StableNode* method), *d1_common.type_conversions*), 236
 316
 sort_elements_by_child_values() (in module *d1_common.xml*), 243
 sort_value_list_pyxb() (in module *d1_common.type_conversions*), 236
d1_common.xml), 243
 SQLite3, 24
 SSL, 22
 SSL handshake, 22
 StableNode (class in *d1_test.xml_normalize*), 316
 StableTree (in module *d1_test.xml_normalize*), 316
 start_task() (*d1_common.utils.progress_logger.ProgressLogger* (in module *d1_common.url*), 237
 method), 193
 start_task_type() (*d1_common.wrap.access_policy.AccessPolicyWrap* (in module *d1_common.wrap.access_policy*), 199
d1_common.utils.progress_logger.ProgressLogger method), 193
 method), 193
 Startup (class in *d1_gmn.app.gmn*), 150
 stop() (*d1_test.replication_tester.replication_server.HTTPServer* (in module *d1_common.cert.subject_info*), 171
 method), 310
 attribute), 171
 str_is_error() (in module *SubjectInfoNode* (class in *d1_common.cert.subject_info*), 171
d1_common.type_conversions), 234
 str_is_identifier() (in module *SubjectInfoTree* (in module *d1_common.cert.subject_info*), 172
d1_common.type_conversions), 234
 str_is_objectList() (in module *Subversion*, 23
d1_common.type_conversions), 234
 str_is_v1() (in module *sync_cache_with_source_tree* (in module *d1_onedrive.impl.object_tree.ObjectTree*
d1_common.type_conversions), 234
 method), 56
 str_is_v2() (in module *SynchronizationFailed*, 190
d1_common.type_conversions), 234
 str_is_well_formed() (in module *synchronizationFailed* (in module *d1_client.mnclient.MemberNodeClient*
d1_common.type_conversions), 234
 method), 277
 str_to_etree() (in module *synchronizationFailedResponse* (in module *d1_client.mnclient.MemberNodeClient*
d1_common.type_conversions), 235
 method), 277
 str_to_pyxb() (in module

[synchronize\(\) \(d1_client.aio.async_client.AsyncDataONEClient_by_status_code\(\) \(in module *d1_test.mock_api.d1_exception*\), 307](#)
[method\), 248](#)
[synchronize\(\) \(d1_client.cnclient_2_0.CoordinatingNodeClient_2_0\(\) \(in module *d1_common.date_time*\), 210](#)
[method\), 274](#)
[synchronizeResponse\(\) \(d1_client.cnclient_2_0.CoordinatingNodeClient_2_0\(\) \(in module *d1_common.cert.jwt*\), 166](#)
[method\), 274](#)
[sysmeta_add_blocked\(\) \(in module *d1_common.replication_policy*\), 218](#)
[sysmeta_add_preferred\(\) \(in module *d1_common.replication_policy*\), 217](#)
[sysmeta_set_default_rp\(\) \(in module *d1_common.replication_policy*\), 218](#)
[SysMetaRetrieveError, 17](#)
[System Metadata, 21](#)
[system_metadata_get\(\) \(d1_cli.impl.command_processor.CommandProcessor \(in module *d1_cli.impl.command_processor*\), 56](#)
[method\), 80](#)
[systemMetadataChanged\(\) \(d1_client.mnclient.MemberNodeClient \(in module *d1_client.mnclient*\), 278](#)
[method\), 278](#)
[systemMetadataChangedResponse\(\) \(d1_client.mnclient.MemberNodeClient \(in module *d1_client.mnclient*\), 278](#)
[method\), 278](#)
[SystemMetadataCreator \(class in *d1_cli.impl.system_metadata*\), 84](#)
[SystemMetadataIteratorMulti \(class in *d1_client.iter.sysmeta_multi*\), 253](#)

T

[test_dst_mn\(\) \(d1_test.replication_tester.replication_tester.ReplicationTester \(in module *d1_test.replication_tester*\), 310](#)
[method\), 310](#)
[test_src_mn\(\) \(d1_test.replication_tester.replication_tester.ReplicationTester \(in module *d1_test.replication_tester*\), 310](#)
[method\), 310](#)
[TestHTTPServer \(class in *d1_test.replication_tester.replication_server*\), 309](#)
[TLS, 22](#)
[to_iso8601_utc\(\) \(in module *d1_common.date_time*\), 213](#)
[ToJsonCompatibleTypes \(class in *d1_common.util*\), 240](#)
[topological_sort\(\) \(in module *d1_common.revision*\), 226](#)
[total\(\) \(d1_client.iter.base_multi.MultiprocessedIteratorBase \(in module *d1_client.iter.base_multi*\), 249](#)
[property\), 249](#)
[transform\(\) \(d1_client.solr_client.SolrArrayTransformer \(in module *d1_client.solr_client*\), 283](#)
[method\), 283](#)
[transform\(\) \(d1_client.solr_client.SolrRecordTransformerBase \(in module *d1_client.solr_client*\), 283](#)
[method\), 283](#)
[trigger_by_header\(\) \(in module *d1_test.mock_api.d1_exception*\), 307](#)
[trigger_by_pid\(\) \(in module *d1_test.mock_api.d1_exception*\), 307](#)

