

---

# DataONE Python Products

unknown

2019-04-05



## CONTENTS

<b>1</b>	<b>Utilities (for end users)</b>	<b>3</b>
1.1	DataONE ONEDrive . . . . .	3
1.2	DataONE Command Line Interface . . . . .	3
<b>2</b>	<b>Member Node (for Member Node partners)</b>	<b>5</b>
2.1	Generic Member Node (GMN) . . . . .	5
<b>3</b>	<b>Python Libraries (for software developers)</b>	<b>7</b>
3.1	DataONE Common Library for Python . . . . .	7
3.2	DataONE Client Library for Python . . . . .	7
3.3	DataONE Test Utilities . . . . .	7
<b>4</b>	<b>Contents</b>	<b>9</b>
4.1	DataONE ONEDrive . . . . .	9
4.2	DataONE Command Line Interface (CLI) . . . . .	37
4.3	Generic Member Node (GMN) . . . . .	55
4.4	Indices and tables . . . . .	110
4.5	DataONE Common Library for Python . . . . .	110
4.6	Indices and tables . . . . .	187
4.7	DataONE Client Library for Python . . . . .	187
4.8	Indices and tables . . . . .	221
4.9	DataONE Test Utilities . . . . .	221
	<b>Python Module Index</b>	<b>245</b>



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 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.

### 1.2 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 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

### 4.1 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:

#### 4.1.1 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.

### 4.1.2 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:
```

(continues on next page)

(continued from previous page)

```
--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.

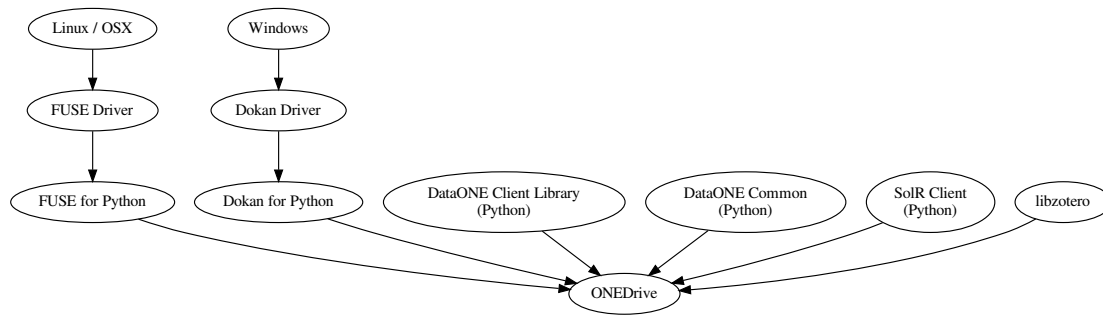
Because the DataONE API currently does not specify a way for Member Nodes to allow partial downloads of objects, ONEDrive downloads the entire object the first time it is accessed through the filesystem. If the object is large, the filesystem will appear to freeze while this download is being performed 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.

### 4.1.3 Architecture



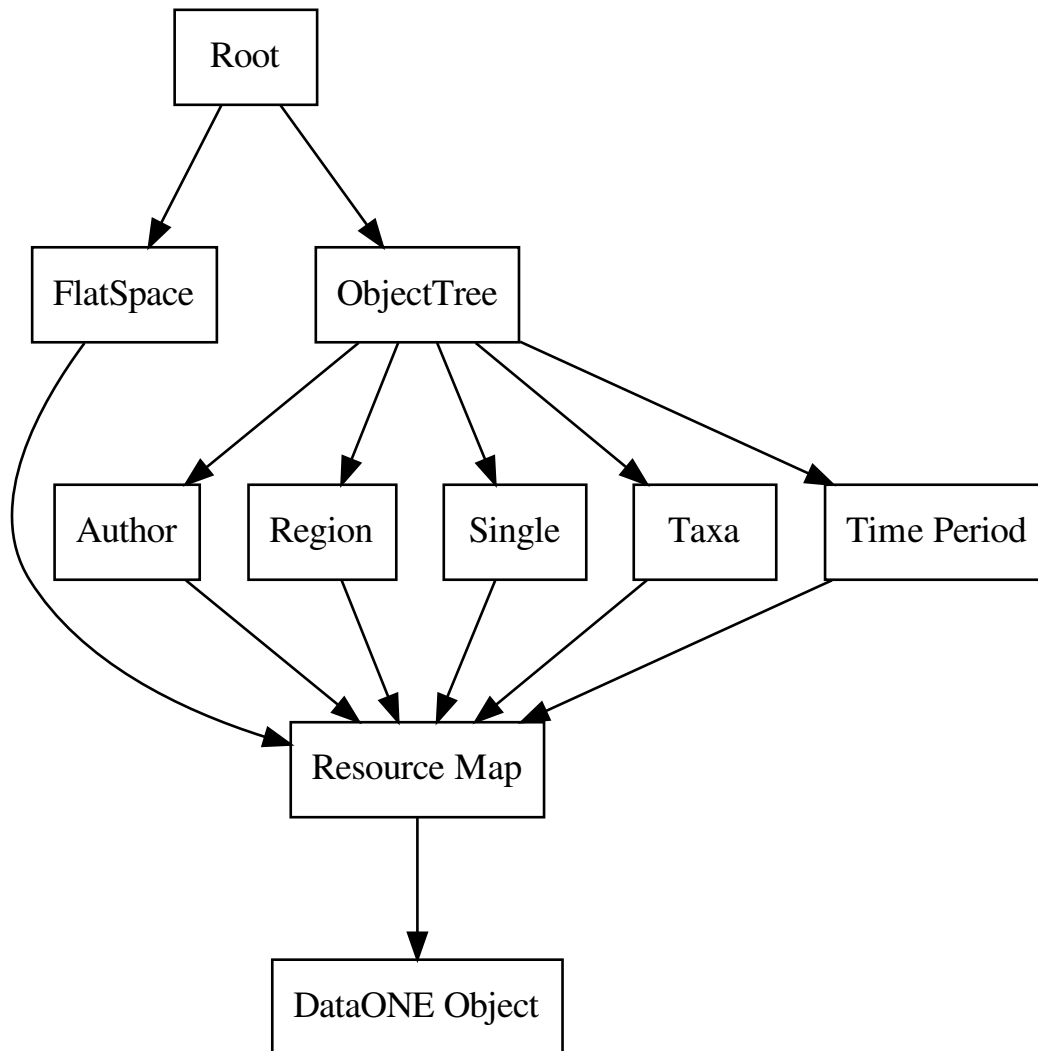
### Resolvers

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.

#### 4.1.4 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.

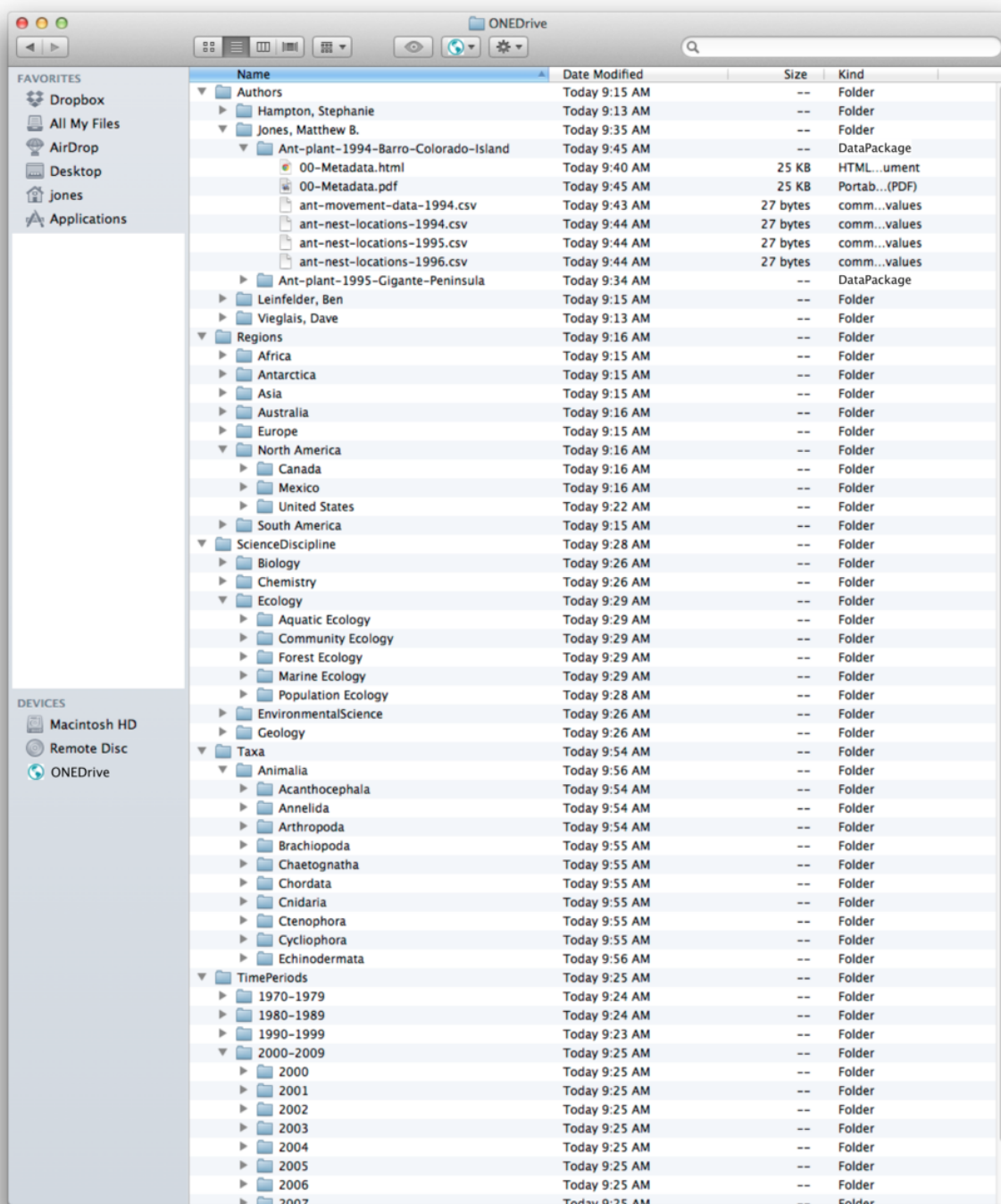


Fig. 1: *Figure 1.* Mac OS X Mockup of the hierarchical folder view.

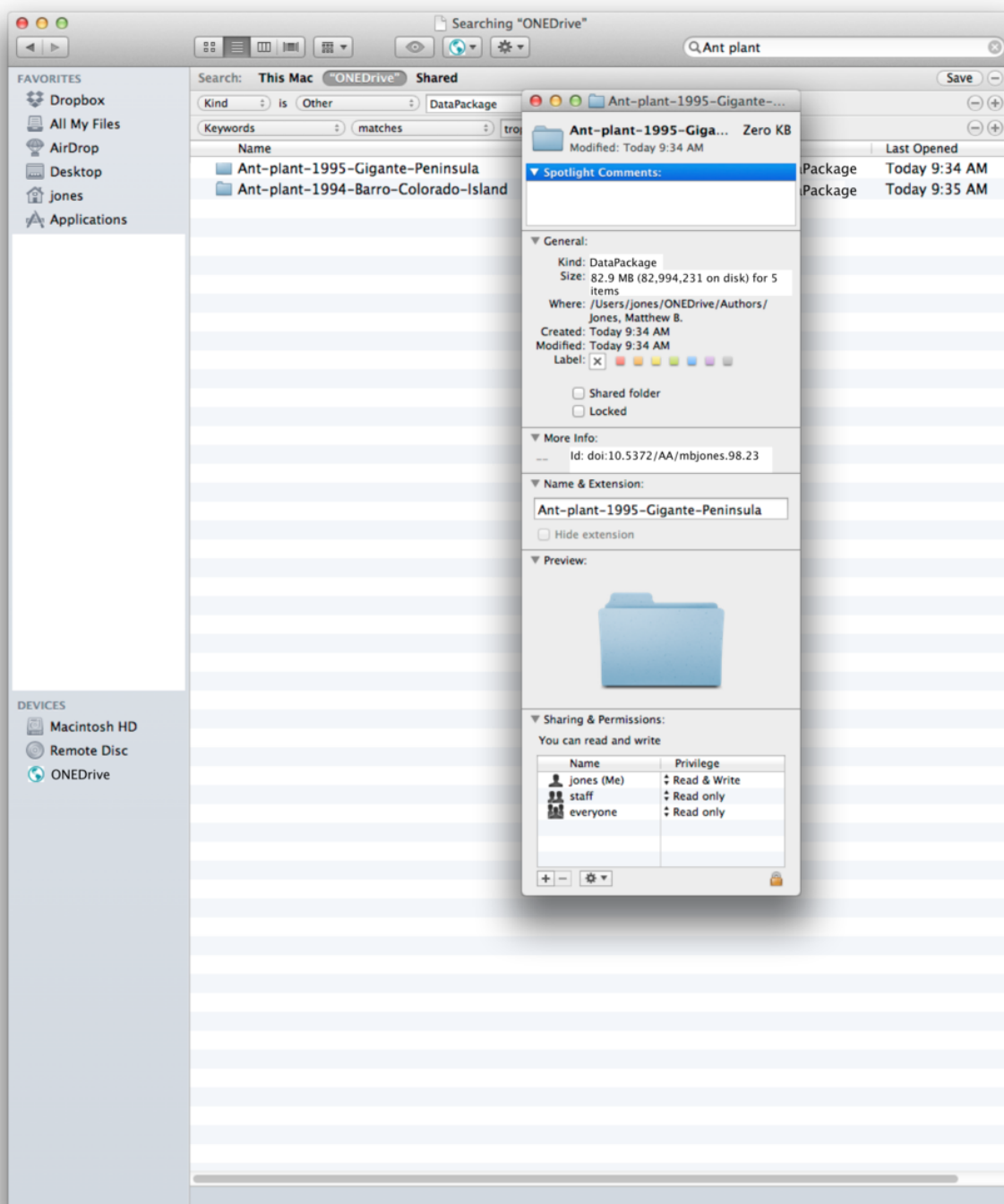


Fig. 2: Figure 2. Mac OS X Get Info dialog is used to display system metadata.

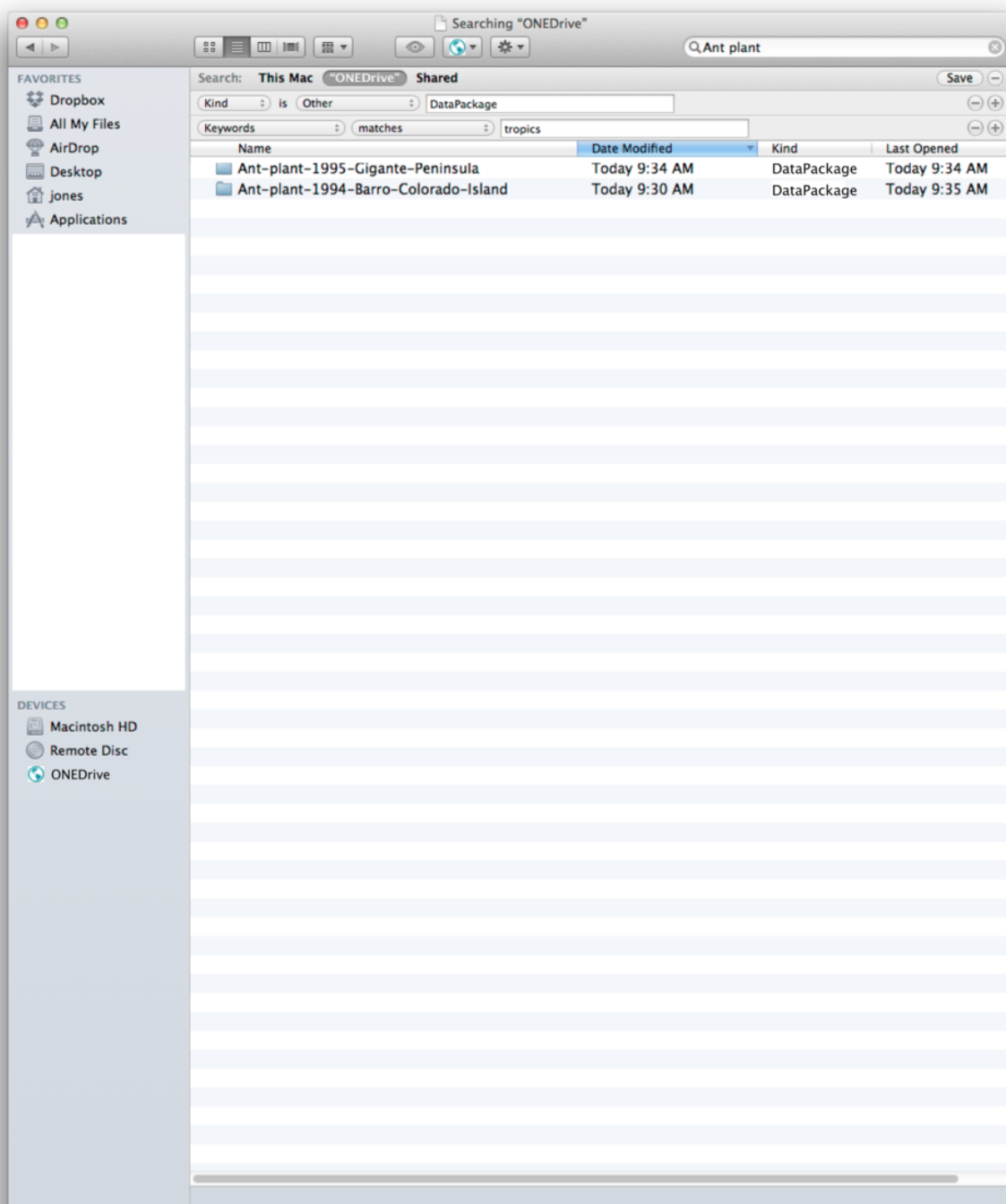


Fig. 3: *Figure 3.* Mac OS X Mockup of the filter UI showing a restricted set of data packages.

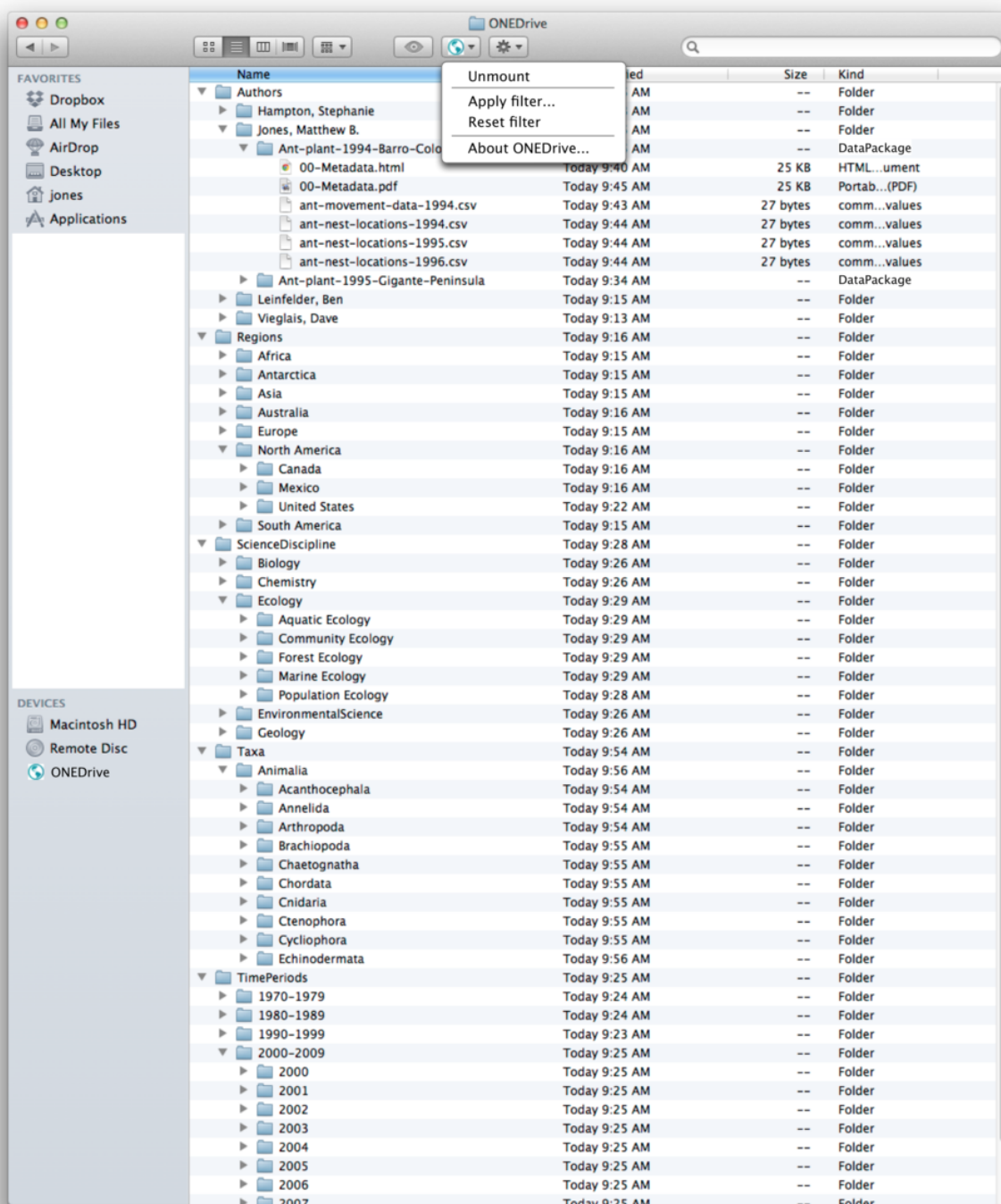


Fig. 4: Figure 1. Mac OS X Mockup showing the ONEDrive dropdown menu.

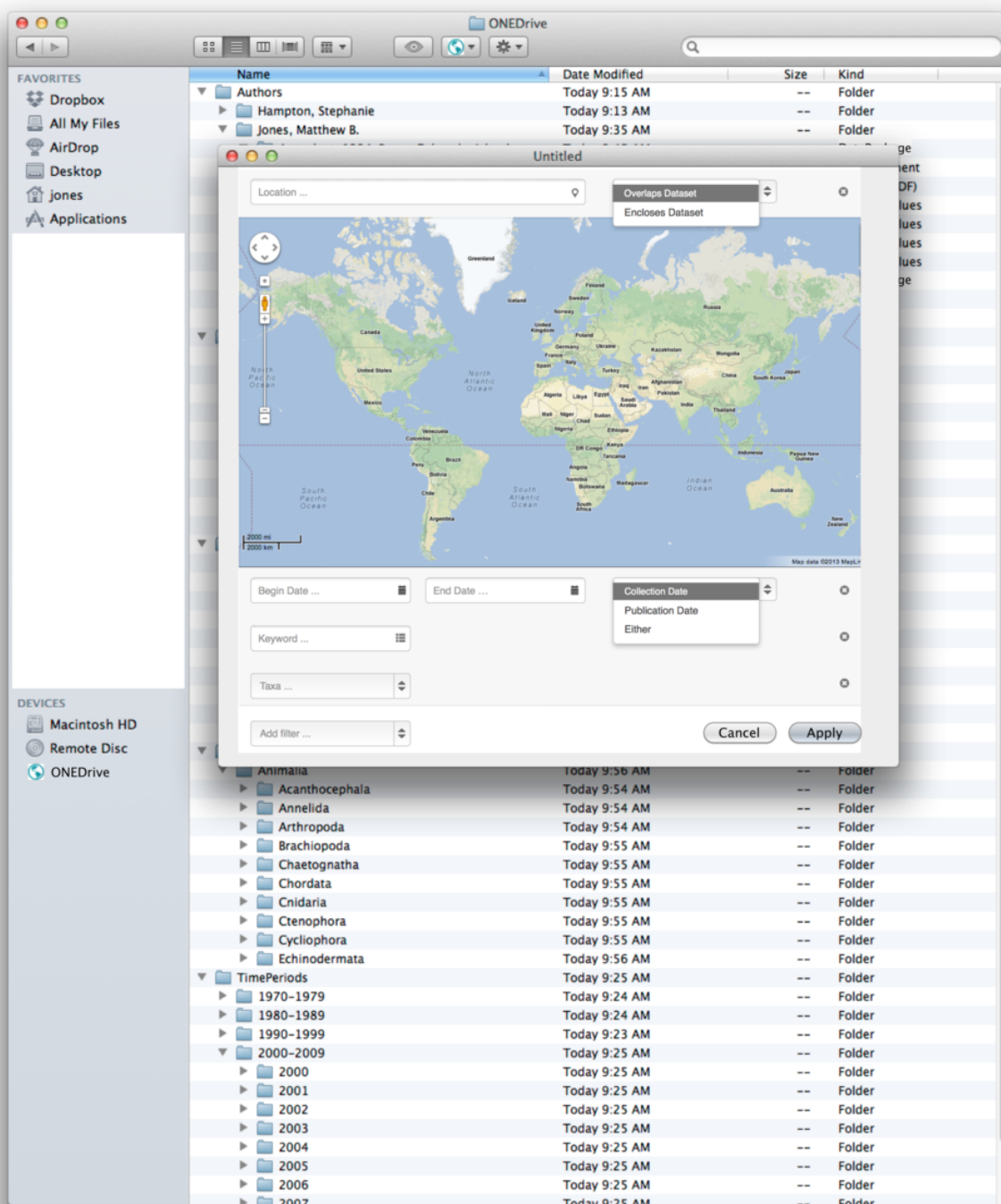
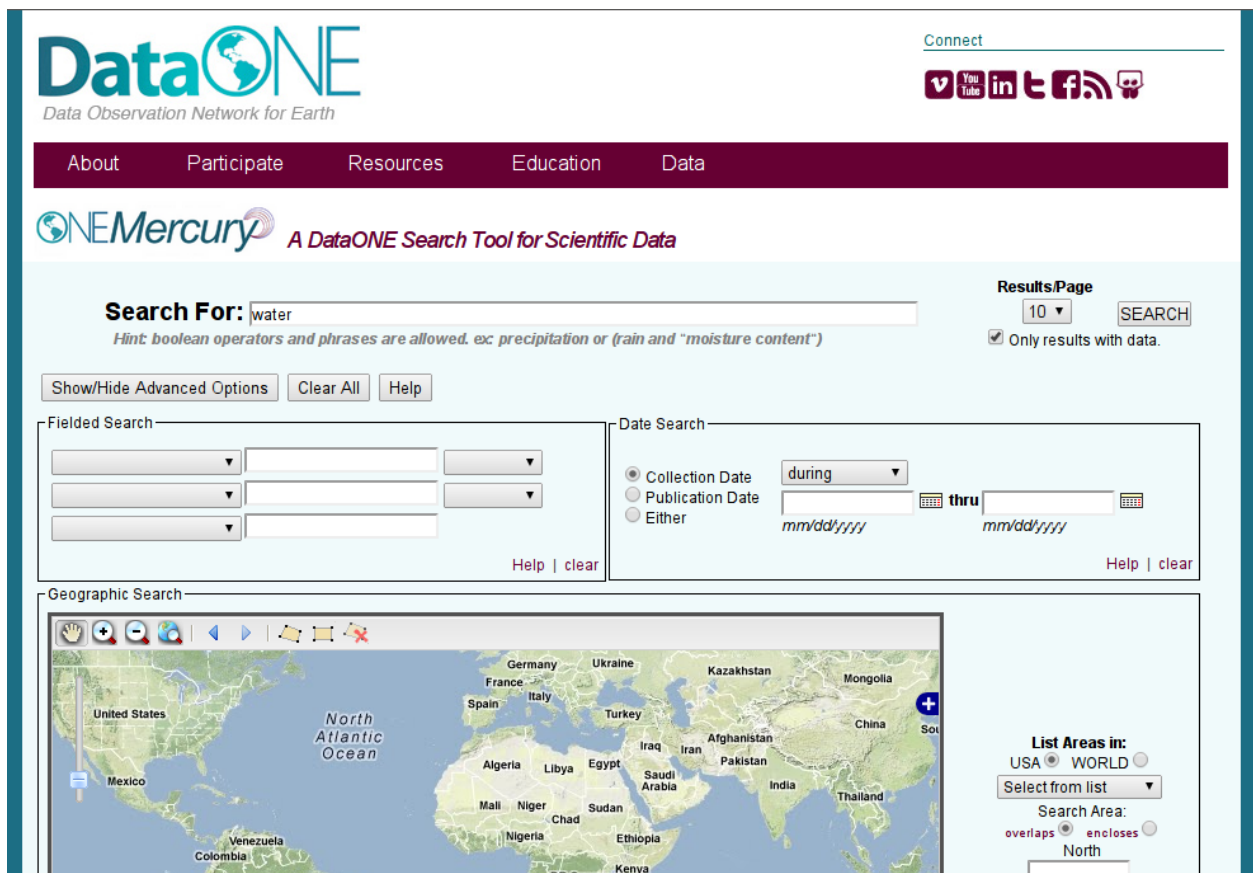