[unpack_echo_header\(\) \(in module *d1_test.mock_api.create*\), 307](#)
[unquote\(\) \(in module *d1_onedrive.impl.os_escape*\), 56](#)
[UnsupportedMetadataType, 189](#)
[UnsupportedType, 190](#)
[update\(\) \(d1_cli.impl.operation_maker.OperationMaker \(in module *d1_cli.impl.operation_maker*\), 82](#)
[method\), 82](#)
[update\(\) \(d1_client.mnclient.MemberNodeClient \(in module *d1_client.mnclient*\), 278](#)
[method\), 278](#)
[update\(\) \(d1_common.wrap.access_policy.AccessPolicyWrapper \(in module *d1_common.wrap.access_policy*\), 197](#)
[method\), 197](#)
[update\(\) \(d1_test.slender_node_test_client.SlenderNodeTestClient \(in module *d1_test.slender_node_test_client*\), 314](#)
[method\), 314](#)
[update\(\) \(in module *d1_common.wrap.access_policy*\), 199](#)
[update_access_policy\(\) \(d1_cli.impl.command_processor.CommandProcessor \(in module *d1_cli.impl.command_processor*\), 80](#)
[method\), 80](#)
[update_replication_policy\(\) \(d1_cli.impl.operation_maker.OperationMaker \(in module *d1_cli.impl.operation_maker*\), 82](#)
[method\), 82](#)
[update_elements\(\) \(in module *d1_common.system_metadata*\), 229](#)
[update_replication_policy\(\) \(d1_cli.impl.operation_maker.OperationMaker \(in module *d1_cli.impl.operation_maker*\), 82](#)
[method\), 82](#)
[updateAccount\(\) \(d1_client.cnclient.CoordinatingNodeClient \(in module *d1_client.cnclient*\), 266](#)
[method\), 266](#)
[updateAccountResponse\(\) \(d1_client.cnclient.CoordinatingNodeClient \(in module *d1_client.cnclient*\), 266](#)
[method\), 266](#)
[updateBaseGroup\(\) \(d1_client.cnclient.CoordinatingNodeClient \(in module *d1_client.cnclient*\), 270](#)
[method\), 270](#)
[updateGroupResponse\(\) \(d1_client.cnclient.CoordinatingNodeClient \(in module *d1_client.cnclient*\), 270](#)
[method\), 270](#)
[updateNodeCapabilities\(\) \(in module *d1_client.cnclient*\), 270](#)

- (*d1_client.cnclient.CoordinatingNodeClient* method), 273
- updateNodeCapabilitiesResponse()* (*d1_client.cnclient.CoordinatingNodeClient* method), 272
- updateReplicationMetadata()* (*d1_client.cnclient.CoordinatingNodeClient* method), 271
- updateReplicationMetadataResponse()* (*d1_client.cnclient.CoordinatingNodeClient* method), 271
- updateResponse()* (*d1_client.mnclient.MemberNodeClient* method), 277
- updateSystemMetadata()* (*d1_client.baseclient_2_0.DataONEBaseClient_2_0* method), 260
- updateSystemMetadataResponse()* (*d1_client.baseclient_2_0.DataONEBaseClient_2_0* method), 260
- url_is_http_or_https()* (in module *d1_gmn.app.views.assert_db*), 147
- url_is_retrievable()* (in module *d1_gmn.app.views.assert_db*), 147
- urlencode()* (in module *d1_common.url*), 237
- UTC (class in *d1_common.date_time*), 208
- utc_now()* (in module *d1_common.date_time*), 212
- utcoffset()* (*d1_common.date_time.FixedOffset* method), 209
- utcoffset()* (*d1_common.date_time.UTC* method), 208
- ## V
- v2_0_tag()* (in module *d1_common.type_conversions*), 236
- validate_and_decode()* (in module *d1_common.cert.jwt*), 165
- validate_and_decode()* (in module *d1_util.jwt_token_tasks*), 18
- validate_bagit_file()* (in module *d1_common.bagit*), 205
- validate_bulk()* (in module *d1_util.download_test_docs*), 17
- validate_bytes()* (in module *d1_scimeta.validate*), 292
- validate_format_id()* (in module *d1_scimeta.tests.validate_test_docs*), 286
- validate_jwt_and_get_subject_list()* (in module *d1_gmn.app.middleware.session_jwt*), 143
- validate_path()* (in module *d1_scimeta.validate*), 292
- validate_tree()* (in module *d1_scimeta.validate*), 292
- validate_xml()* (in module *d1_scimeta.tests.validate_test_docs*), 286
- Vendor specific extensions, 21
- verifyAccount()* (*d1_client.cnclient.CoordinatingNodeClient* method), 267
- verifyAccountResponse()* (*d1_client.cnclient.CoordinatingNodeClient* method), 267
- VersionMismatch, 190
- view()* (*d1_client.cnclient_2_0.CoordinatingNodeClient_2_0* method), 275
- view()* (*d1_client.mnclient_1_2.MemberNodeClient_1_2* method), 278
- viewResponse()* (*d1_client.cnclient_2_0.CoordinatingNodeClient_2_0* method), 274
- viewResponse()* (*d1_client.mnclient_1_2.MemberNodeClient_1_2* method), 278
- ## W
- whitelist_subject()* (*d1_gmn.tests.gmn_test_client.GMNTTestClient* method), 157
- windows_filename_from_identifier()* (in module *d1_onedrive.impl.os_escape*), 56
- windows_identifier_from_filename()* (in module *d1_onedrive.impl.os_escape*), 56
- Workspace, 21
- wrap()* (in module *d1_common.wrap.access_policy*), 196
- wrap()* (in module *d1_common.wrap.simple_xml*), 201
- wrap_sysmeta_pyxb()* (in module *d1_common.wrap.access_policy*), 196
- WSGI, 23
- WSGIClient (class in *d1_gmn.tests.gmn_wsgi*), 158
- ## X
- X.509, 21
- xml_is_dataone_exception()* (in module *d1_common.types.exceptions*), 186
- xml_to_stabletree()* (in module *d1_test.xml_normalize*), 316
- xsd_datetime_str_from_dt()* (in module *d1_common.date_time*), 211
- ## Z
- ZoteroClient (class in *d1_onedrive.impl.clients.onedrive_zotero_client*), 49