Fig. 5: *Figure 1.* Mac OS X Mockup showing the filter dialog for constraining what is shown in the window.



**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 authors with their document counts, such as 'CDPH Merced District (810)' and 'Cal Water Service (713)'. The 'Filter by keywords' panel lists keywords like 'California (16788)', 'contaminants (16788)', and 'drinking water (16788)'. Below the filters, there is a 'Sort By' section with options: '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.' It includes an identifier and a description. Below the description are four buttons: 'View full metadata', 'Download', 'Open', and 'Add to Folder'. The 'Open' button is circled in red, and a mouse cursor is pointing at it. The second document is 'Cal Water Service. 04/05/2002. DWSAP Assessment for 4900546-001: HAWKINS WATER CO-CAL WATER SERVICE (PUC) / WELL 02.' It also includes an identifier and a description, with the same four buttons below it. 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 at the bottom: '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.

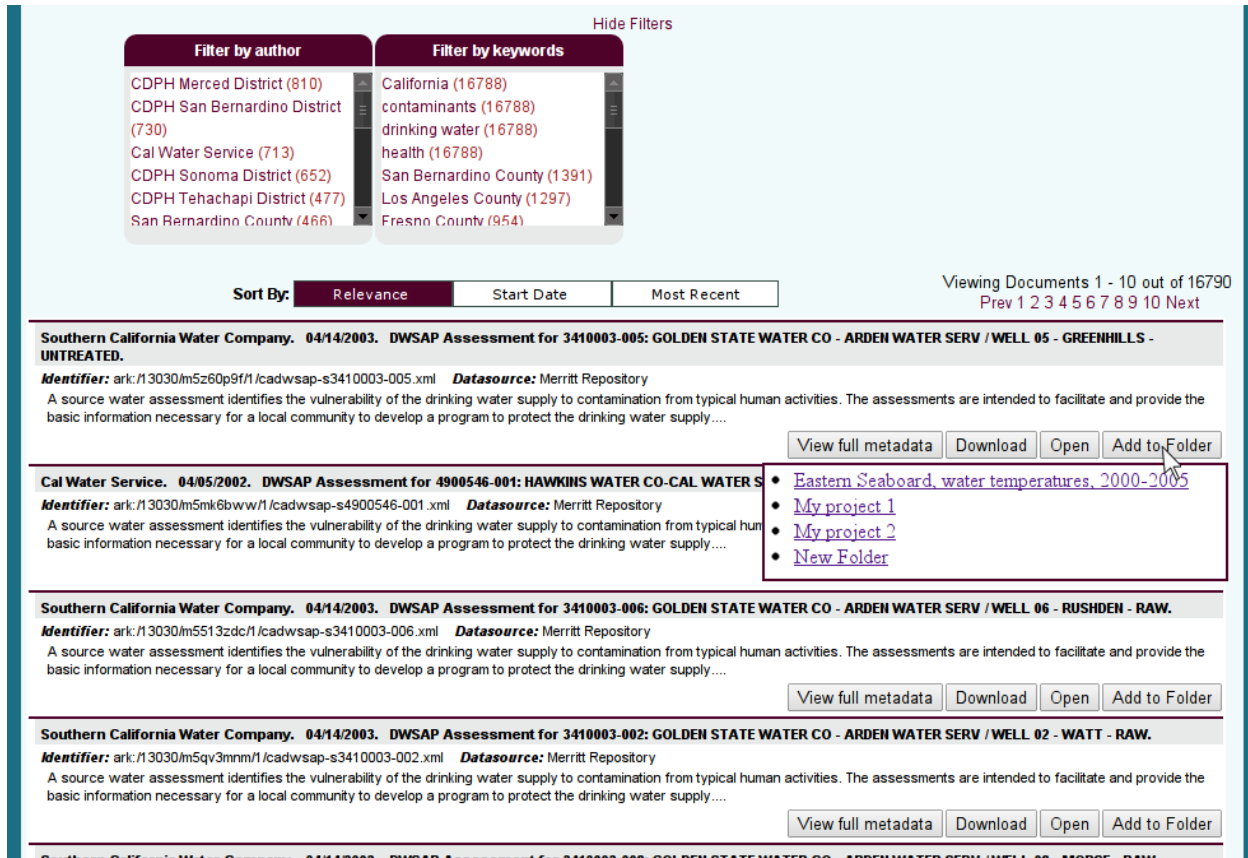


Fig. 8: Figure 3. Saving a single object

- 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.

**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: 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.

### 4.1.5 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`.

### 4.1.6 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](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](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](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](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](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](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.

### 4.1.7 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](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](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/>

### 4.1.8 Indices and tables

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

## 4.2 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:

### 4.2.1 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
```

### 4.2.2 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/client/certs/myclientcert.pem
```

View all the session variables:

```
> set
```

Save the session variables to a file for later use:

```
> save ~/dl/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/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
```

### 4.2.3 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 *myembernode*, store it in *myfile* and exit.

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

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

```
$ dataone 'set mn-url http://myembernode.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

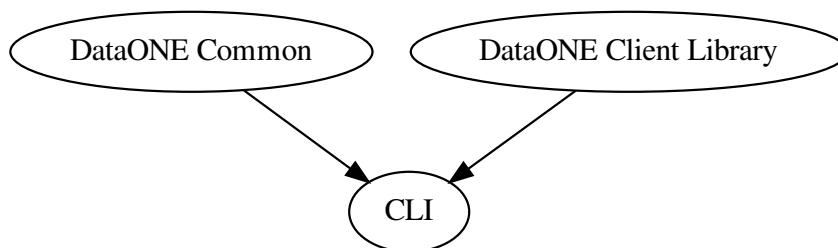
Remove the operations in the queue of write operations without performing them

### Overview of session variables

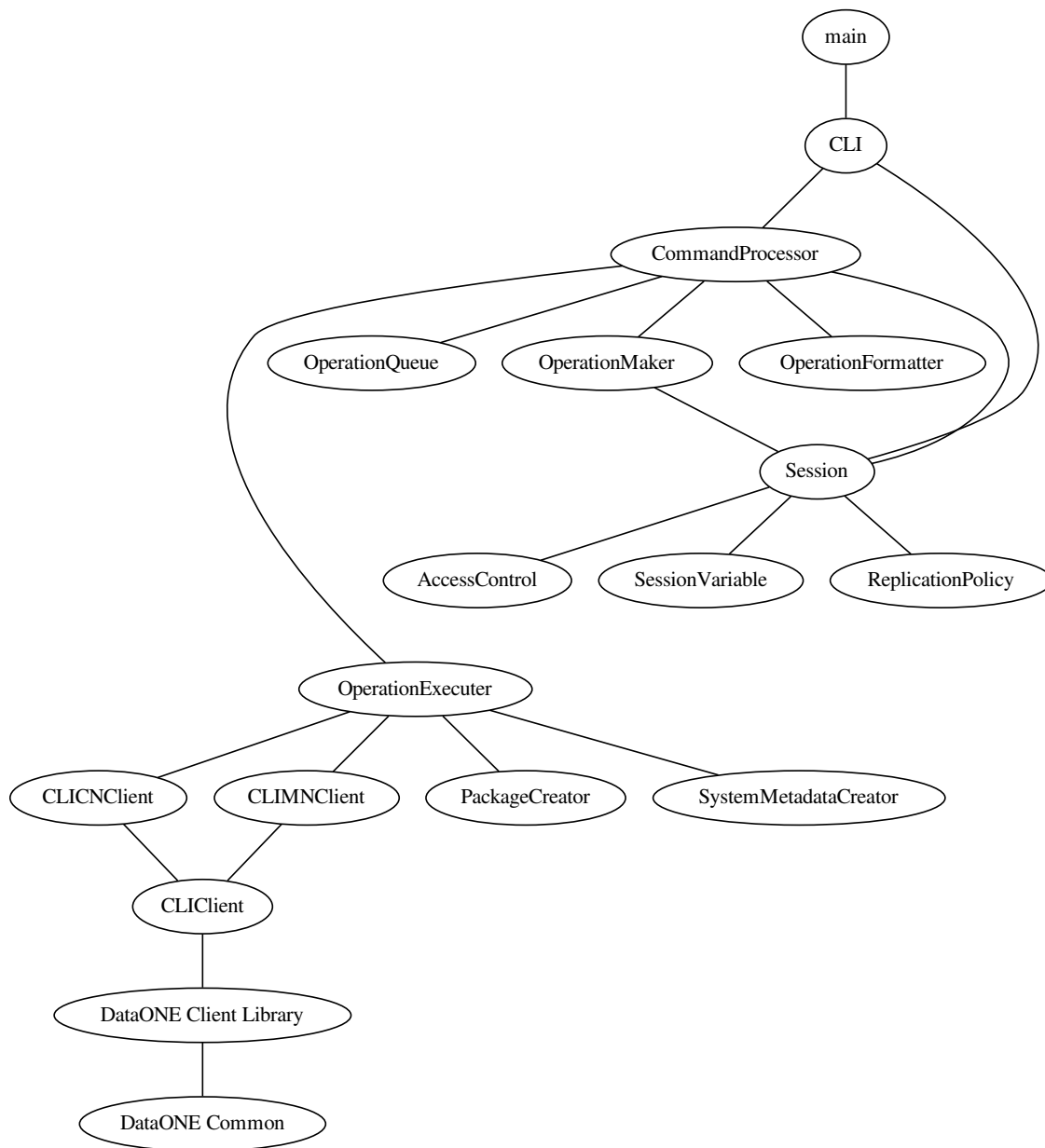
Name	Default	Type	Description
<b>CLI configuration</b>			
verbose	False	Boolean	Display more information editor nano String I
<b>Target Nodes</b>			
cn-url	<a href="https://cn.dataone.org/cn">https://cn.dataone.org/cn</a>	String	Node to which to connect for operations that a
mn-url	<a href="https://localhost/mn/">https://localhost/mn/</a>	String	Node to which to connect for operations that a
<b>Authentication</b>			
anonymous	True	Boolean	Ignore any installed certificates and connect a
key-file	None	String	Filesystem path to the client certificate private
<b>Slicing</b>			
start	0	Integer	First item to display for operations that display
count	1000	Integer	Maximum number of items to display for oper
<b>Searching</b>			
query	*.*	String	Query string (SOLR or Lucene query syntax)
query-type	solr	String	Select search engine (currently, only SOLR is
from-date	None	String	Start time used by operations that accept a tim
to-date	None	String	End time used by operations that accept a time
search-format-id	None	String	Include only objects of this format
<b>Parameters   Misc</b>			
algorithm	SHA-1	String	Checksum algorithm to use when calculating
format-id	None	String	ID for the Object Format to use when generati
<b>Parameters   Reference Nodes</b>			
authoritative-mn	None	String	Authoritative Member Node to use when gene
origin-mn	None	String	Originating Member Node to use when genera
<b>Parameters   Subjects</b>			
rights-holder	None	String	Subject of the rights holder to use when gene
submitter	None	String	Subject of the submitter to use when generati
<b>Access Control</b>			
Access Control Policy parameters managed by a <i>separate set of commands</i> .			
<b>Replication</b>			
Replication Policy parameters managed by a <i>separate set of commands</i> .			

## 4.2.4 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.

**4.2.5 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

```

### 4.2.6 Indices and tables

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

## 4.3 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:

### 4.3.1 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 gmnn 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
'
```

(continues on next page)

(continued from previous page)

```
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 gmh user account with password login disabled
id -u gmh 1>/dev/null 2>&1 || adduser --ingroup www-data \
    --gecos "DataONE Generic Member Node" --disabled-password gmh

ERR=$(sudo -u postgres createuser gmh 2>&1)
[[ ${ERR} =~ "already exists" ]] || echo ${ERR}
ERR=$(sudo -u postgres createdb -E UTF8 gmh3 2>&1)
[[ ${ERR} =~ "already exists" ]] || echo ${ERR}

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

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

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

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

# Allow the gmh user to read future Postgres and Apache logs
```

(continues on next page)

(continued from previous page)

```
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}/dl_gmn/deployment/bashrc ~/.bashrc
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
'
```

(continues on next page)

(continued from previous page)

```
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}/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
'
```

- 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 \
-out ca_cert.pem -infiles 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}/dl_gmn/settings.py
  cp ${GMN_PKG_DIR}/dl_gmn/settings_template.py ${SETTINGS_PATH}
  sed -Ei "s/MIDDLEWARE_CLASSES/MIDDLEWARE/" ${SETTINGS_PATH}
  sed -Ei "s/'gm2'/'gm2'/'gm3'/'gm3'/" ${SETTINGS_PATH}
  sed -Ei "s/(.*) (.*)my\\.server\\.name\\.com\\.*/\\1'${FQDN}'\\2'/" ${SETTINGS_PATH}
  python ${GMN_PKG_DIR}/dl_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 [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 [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"/>
    <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>
</ns1:node>
```

(continues on next page)

(continued from previous page)

```

</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 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
'

```

(continues on next page)

(continued from previous page)

```
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/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
```

### 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-contextlib2 \
python3-cryptography \
python3-django \
python3-idna \
python3-iso8601 \
python3-isodate \
python3-lxml \
python3-msgpack \
python3-pkg-resources \
python3-psycopg2 \
python3-pyasn1 \
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
```

(continues on next page)

(continued from previous page)

```
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 firewallld
$ sudo systemctl unmask firewallld
$ sudo systemctl start firewallld
```

#### 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 “*LOOP-BACK,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/dl_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 SSL 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 gm3-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 gmnn.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://centos7-3gmnn.kitty.ninja

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=gmnn.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/d1_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/dl_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 gmh:apache /var/local/dataone
$ sudo chmod -R g+w /var/local/dataone/
```

## Initialize the database

```
sudo -Hu gmh bash -c '
  cd /var/local/dataone
  source gmh_venv_py3/bin/activate
  python /var/local/dataone/gmh_venv_py3/lib/python3.6/site-packages/d1_gmh/
  ↪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"/>
    <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](mailto: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	<a href="https://cn.dataone.org/portal">https://cn.dataone.org/portal</a>
Staging	<a href="https://cn-stage.test.dataone.org/portal">https://cn-stage.test.dataone.org/portal</a>
Sandbox	<a href="https://cn-sandbox.test.dataone.org/portal">https://cn-sandbox.test.dataone.org/portal</a>
Development	<a href="https://cn-dev.test.dataone.org/portal">https://cn-dev.test.dataone.org/portal</a>

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](mailto: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/d1_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/d1_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](mailto: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-
→gm-test-ca.crt
$ sudo dpkg-reconfigure ca-certificates
$ sudo update-ca-certificates
```

In the dpkg-reconfigure GUI, enable the dataone-gm-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/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 a2disssite 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](#).

```
$ cd gmn
$ 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

$ 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
```

(continues on next page)

(continued from previous page)

```
$ 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(SSL error(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

```
SSL error(SSL error(1, '[SSL: TLSV1_ALERT_UNKNOWN_CA] tlsv1 alert unknown
ca (_ssl.c:2273)'), ssl.SSL error: [Errno 1] _ssl.c:510: error:14094418:SSL rou-
tines:SSL3_READ_BYTES:tlsv1 alert unknown ca SSL error(SSL error("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 `gmn3-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 = dl_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',) SSLError: _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 `gmh` distribution includes a `.bashrc` file that contains various convenient settings and aliases for the `gmh` 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 `gmh` user, which causes the `gmh` 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.

### 4.3.2 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.
  Active 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 `gmn` folder:

```
sudo -Hu gmn bash -c '
  cd /var/local/dataone/gmn_venv_py3/lib/python3.6/site-packages/gmn
  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.

### 4.3.3 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 *Test the installation on Ubuntu*, *Test the installation on CentOS* 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.

### 4.3.4 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.

### 4.3.5 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  
'
```

## 4.3.6 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 `gmh` user:

```
$ sudo -Hu gmh
$ ./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.

---

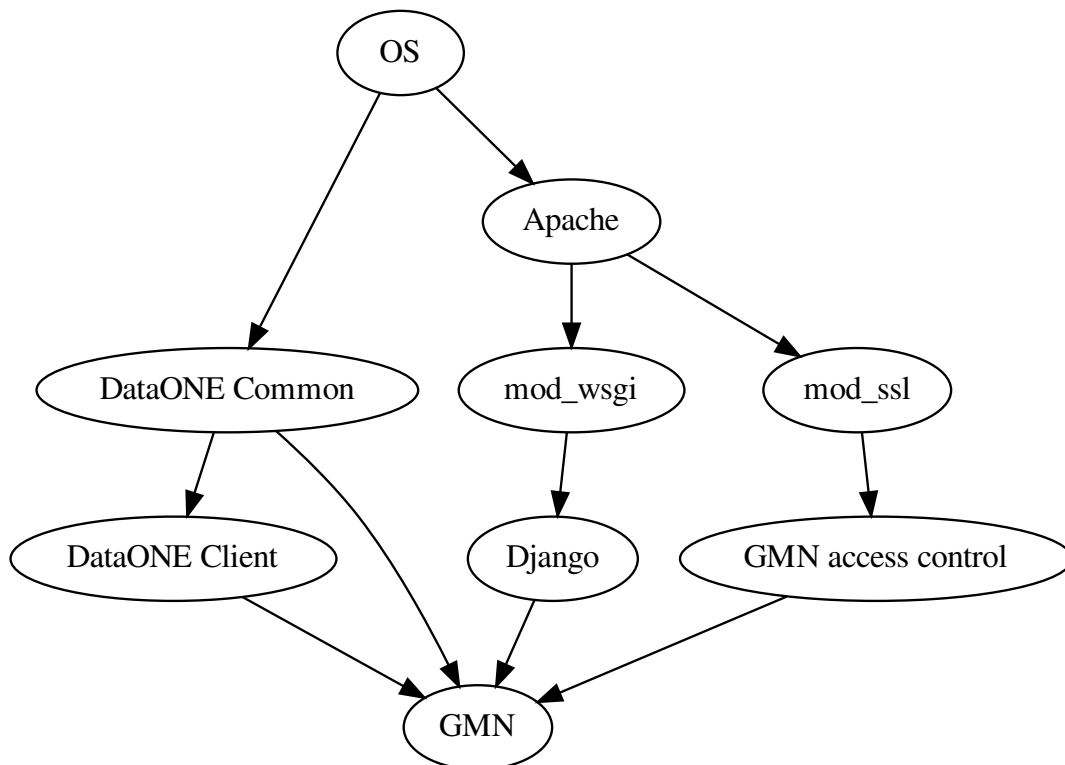
## 4.3.7 Implementation

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.

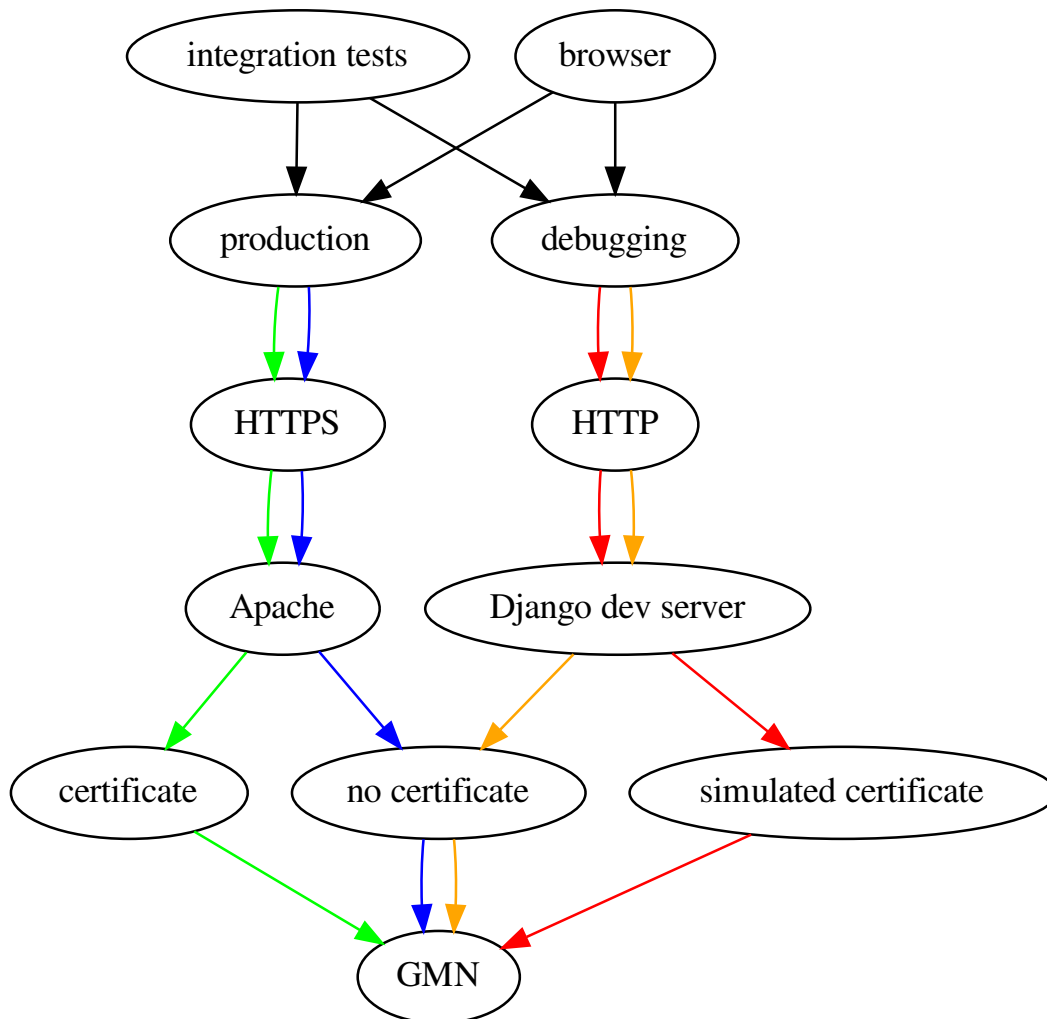


Figure: The various scenarios that GMN can be served under.

- 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 Indices and tables

- `genindex`
- `modindex`
- `search`

## 4.5 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:

### 4.5.1 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.

### 4.5.2 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
```

### 4.5.3 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
```

### 4.5.4 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
```

### 4.5.5 Unit Tests

This library is shipped with unit tests that verify correct operation. It is recommended that these are executed after installation.

### 4.5.6 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.

### 4.5.7 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

`d1_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

`d1_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

`d1_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`

`dl_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`

`dl_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`

`dl_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

`dl_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

`dl_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

`dl_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** `dl_common.cert.jwt.JwtException`

Bases: `Exception`

Exceptions raised directly by this module.

## `dl_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 identifies, 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.

`dl_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

`dl_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.

**node\_gen**

Generate all nodes for the tree rooted at this node.

Yields: SubjectInfoNode All nodes rooted at this node.

**leaf\_node\_gen**

Generate all leaf nodes for the tree rooted at this node.

Yields: SubjectInfoNode All leaf nodes rooted at this node.

**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

**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.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.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.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)`

Download public certificate from a TLS/SSL web server as DER encoded bytes.

*time-*

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',           time-
                                   out_sec=60.0)
```

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.log_cert_info(log, msg_str, cert_obj)`

Dump basic certificate values to the log.

**Parameters**

- **log** – 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_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.cert.x509.get_val_list(obj, path_list, reverse=False)`

Extract values from nested objects by attribute names.

Objects contain attributes which are named references to objects. This will descend down a tree of nested objects, starting at the given object, following the given path.

**Parameters**

- **obj** – object Any type of object
- **path\_list** – list Attribute names
- **reverse** – bool Reverse the list of values before concatenation.

**Returns** list of objects

`d1_common.cert.x509.get_val_str(obj, path_list=None, reverse=False)`

Extract values from nested objects by attribute names and concatenate their string representations.

**Parameters**

- **obj** – object Any type of object
- **path\_list** – list Attribute names
- **reverse** – bool Reverse the list of values before concatenation.

**Returns** Concatenated extracted values.

**Return type** str

`d1_common.cert.x509.get_ext_val_str(cert_obj, extension_name, path_list=None)`

Get value from certificate extension.

**Parameters**

- **cert\_obj** – cryptography.Certificate Certificate containing a standard extension.
- **extension\_name** – str Extension name. E.g., 'SUBJECT\_DIRECTORY\_ATTRIBUTES'.
- **path\_list** – list Attribute names

**Returns** String value of extension

**Return type** str

`d1_common.cert.x509.serialize_cert_to_pem(cert_obj)`  
Serialize certificate to PEM.

**Parameters** **cert\_obj** – cryptography.Certificate

**Returns** PEM encoded certificate

**Return type** bytes

`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.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.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 behaves 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.

**size**

Returns:

int: The total number of bytes that will be returned by the iterator.

## d1\_common.iter.file module

Generator that returns the bytes of a file in chunks.

```
class d1_common.iter.file.FileIterator (path, chunk_size=1024)
```

Bases: object

Generator that returns the bytes of a file in chunks.

**size**

Returns:

int : The total number of bytes that will be returned by the iterator.

```
class d1_common.iter.file.FileLikeObjectIterator (file, chunk_size=1024)
```

Bases: object

Generator that returns the bytes of a file-like object in chunks.

**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-  
                                include_glob_list=None,          recursive=True,          ig-  
                                nore_invalid=False,          default_excludes=True,          re-  
                                turn_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\_dir\_paths** – bool

- **False**: Only file paths are returned.

- **True**: Directory paths are also returned.

**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_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_dir_paths=True)
try:
    path = itr.next()
    while True:
        skip_dir = determine_if_dir_should_be_skipped(path)
        file_path = itr.send(skip_dir)
```

(continues on next page)

(continued from previous page)

```
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.

### 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.

**size**

Returns:

`int` : The total number of characters that will be returned by the iterator.

### 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, detailCode='0', description="", traceInformation=None, identifier=None, nodeId=None)`

Create a DataONE Exception object by errorCode.

See Also: For args, see: `DataONEException()`

```
exception dl_common.types.exceptions.DataONEException(errorCode, detailCode='0',
                                                         description="", traceInformation=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.

**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** `d1_common.types.exceptions.InvalidRequest` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type InvalidRequest.

See Also: `DataONEException()`

**exception** `d1_common.types.exceptions.InvalidSystemMetadata` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_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, trace-
                                              Information=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, de-
                                                              scription=None,
                                                              traceInforma-
                                                              tion=None, identi-
                                                              fier=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** `d1_common.types.exceptions.SynchronizationFailed` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type `SynchronizationFailed`.

See Also: `DataONEException()`

**exception** `d1_common.types.exceptions.VersionMismatch` (*detailCode*, *description=None*, *traceInformation=None*, *identifier=None*, *nodeId=None*)

Bases: `d1_common.types.exceptions.DataONEException`

DataONE Exception of type `VersionMismatch`.

See Also: `DataONEException()`

## d1\_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

### d1\_common.utils.filesystem module

Utilities for filesystem paths and operations.

`d1_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

`d1_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.

**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

`d1_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()`

`d1_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()`

`d1_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

`d1_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 dl_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.
- **level** – The level of severity to set for the progress log entries.
- **event\_counter** – Optional EventCounter to use for recording events
- **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*)

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
```

(continues on next page)

(continued from previous page)

```

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>

```

(continues on next page)

(continued from previous page)

```

<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',

```

(continues on next page)

(continued from previous page)

```
{
  '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, 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, 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** `d1_common.wrap.access_policy.AccessPolicyWrapper(access_pyxb)`

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 read perm.

Add 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 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.

`d1_common.wrap.simple_xml.wrap(xml_str)`  
Simple processing of XML.

`class d1_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)`

**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)`

**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=d1_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** `d1_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-v2.0.1/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

`d1_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.

`d1_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()`.

`d1_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`

`d1_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.

`d1_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`

`d1_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** `dl_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** `dl_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.

`dl_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

`dl_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.

`dl_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

`dl_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`.

`dl_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.

`dl_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.

`dl_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>

`d1_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`.

`d1_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>

`d1_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.

```
d1_common.date_time.utcnow()
```

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'`.

`d1_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.

`d1_common.date_time.local_now()`

Returns:

tz-aware datetime : The current local date and time in the local timezone

`d1_common.date_time.local_now_iso()`

Returns:

**str** : The current local date and time as an ISO 8601 string in the local timezone

`d1_common.date_time.to_iso8601_utc(dt)`

Args: dt: datetime.

Returns: str: ISO 8601 string in the UTC timezone

`d1_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

`d1_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,
    'replicate': 0,
    'state': u'up',
    'synchronize': 1,
    'type': u'mn'
  },
}
```

### d1\_common.object\_format module

### 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# >
rdfs: <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`.

---

`d1_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*

`d1_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 3.4.0', *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 3.4.0', *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 3.4.0')
```

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 `rd/xml`. Format support can be extended with plugins.  
Built-in: `xml, n3, nt, trix, rdfa`
- **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\_pymb*)

Get set of identifiers that provide revision context for SciObj.

Returns: tuple: PID, SID, OBSOLETE\_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 identified 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, OBSOLETE\_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>
```

(continues on next page)

(continued from previous page)

```

    <!--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>

<!--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>

```

`d1_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

`d1_common.system_metadata.normalize_in_place` (*sysmeta\_pyxb*, *reset\_timestamps=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.

```
d1_common.system_metadata.are_equivalent_pyxb(a_pyxb, b_pyxb, ignore_timestamps=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
- **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.

**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.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., `d1_common.types.dataoneTypes_v1_1`.

**Return type** PyXB binding

`d1_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

`d1_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

`d1_common.type_conversions.is_pyxb(pyxb_obj)`

Returns:

bool: **True** if `pyxb_obj` is a PyXB object.

`d1_common.type_conversions.is_pyxb_d1_type(pyxb_obj)`

Returns:

bool: **True** if `pyxb_obj` is a PyXB object holding a DataONE API type.

`d1_common.type_conversions.is_pyxb_d1_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

`d1_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

`d1_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

`dl_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

`dl_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** API XML doc matching the API version of `etree_obj`.

**Return type** str

`dl_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

`dl_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

`dl_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

`dl_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 is 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.

`dl_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
```

`dl_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** `dl_common.util.EventCounter`

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
```

**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.

**dl\_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.

**dl\_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 datetimes 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()`

**dl\_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** `d1_common.util.ToJsonCompatibleTypes` (\*, `skipkeys=False`, `ensure_ascii=True`,  
`check_circular=True`, `allow_nan=True`,  
`sort_keys=False`, `indent=None`, `separators=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)
```

`d1_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`

## d1\_common.xml module

Utilities for handling XML docs.

`d1_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_d1_exception()` for deserializing DataONE Exception types.

`d1_common.xml.deserialize_d1_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

`d1_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.

`d1_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

`d1_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.

## 4.6 Indices and tables

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

## 4.7 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:

### 4.7.1 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
```

### 4.7.2 Unit Tests

This library is shipped with unit tests that verify correct operation. It is recommended that these are executed after installation.

### 4.7.3 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
```

### 4.7.4 API

#### d1\_client package

DataONE Client Library.

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 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.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.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 d1_client.client
import sys

logging.basicConfig(level=logging.INFO)
target = "https://mn-unm-1.dataone.org/mn"
client = d1_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()
    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, *fromDate*, *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,  
                                                             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](https://releases.dataone.org/online/api-documentation-v2.0/apis/Types.html#Types.Node)

```
class d1_client.iter.node.NodeListIterator (base_url,                                api_major=2,
                                             client_arg_dict=None, listNodes_dict=None)
    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 dlbaseclient
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.
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 "--"
```

(continues on next page)

(continued from previous page)

```

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 d1_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,
                                                             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,
                                                                  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
```

## Submodules

### d1\_client.baseclient module

**class** d1\_client.baseclient.**DataONEBaseClient** (*base\_url*, \**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/MN_APIs.html) [https://releases.dataone.org/online/api-documentation-v2.0/apis/CN\\_APIs.html](https://releases.dataone.org/online/api-documentation-v2.0/apis/CN_APIs.html)

On error response, raises a DataONEException.

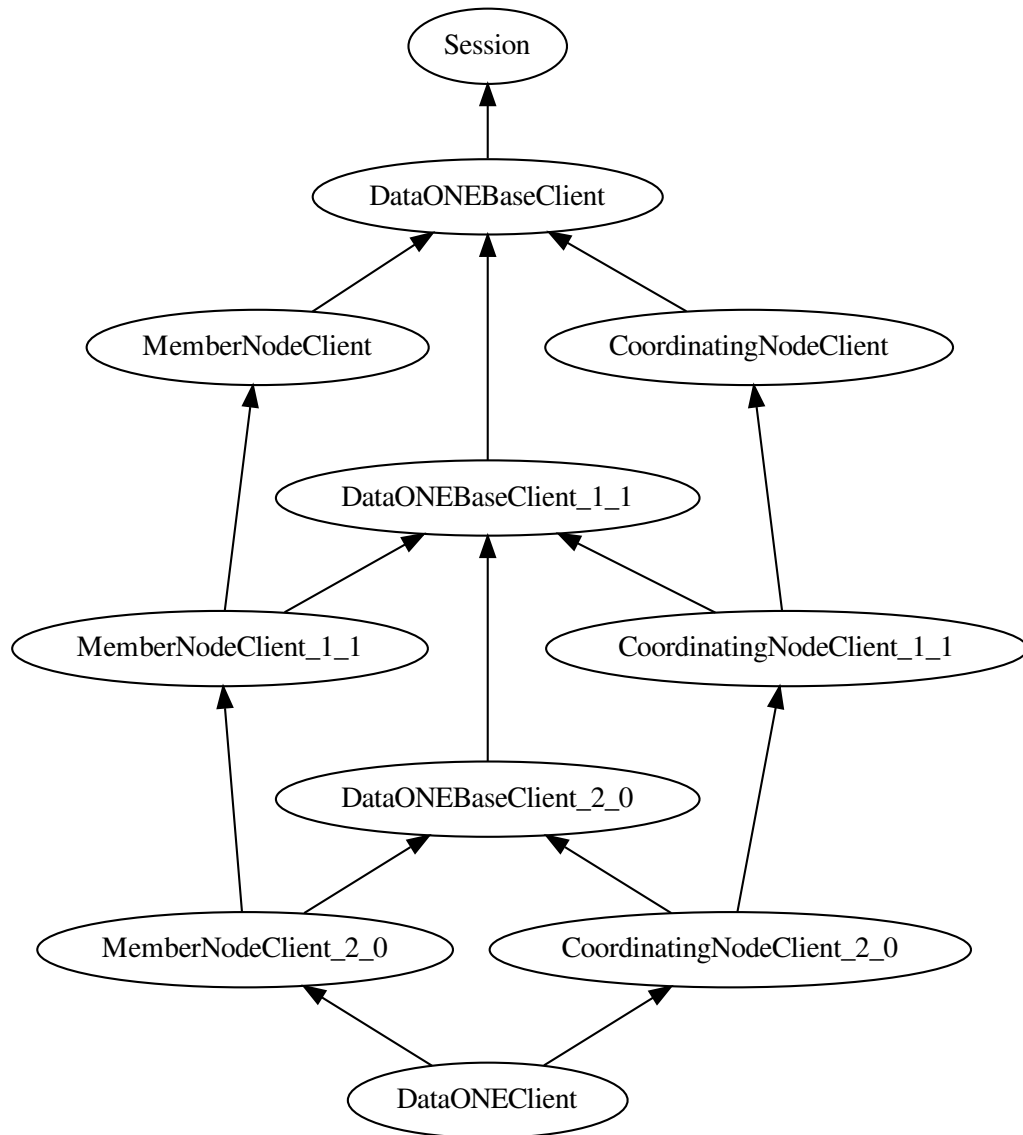
Methods with names that end in “Response” return the HTTPResponse object directly for manual processing by the client. The \*Response methods are only needed in rare cases where the default handling is inadequate, e.g., for dealing with nodes that don’t fully comply with the spec.

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*, \*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

**api\_version\_tup**

**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\_path*, *create\_missing\_dirs=False*, *vendorSpecific=None*)

Like `MNRead.get()`, but also retrieve the object bytes and store them in a local file at `sciobj_path`. 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/MN_APIs.html)    [https://releases.dataone.org/online/api-documentation-v2.0/apis/CN\\_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](https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN_APIs.html#CNRead.query) `MNQuery.query(session, queryEngine, query) → OctetStream` [http://jenkins-](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](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*, \*\*kwargs)

`CNRead.getQueryEngineDescription(session, queryEngine) → QueryEngineDescription` [https://releases.dataone.org/online/api-documentation-v2.0.1/apis/CN\\_APIs.html#CNRead.getQueryEngineDescription](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-](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](/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/MN_APIs.html)    [https://releases.dataone.org/online/api-documentation-v2.0/apis/CN\\_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/MN_APIs.html)    [https://releases.dataone.org/online/api-documentation-v2.0/apis/CN\\_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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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-](https://releases.dataone.org/online/api-v2.0.1/apis/CN_APIs.html#CNReplication.isNodeAuthorized) documentation-  
 v2.0.1/apis/CN\_APIs.html#CNReplication.isNodeAuthorized.

#### 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](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](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](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](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](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](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)
view (theme, did)
listViewsResponse ()
listViews ()
echoCredentialsResponse ()
echoCredentials ()
echoSystemMetadataResponse (sysmeta_pyxb)
echoSystemMetadata (sysmeta_pyxb)
echoIndexedObjectResponse (queryEngine, sysmeta_pyxb, obj)
echoIndexedObject (queryEngine, sysmeta_pyxb, obj)
```

### **d1\_client.d1client module**

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.

```
class d1_client.d1client.DataONEClient
```

Bases: `object`

```
d1_client.d1client.get_api_major_by_base_url (base_url, *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](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, **kwargs)

```

```
view (theme, did, **kwargs)  
listViewsResponse (**kwargs)  
listViews (**kwargs)  
getPackageResponse (did, packageType='application/bagit-097', **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.object\_format\_info module**

### **d1\_client.session module**

```
class d1_client.session.Session (base_url, cert_pem_path=None, cert_key_path=None, **kwargs_dict)  
  
    Bases: object  
  
    __init__ (base_url, 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.
- **retries** (*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)

**Returns** None

#### **base\_url**

**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
    '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.SolrClient` (*\*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:

`solr_client = SolrClient('https://cn.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='id:abc*', fq='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='id:abc*', fq=['id:def*', 'id:ghi'])
```

- Do not encode the query parameters before passing them to the methods.

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,
transformer=<d1_client.solr_client.SolrRecordTransformer
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)`

## 4.8 Indices and tables

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

## 4.9 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:

### 4.9.1 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
```

### 4.9.2 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

#### 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)
```

```
d1_test.instance_generator.random_data.random_lower_ascii(min_len=2,
                                                           max_len=2)
```

```
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,
                                              supplemen-
                                              tal_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**

#### **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()

```

Note: for `get()`, GMN returns a `StreamingHttpResponse` that Requests detects as a streaming response and handles accordingly. However, when returning a `StreamingHttpResponse` from Responses, no special handling occurs. This breaks test code that converts streams to strings by accessing `.content` (production code should not do this since it causes the entire stream to be buffered in memory). So we convert streaming responses to string before passing them to Responses.

```

d1_test.mock_api.django_client.add_callback(base_url)

```

## 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](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](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](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](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**



```
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.utilities package**

### **Submodules**

## **d1\_test.utilities.catch\_multiprocessing\_exception module**

## **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, in-
                                                clude_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,
                                                                digest='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=True,  
                                                                              include_revision_bootstrap=False,  
                                                                              use_v1_bootstrap=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_json(filename)`

`d1_test.test_files.load_cert(filename)`

`d1_test.test_files.load_jwt(filename)`

## **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`

### 4.9.3 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.

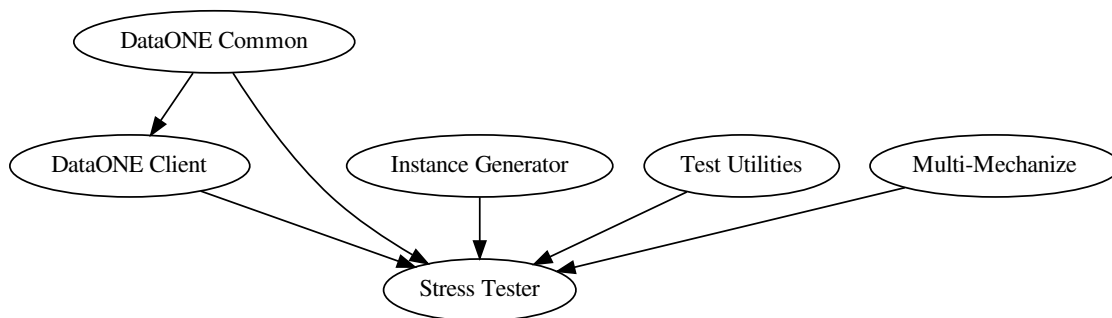
Contents:

#### Indices and tables

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

### 4.9.4 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 generated 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 is 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 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 of 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`

### 4.9.5 DataONE Test Utilities

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

### 4.9.6 Indices and tables

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### d

- `dl_client`, 188
- `dl_client.baseclient`, 194
- `dl_client.baseclient_1_1`, 197
- `dl_client.baseclient_1_2`, 199
- `dl_client.baseclient_2_0`, 199
- `dl_client.cnclient`, 199
- `dl_client.cnclient_1_1`, 212
- `dl_client.cnclient_1_2`, 213
- `dl_client.cnclient_2_0`, 213
- `dl_client.dlclient`, 214
- `dl_client.iter`, 188
- `dl_client.iter.logrecord`, 189
- `dl_client.iter.logrecord_multi`, 190
- `dl_client.iter.node`, 191
- `dl_client.iter.objectlist`, 191
- `dl_client.iter.objectlist_multi`, 193
- `dl_client.iter.sysmeta_multi`, 193
- `dl_client.mnclient`, 214
- `dl_client.mnclient_1_1`, 215
- `dl_client.mnclient_1_2`, 215
- `dl_client.mnclient_2_0`, 216
- `dl_client.session`, 216
- `dl_client.solr_client`, 218
- `dl_client.util`, 221
- `dl_common`, 112
- `dl_common.bagit`, 149
- `dl_common.cert`, 112
- `dl_common.cert.jwt`, 112
- `dl_common.cert.subject_info`, 115
- `dl_common.cert.subjects`, 121
- `dl_common.cert.x509`, 121
- `dl_common.checksum`, 150
- `dl_common.const`, 152
- `dl_common.date_time`, 152
- `dl_common.env`, 158
- `dl_common.ext`, 125
- `dl_common.ext.mimeparser`, 125
- `dl_common.iter`, 126
- `dl_common.iter.bytes`, 126
- `dl_common.iter.file`, 127
- `dl_common.iter.path`, 127
- `dl_common.iter.string`, 129
- `dl_common.logging_context`, 159
- `dl_common.multipart`, 159
- `dl_common.node`, 160
- `dl_common.replication_policy`, 160
- `dl_common.resource_map`, 164
- `dl_common.revision`, 169
- `dl_common.system_metadata`, 170
- `dl_common.type_conversions`, 173
- `dl_common.types`, 129
- `dl_common.types.dataoneErrors`, 130
- `dl_common.types.dataoneTypes`, 130
- `dl_common.types.dataoneTypes_v1`, 130
- `dl_common.types.dataoneTypes_v1_1`, 130
- `dl_common.types.dataoneTypes_v1_2`, 130
- `dl_common.types.dataoneTypes_v2_0`, 130
- `dl_common.types.exceptions`, 130
- `dl_common.url`, 178
- `dl_common.util`, 179
- `dl_common.utils`, 135
- `dl_common.utils.filesystem`, 135
- `dl_common.utils.progress_logger`, 136
- `dl_common.wrap`, 138
- `dl_common.wrap.access_policy`, 138
- `dl_common.wrap.simple_xml`, 145
- `dl_common.xml`, 183
- `dl_test`, 221
- `dl_test.instance_generator`, 222
- `dl_test.instance_generator.checksum`, 222
- `dl_test.instance_generator.format_id`, 222
- `dl_test.instance_generator.identifier`, 222
- `dl_test.instance_generator.media_type`, 223
- `dl_test.instance_generator.names`, 223
- `dl_test.instance_generator.node_ref`, 223
- `dl_test.instance_generator.person`, 223
- `dl_test.instance_generator.random_data`, 223
- `dl_test.instance_generator.replication_policy`, 225

`dl_test.instance_generator.replication_status,`  
    [225](#)  
`dl_test.instance_generator.subject,` [225](#)  
`dl_test.instance_generator.unicode_names,`  
    [226](#)  
`dl_test.instance_generator.unicode_test_strings,`  
    [226](#)  
`dl_test.instance_generator.user_agent,`  
    [226](#)  
`dl_test.instance_generator.words,` [226](#)  
`dl_test.mock_api,` [226](#)  
`dl_test.mock_api.create,` [226](#)  
`dl_test.mock_api.dl_exception,` [227](#)  
`dl_test.mock_api.django_client,` [227](#)  
`dl_test.mock_api.generate_identifier,`  
    [227](#)  
`dl_test.mock_api.ping,` [228](#)  
`dl_test.mock_api.post,` [228](#)  
`dl_test.replication_tester,` [229](#)  
`dl_test.replication_tester.replication_error,`  
    [229](#)  
`dl_test.replication_tester.replication_server,`  
    [229](#)  
`dl_test.replication_tester.replication_tester,`  
    [229](#)  
`dl_test.replication_tester.test_object_generator,`  
    [230](#)  
`dl_test.slender_node_test_client,` [233](#)  
`dl_test.test_files,` [234](#)  
`dl_test.utilities,` [230](#)  
`dl_test.utilities.create_from_file,` [230](#)  
`dl_test.utilities.generate_test_subject_certs,`  
    [230](#)  
`dl_test.utilities.my_subject,` [232](#)  
`dl_test.utilities.pem_in_http_header,`  
    [232](#)  
`dl_test.utilities.populate_mn,` [232](#)  
`dl_test.utilities.test_object_generator,`  
    [232](#)  
`dl_test.utilities.urlencode,` [233](#)  
`dl_test.xml_normalize,` [234](#)

## Symbols

<code>__init__()</code> ( <i>d1_client.baseclient.DataONEBaseClient</i> method), 196	<code>__init__()</code> ( <i>d1_test.slender_node_test_client.SlenderNodeTestClient</i> method), 233
<code>__init__()</code> ( <i>d1_client.baseclient_1_1.DataONEBaseClient_1_1</i> method), 198	<code>__init__()</code> ( <i>d1_test.xml_normalize.StableNode</i> method), 235
<code>__init__()</code> ( <i>d1_client.baseclient_1_2.DataONEBaseClient_1_2</i> method), 199	
<code>__init__()</code> ( <i>d1_client.baseclient_2_0.DataONEBaseClient_2_0</i> method), 199	
<code>__init__()</code> ( <i>d1_client.cnclient.CoordinatingNodeClient</i> method), 200	
<code>__init__()</code> ( <i>d1_client.cnclient_1_1.CoordinatingNodeClient_1_1</i> method), 212	
<code>__init__()</code> ( <i>d1_client.cnclient_1_2.CoordinatingNodeClient_1_2</i> method), 213	
<code>__init__()</code> ( <i>d1_client.cnclient_2_0.CoordinatingNodeClient_2_0</i> method), 213	
<code>__init__()</code> ( <i>d1_client.iter.logrecord.LogRecordIterator</i> method), 189	
<code>__init__()</code> ( <i>d1_client.iter.objectlist.ObjectListIterator</i> method), 192	
<code>__init__()</code> ( <i>d1_client.mnclient.MemberNodeClient</i> method), 214	
<code>__init__()</code> ( <i>d1_client.mnclient_1_1.MemberNodeClient_1_1</i> method), 215	
<code>__init__()</code> ( <i>d1_client.mnclient_1_2.MemberNodeClient_1_2</i> method), 215	
<code>__init__()</code> ( <i>d1_client.mnclient_2_0.MemberNodeClient_2_0</i> method), 216	
<code>__init__()</code> ( <i>d1_client.session.Session</i> method), 216	
<code>__init__()</code> ( <i>d1_client.solr_client.SolrValuesResponseIterator</i> method), 220	
<code>__init__()</code> ( <i>d1_common.date_time.FixedOffset</i> method), 153	
<code>__init__()</code> ( <i>d1_common.logging_context.LoggingContext</i> method), 159	
<code>__init__()</code> ( <i>d1_common.resource_map.ResourceMap</i> method), 165	
<code>__init__()</code> ( <i>d1_common.types.exceptions.DataONEException</i> method), 132	
<code>__init__()</code> ( <i>d1_common.utils.progress_logger.ProgressLogger</i> method), 137	
	<b>A</b>
	<code>abs_path()</code> (in module <i>d1_common.utils.filesystem</i> ), 136
	<code>abs_path_from_base()</code> (in module <i>d1_common.utils.filesystem</i> ), 136
	<code>AccessPolicyWrapper</code> (class in <i>d1_common.wrap.access_policy</i> ), 141
	<code>add()</code> ( <i>d1_client.solr_client.SolrClient</i> method), 219
	<code>add_authenticated_read()</code> ( <i>d1_common.wrap.access_policy.AccessPolicyWrapper</i> method), 143
	<code>add_authenticated_read()</code> (in module <i>d1_common.wrap.access_policy</i> ), 144
	<code>add_blocked()</code> (in module <i>d1_common.replication_policy</i> ), 163
	<code>add_callback()</code> (in module <i>d1_test.mock_api.create</i> ), 226
	<code>add_callback()</code> (in module <i>d1_test.mock_api.django_client</i> ), 227
	<code>add_callback()</code> (in module <i>d1_test.mock_api.generate_idenfier</i> ), 228
	<code>add_callback()</code> (in module <i>d1_test.mock_api.ping</i> ), 228
	<code>add_callback()</code> (in module <i>d1_test.mock_api.post</i> ), 228
	<code>add_child()</code> ( <i>d1_common.cert.subject_info.SubjectInfoNode</i> method), 119
	<code>add_child()</code> ( <i>d1_test.xml_normalize.StableNode</i> method), 235
	<code>add_docs()</code> ( <i>d1_client.solr_client.SolrClient</i> method), 219
	<code>add_perm()</code> ( <i>d1_common.wrap.access_policy.AccessPolicyWrapper</i> method), 143
	<code>add_perm()</code> (in module <i>d1_common.wrap.access_policy</i> ), 144
	<code>add_preferred()</code> (in module <i>d1_common.replication_policy</i> ), 163

add\_public\_read() (in module *d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 143  
 add\_public\_read() (in module *d1\_common.wrap.access\_policy*), 144  
 add\_verified\_read() (in module *d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 143  
 add\_verified\_read() (in module *d1\_common.wrap.access\_policy*), 144  
 addDataDocuments() (in module *d1\_common.resource\_map.ResourceMap* method), 167  
 addMetadataDocument() (in module *d1\_common.resource\_map.ResourceMap* method), 167  
 addResource() (in module *d1\_common.resource\_map.ResourceMap* method), 166  
 Apache, 35  
 api\_version\_tuple(*d1\_client.baseclient.DataONEBaseClient* attribute), 196  
 archive() (in module *d1\_client.baseclient.DataONEBaseClient* method), 197  
 archiveResponse() (in module *d1\_client.baseclient.DataONEBaseClient* method), 197  
 are\_checksums\_equal() (in module *d1\_common.checksum*), 151  
 are\_equal() (in module *d1\_common.date\_time*), 154  
 are\_equal\_elements() (in module *d1\_common.xml*), 185  
 are\_equal\_or\_superset() (in module *d1\_common.xml*), 184  
 are\_equal\_pyxb() (in module *d1\_common.xml*), 185  
 are\_equal\_xml() (in module *d1\_common.xml*), 184  
 are\_equivalent() (in module *d1\_common.xml*), 184  
 are\_equivalent\_pyxb() (in module *d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
 are\_equivalent\_pyxb() (in module *d1\_common.replication\_policy*), 162  
 are\_equivalent\_pyxb() (in module *d1\_common.system\_metadata*), 172  
 are\_equivalent\_pyxb() (in module *d1\_common.wrap.access\_policy*), 144  
 are\_equivalent\_pyxb() (in module *d1\_common.xml*), 184  
 are\_equivalent\_xml() (in module *d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 143  
 are\_equivalent\_xml() (in module *d1\_common.replication\_policy*), 162  
 are\_equivalent\_xml() (in module *d1\_common.system\_metadata*), 172  
 are\_equivalent\_xml() (in module *d1\_common.wrap.access\_policy*), 144  
 asGraphvizDot() (in module *d1\_common.resource\_map.ResourceMap* method), 169  
 authenticationTimeout, 133

## B

base\_url(*d1\_client.session.Session* attribute), 217  
 Bash, 35  
 best\_match() (in module *d1\_common.ext.mimeparser*), 126  
 BytesIterator (class in *d1\_common.iter.bytes*), 126

## C

CA, 33  
 CA certificate, 33  
 CA signing key, 33  
 calculate\_checksum\_on\_bytes() (in module *d1\_common.checksum*), 151  
 calculate\_checksum\_on\_iterator() (in module *d1\_common.checksum*), 151  
 calculate\_checksum\_on\_stream() (in module *d1\_common.checksum*), 150  
 cast\_naive\_datetime\_to\_tz() (in module *d1\_common.date\_time*), 157  
 Certificate, 33  
 Chain of trust, 34  
 CILogon, 36  
 clear() (in module *d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 143  
 clear() (in module *d1\_common.wrap.access\_policy*), 144  
 clear\_elements() (in module *d1\_common.system\_metadata*), 172  
 client, 33  
 Client side authentication, 34  
 Client side certificate, 34  
 client, 33  
 commit() (in module *d1\_client.solr\_client.SolrClient* method), 219  
 CompareError, 187  
 completed() (in module *d1\_common.utils.progress\_logger.ProgressLogger* method), 138  
 confirmMapIdentity() (in module *d1\_client.cnclient.CoordinatingNodeClient* method), 208  
 confirmMapIdentityResponse() (in module *d1\_client.cnclient.CoordinatingNodeClient* method), 208  
 CoordinatingNodeClient (class in *d1\_client.cnclient*), 199

[CoordinatingNodeClient\\_1\\_1](#) (class in [createResourceMapFromStream\(\)](#) (in module [d1\\_client.cnclient\\_1\\_1](#)), 212  
[d1\\_common.resource\\_map](#)), 164  
[CoordinatingNodeClient\\_1\\_2](#) (class in [createResponse\(\)](#) ([d1\\_client.mnclient.MemberNodeClient](#) method), 215  
[d1\\_client.cnclient\\_1\\_2](#)), 213  
[CoordinatingNodeClient\\_2\\_0](#) (class in [createSimpleResourceMap\(\)](#) (in module [d1\\_client.cnclient\\_2\\_0](#)), 213  
[d1\\_common.resource\\_map](#)), 164  
[count\(\)](#) ([d1\\_client.solr\\_client.SolrClient](#) method), 219  
[count\(\)](#) ([d1\\_common.util.EventCounter](#) method), 180  
[create\(\)](#) ([d1\\_client.mnclient.MemberNodeClient](#) method), 215  
[create\(\)](#) ([d1\\_test.slender\\_node\\_test\\_client.SlenderNodeTestClient](#) method), 233  
[create\\_bagit\\_stream\(\)](#) (in module [d1\\_common.bagit](#)), 149  
[create\\_cert\\_request\(\)](#) (in module [d1\\_test.utilities.generate\\_test\\_subject\\_certs](#)), 231  
[create\\_certificate\(\)](#) (in module [d1\\_test.utilities.generate\\_test\\_subject\\_certs](#)), 231  
[create\\_checksum\\_object\\_from\\_bytes\(\)](#) (in module [d1\\_common.checksum](#)), 150  
[create\\_checksum\\_object\\_from\\_iterator\(\)](#) (in module [d1\\_common.checksum](#)), 150  
[create\\_checksum\\_object\\_from\\_stream\(\)](#) (in module [d1\\_common.checksum](#)), 150  
[create\\_exception\\_by\\_error\\_code\(\)](#) (in module [d1\\_common.types.exceptions](#)), 131  
[create\\_exception\\_by\\_name\(\)](#) (in module [d1\\_common.types.exceptions](#)), 131  
[create\\_header\\_d1\\_exception\(\)](#) (in module [d1\\_test.mock\\_api.d1\\_exception](#)), 227  
[create\\_key\\_pair\(\)](#) (in module [d1\\_test.utilities.generate\\_test\\_subject\\_certs](#)), 230  
[create\\_missing\\_directories\\_for\\_dir\(\)](#) (in module [d1\\_common.utils.filesystem](#)), 136  
[create\\_missing\\_directories\\_for\\_file\(\)](#) (in module [d1\\_common.utils.filesystem](#)), 135  
[create\\_regular\\_d1\\_exception\(\)](#) (in module [d1\\_test.mock\\_api.d1\\_exception](#)), 227  
[create\\_session\\_extension\(\)](#) (in module [d1\\_test.utilities.generate\\_test\\_subject\\_certs](#)), 231  
[create\\_test\\_object\\_on\\_mn\(\)](#) (in module [d1\\_test.replication\\_tester.replication\\_tester](#)), 229  
[create\\_utc\\_datetime\(\)](#) (in module [d1\\_common.date\\_time](#)), 158  
[createGroup\(\)](#) ([d1\\_client.cnclient.CoordinatingNodeClient](#) method), 209  
[createGroupResponse\(\)](#) ([d1\\_client.cnclient.CoordinatingNodeClient](#) method), 208  
[createResourceMapFromStream\(\)](#) (in module [d1\\_common.resource\\_map](#)), 164  
[createResponse\(\)](#) ([d1\\_client.mnclient.MemberNodeClient](#) method), 215  
[createSimpleResourceMap\(\)](#) (in module [d1\\_common.resource\\_map](#)), 164  
[cron](#), 36  
[CSR](#), 33

## D

[d1\\_client](#) (module), 188  
[d1\\_client.baseclient](#) (module), 194  
[d1\\_client.baseclient\\_1\\_1](#) (module), 197  
[d1\\_client.baseclient\\_1\\_2](#) (module), 199  
[d1\\_client.baseclient\\_2\\_0](#) (module), 199  
[d1\\_client.cnclient](#) (module), 199  
[d1\\_client.cnclient\\_1\\_1](#) (module), 212  
[d1\\_client.cnclient\\_1\\_2](#) (module), 213  
[d1\\_client.cnclient\\_2\\_0](#) (module), 213  
[d1\\_client.d1client](#) (module), 214  
[d1\\_client.iter](#) (module), 188  
[d1\\_client.iter.logrecord](#) (module), 189  
[d1\\_client.iter.logrecord\\_multi](#) (module), 190  
[d1\\_client.iter.node](#) (module), 191  
[d1\\_client.iter.objectlist](#) (module), 191  
[d1\\_client.iter.objectlist\\_multi](#) (module), 193  
[d1\\_client.iter.sysmeta\\_multi](#) (module), 193  
[d1\\_client.mnclient](#) (module), 214  
[d1\\_client.mnclient\\_1\\_1](#) (module), 215  
[d1\\_client.mnclient\\_1\\_2](#) (module), 215  
[d1\\_client.mnclient\\_2\\_0](#) (module), 216  
[d1\\_client.session](#) (module), 216  
[d1\\_client.solr\\_client](#) (module), 218  
[d1\\_client.util](#) (module), 221  
[d1\\_common](#) (module), 112  
[d1\\_common.bagit](#) (module), 149  
[d1\\_common.cert](#) (module), 112  
[d1\\_common.cert.jwt](#) (module), 112  
[d1\\_common.cert.subject\\_info](#) (module), 115  
[d1\\_common.cert.subjects](#) (module), 121  
[d1\\_common.cert.x509](#) (module), 121  
[d1\\_common.checksum](#) (module), 150  
[d1\\_common.const](#) (module), 152  
[d1\\_common.date\\_time](#) (module), 152  
[d1\\_common.env](#) (module), 158  
[d1\\_common.ext](#) (module), 125  
[d1\\_common.ext.mimeparser](#) (module), 125  
[d1\\_common.iter](#) (module), 126  
[d1\\_common.iter.bytes](#) (module), 126  
[d1\\_common.iter.file](#) (module), 127  
[d1\\_common.iter.path](#) (module), 127  
[d1\\_common.iter.string](#) (module), 129

`dl_common.logging_context (module)`, 159  
`dl_common.multipart (module)`, 159  
`dl_common.node (module)`, 160  
`dl_common.replication_policy (module)`, 160  
`dl_common.resource_map (module)`, 164  
`dl_common.revision (module)`, 169  
`dl_common.system_metadata (module)`, 170  
`dl_common.type_conversions (module)`, 173  
`dl_common.types (module)`, 129  
`dl_common.types.dataoneErrors (module)`, 130  
`dl_common.types.dataoneTypes (module)`, 130  
`dl_common.types.dataoneTypes_v1 (module)`, 130  
`dl_common.types.dataoneTypes_v1_1 (module)`, 130  
`dl_common.types.dataoneTypes_v1_2 (module)`, 130  
`dl_common.types.dataoneTypes_v2_0 (module)`, 130  
`dl_common.types.exceptions (module)`, 130  
`dl_common.url (module)`, 178  
`dl_common.util (module)`, 179  
`dl_common.utils (module)`, 135  
`dl_common.utils.filesystem (module)`, 135  
`dl_common.utils.progress_logger (module)`, 136  
`dl_common.wrap (module)`, 138  
`dl_common.wrap.access_policy (module)`, 138  
`dl_common.wrap.simple_xml (module)`, 145  
`dl_common.xml (module)`, 183  
`dl_test (module)`, 221  
`dl_test.instance_generator (module)`, 222  
`dl_test.instance_generator.checksum (module)`, 222  
`dl_test.instance_generator.format_id (module)`, 222  
`dl_test.instance_generator.identifier (module)`, 222  
`dl_test.instance_generator.media_type (module)`, 223  
`dl_test.instance_generator.names (module)`, 223  
`dl_test.instance_generator.node_ref (module)`, 223  
`dl_test.instance_generator.person (module)`, 223  
`dl_test.instance_generator.random_data (module)`, 223  
`dl_test.instance_generator.replication_policy (module)`, 225  
`dl_test.instance_generator.replication_status (module)`, 225  
`dl_test.instance_generator.subject (module)`, 225  
`dl_test.instance_generator.unicode_names (module)`, 226  
`dl_test.instance_generator.unicode_test_strings (module)`, 226  
`dl_test.instance_generator.user_agent (module)`, 226  
`dl_test.instance_generator.words (module)`, 226  
`dl_test.mock_api (module)`, 226  
`dl_test.mock_api.create (module)`, 226  
`dl_test.mock_api.dl_exception (module)`, 227  
`dl_test.mock_api.django_client (module)`, 227  
`dl_test.mock_api.generate_identifier (module)`, 227  
`dl_test.mock_api.ping (module)`, 228  
`dl_test.mock_api.post (module)`, 228  
`dl_test.replication_tester (module)`, 229  
`dl_test.replication_tester.replication_error (module)`, 229  
`dl_test.replication_tester.replication_server (module)`, 229  
`dl_test.replication_tester.replication_tester (module)`, 229  
`dl_test.replication_tester.test_object_generator (module)`, 230  
`dl_test.slender_node_test_client (module)`, 233  
`dl_test.test_files (module)`, 234  
`dl_test.utilities (module)`, 230  
`dl_test.utilities.create_from_file (module)`, 230  
`dl_test.utilities.generate_test_subject_certs (module)`, 230  
`dl_test.utilities.my_subject (module)`, 232  
`dl_test.utilities.pem_in_http_header (module)`, 232  
`dl_test.utilities.populate_mn (module)`, 232  
`dl_test.utilities.test_object_generator (module)`, 232  
`dl_test.utilities.urlencode (module)`, 233  
`dl_test.xml_normalize (module)`, 234  
DataONE, 32  
DataONE Client Library for Python, 32  
DataONE Common Library for Python, 32  
DataONE Test Utilities for Python, 32  
`DataONEBaseClient (class in dl_client.baseclient)`, 194  
`DataONEBaseClient_1_1 (class in dl_client.baseclient_1_1)`, 197

DataONEBaseClient\_1\_2 (class in *d1\_common.xml*), 183  
*d1\_client.baseclient\_1\_2*), 199  
 DataONEBaseClient\_2\_0 (class in *d1\_common.types.exceptions*), 131  
*d1\_client.baseclient\_2\_0*), 199  
 DataONEClient (class in *d1\_client.d1client*), 214  
 DataONEException, 132  
 date\_utc\_now\_iso() (in module *d1\_common.date\_time*), 157  
 decode\_bu64() (in module *d1\_common.cert.jwt*), 115  
 decode\_der() (in module *d1\_common.cert.x509*), 123  
 decodePathElement() (in module *d1\_common.url*), 178  
 decodeQueryElement() (in module *d1\_common.url*), 179  
 default() (*d1\_common.util.ToJsonCompatibleTypes* method), 182  
 delete() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 213  
 delete() (*d1\_client.mnclient.MemberNodeClient* method), 215  
 DELETE() (*d1\_client.session.Session* method), 217  
 delete() (*d1\_client.solr\_client.SolrClient* method), 219  
 delete\_by\_query() (*d1\_client.solr\_client.SolrClient* method), 219  
 deleteReplicationMetadata() (*d1\_client.cnclient.CoordinatingNodeClient* method), 211  
 deleteReplicationMetadataResponse() (*d1\_client.cnclient.CoordinatingNodeClient* method), 211  
 deleteResponse() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 213  
 deleteResponse() (*d1\_client.mnclient.MemberNodeClient* method), 215  
 denyMapIdentity() (*d1\_client.cnclient.CoordinatingNodeClient* method), 208  
 denyMapIdentityResponse() (*d1\_client.cnclient.CoordinatingNodeClient* method), 207  
 describe() (*d1\_client.baseclient.DataONEBaseClient* method), 197  
 describeResponse() (*d1\_client.baseclient.DataONEBaseClient* method), 197  
 deserialize() (*d1\_common.resource\_map.ResourceMap* method), 166  
 deserialize() (in module *d1\_common.types.exceptions*), 131  
 deserialize() (in module *d1\_common.xml*), 183  
 deserialize\_d1\_exception() (in module *d1\_common.xml*), 183  
 deserialize\_from\_headers() (in module *d1\_common.types.exceptions*), 131  
 deserialize\_pem() (in module *d1\_common.cert.x509*), 122  
 deserialize\_pem\_file() (in module *d1\_common.cert.x509*), 122  
 deserialize\_subject\_info() (in module *d1\_common.cert.subject\_info*), 119  
 dict\_to\_pyxb() (in module *d1\_common.replication\_policy*), 163  
 disable\_cert\_validation() (in module *d1\_common.cert.x509*), 123  
 Django, 35  
 DN, 34  
 do\_GET() (*d1\_test.replication\_tester.replication\_server.Handler* method), 229  
 do\_POST() (*d1\_test.replication\_tester.replication\_server.Handler* method), 229  
 Dokan, 34  
 download\_as\_der() (in module *d1\_common.cert.x509*), 122  
 download\_as\_obj() (in module *d1\_common.cert.x509*), 123  
 download\_as\_pem() (in module *d1\_common.cert.x509*), 123  
 dst() (*d1\_common.date\_time.FixedOffset* method), 153  
 dst() (*d1\_common.date\_time.UTC* method), 153  
 dt\_from\_http\_datetime\_str() (in module *d1\_common.date\_time*), 156  
 dt\_from\_iso8601\_str() (in module *d1\_common.date\_time*), 156  
 dt\_from\_ts() (in module *d1\_common.date\_time*), 156  
 dump() (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
 dump() (in module *d1\_common.wrap.access\_policy*), 144  
 dump\_request\_and\_response() (*d1\_client.session.Session* method), 218  
 dump\_to\_log() (*d1\_common.util.EventCounter* method), 181  
 dumpExtensions() (in module *d1\_test.utilities.my\_subject*), 232

## E

echoCredentials() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 214  
 echoCredentialsResponse() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 214  
 echoIndexedObject() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 214

method), 214  
 echoIndexedObjectResponse() (d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0.id\_url\_mismatches() (in module method), 214  
 echoSystemMetadata() (d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0 method), 214  
 echoSystemMetadataResponse() (d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0 method), 214  
 encode() (d1\_common.types.exceptions.DataONEException method), 132  
 encode\_bu64() (in module d1\_common.cert.jwt), 115  
 encodeAndJoinPathElements() (in module d1\_common.url), 179  
 encodePathElement() (in module d1\_common.url), 178  
 encodeQueryElement() (in module d1\_common.url), 179  
 end\_task\_type() (d1\_common.utils.progress\_logger.ProgressLogger method), 138  
 etree\_replace\_namespace() (in module d1\_common.type\_conversions), 177  
 etree\_to\_pyxb() (in module d1\_common.type\_conversions), 177  
 etree\_to\_stable\_tree() (in module d1\_test.xml\_normalize), 235  
 etree\_to\_str() (in module d1\_common.type\_conversions), 177  
 event() (d1\_common.utils.progress\_logger.ProgressLogger method), 138  
 event\_dict (d1\_common.util.EventCounter attribute), 180  
 EventCounter (class in d1\_common.util), 180  
 extract\_issuer\_ca\_cert\_url() (in module d1\_common.cert.x509), 123  
 extract\_subject\_from\_dn() (in module d1\_common.cert.x509), 121  
 extract\_subject\_info\_extension() (in module d1\_common.cert.x509), 122  
 extract\_subjects() (in module d1\_common.cert.subject\_info), 117  
 extract\_subjects() (in module d1\_common.cert.subjects), 121  
 extract\_subjects() (in module d1\_common.cert.x509), 121  
 extract\_version\_tag\_from\_url() (in module d1\_common.type\_conversions), 173  
**F**  
 field\_alpha\_histogram() (d1\_client.solr\_client.SolrClient method), 219  
 FileIterator (class in d1\_common.iter.file), 127  
 FileLikeObjectIterator (class in d1\_common.iter.file), 127  
 find\_url\_mismatches() (in module d1\_common.url), 179  
 fitness\_and\_quality\_parsed() (in module d1\_common.ext.mimemanager), 126  
 FixedOffset (class in d1\_common.date\_time), 153  
 format\_checksum() (in module d1\_common.checksum), 152  
 format\_diff\_pyxb() (in module d1\_common.xml), 185  
 format\_diff\_xml() (in module d1\_common.xml), 185  
 format\_json\_to\_normalized\_pretty\_json() (in module d1\_common.util), 181  
 format\_sec\_to\_dhm() (in module d1\_common.util), 182  
 friendly\_format() (d1\_common.types.exceptions.DataONEException method), 132  
 FUSE, 34  
 fusepy, 34  
**G**  
 gen\_safe\_path() (in module d1\_common.utils.filesystem), 135  
 gen\_safe\_path\_element() (in module d1\_common.utils.filesystem), 135  
 gen\_subject\_info\_tree() (in module d1\_common.cert.subject\_info), 119  
 gen\_sysmeta() (in module d1\_test.utilities.create\_from\_file), 230  
 Generate (class in d1\_test.instance\_generator.format\_id), 222  
 Generate (class in d1\_test.instance\_generator.user\_agent), 226  
 generate() (in module d1\_test.instance\_generator.checksum), 222  
 generate() (in module d1\_test.instance\_generator.identifier), 222  
 generate() (in module d1\_test.instance\_generator.media\_type), 223  
 generate() (in module d1\_test.instance\_generator.node\_ref), 223  
 generate() (in module d1\_test.instance\_generator.person), 223  
 generate() (in module d1\_test.instance\_generator.replication\_policy), 225  
 generate() (in module d1\_test.instance\_generator.replication\_status), 225

`generate()` (in module `d1_test.instance_generator.subject`), 225  
`generate_bare()` (in module `d1_test.instance_generator.identifier`), 222  
`generate_bare()` (in module `d1_test.instance_generator.node_ref`), 223  
`generate_bare()` (in module `d1_test.instance_generator.subject`), 225  
`generate_pid()` (in module `d1_test.instance_generator.identifier`), 222  
`generate_random_ascii()` (in module `d1_test.replication_tester.test_object_generator`), 230  
`generate_random_ascii()` (in module `d1_test.utilities.test_object_generator`), 232  
`generate_science_object_with_sysmeta()` (in module `d1_test.replication_tester.test_object_generator`) method, 147  
`generate_science_object_with_sysmeta()` (in module `d1_test.utilities.test_object_generator`), 232  
`generate_sid()` (in module `d1_test.instance_generator.identifier`), 222  
`generateIdentifier()` (`d1_client.baseclient.DataONEBaseClient` method), 197  
`generateIdentifierResponse()` (`d1_client.baseclient.DataONEBaseClient` method), 197  
`get()` (`d1_client.baseclient.DataONEBaseClient` method), 196  
`GET()` (`d1_client.session.Session` method), 217  
`get()` (`d1_client.solr_client.SolrClient` method), 219  
`get()` (`d1_test.slender_node_test_client.SlenderNodeTestClient` method), 234  
`get_abs_test_file_path()` (in module `d1_test.test_files`), 234  
`get_and_save()` (`d1_client.baseclient.DataONEBaseClient` method), 197  
`get_api_major_by_base_url()` (in module `d1_client.d1client`), 214  
`get_attr_value()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 147  
`get_auto()` (in module `d1_common.xml`), 186  
`get_bu64_tup()` (in module `d1_common.cert.jwt`), 113  
`get_checksum_calculator_by_dataone_designator()` (in module `d1_common.checksum`), 151  
`get_client_class_by_version_tag()` (in module `d1_client.d1client`), 214  
`get_client_type()` (in module `d1_client.d1client`), 214  
`get_content_type()` (in module `d1_common.util`), 180  
`get_curl_command_line()` (`d1_client.session.Session` method), 218  
`get_d1_env()` (in module `d1_common.env`), 158  
`get_d1_env_by_base_url()` (in module `d1_common.env`), 159  
`get_d1_env_keys()` (in module `d1_common.env`), 158  
`get_default_checksum_algorithm()` (in module `d1_common.checksum`), 152  
`get_effective_perm_list()` (`d1_common.wrap.access_policy.AccessPolicyWrapper` method), 142  
`get_effective_perm_list()` (in module `d1_common.wrap.access_policy`), 144  
`get_element_by_attr_key()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 146  
`get_element_by_name()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 146  
`get_element_by_xpath()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 146  
`get_element_dt()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 148  
`get_element_list_by_attr_key()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 146  
`get_element_list_by_name()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 146  
`get_element_text()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 147  
`get_element_text_by_attr_key()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper` method), 147  
`get_ext_val_str()` (in module `d1_common.cert.x509`), 124  
`get_extension_by_name()` (in module `d1_common.cert.x509`), 124  
`get_in_max()` (`d1_client.solr_client.SolrClient` method), 219  
`get_field_values()` (`d1_client.solr_client.SolrClient` method), 219  
`get_highest_perm_str()` (`d1_common.wrap.access_policy.AccessPolicyWrapper` method), 142  
`get_highest_perm_str()` (in module `d1_common.wrap.access_policy`), 144  
`get_identifiers()` (in module `d1_common.revision`), 169

`get_ids()` (*d1\_client.solr\_client.SolrClient* method), 219  
`get_jwt_bu64()` (in module *d1\_common.cert.jwt*), 113  
`get_jwt_dict()` (in module *d1\_common.cert.jwt*), 114  
`get_jwt_tup()` (in module *d1\_common.cert.jwt*), 113  
`get_label_set()` (*d1\_common.cert.subject\_info.SubjectInfoNode* method), 120  
`get_leaf_node_path_list()` (*d1\_common.cert.subject\_info.SubjectInfoNode* method), 120  
`get_normalized_perm_list()` (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
`get_normalized_perm_list()` (in module *d1\_common.wrap.access\_policy*), 144  
`get_normalized_pyxb()` (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
`get_normalized_pyxb()` (in module *d1\_common.wrap.access\_policy*), 144  
`get_normalized_xml_representation()` (in module *d1\_test.xml\_normalize*), 235  
`get_opt_attr()` (in module *d1\_common.xml*), 186  
`get_opt_val()` (in module *d1\_common.xml*), 186  
`get_path_list()` (*d1\_common.cert.subject\_info.SubjectInfoNode* method), 120  
`get_path_str()` (*d1\_common.cert.subject\_info.SubjectInfoNode* method), 120  
`get_pids_in_revision_chain()` (in module *d1\_common.revision*), 169  
`get_pretty_xml()` (*d1\_common.wrap.simple\_xml.SimpleXMLWrapper* method), 146  
`get_public_key_pem()` (in module *d1\_common.cert.x509*), 125  
`get_pyxb()` (*d1\_common.types.exceptions.DataONEException* method), 132  
`get_pyxb_binding_by_api_version()` (in module *d1\_common.type\_conversions*), 173  
`get_pyxb_namespaces()` (in module *d1\_common.type\_conversions*), 174  
`get_req_val()` (in module *d1\_common.xml*), 187  
`get_sort_key_()` (*d1\_test.xml\_normalize.StableNode* method), 235  
`get_str()` (*d1\_test.xml\_normalize.StableNode* method), 235  
`get_subject_set()` (*d1\_common.cert.subject\_info.SubjectInfoNode* method), 121  
`get_subject_with_file_validation()` (in module *d1\_common.cert.jwt*), 113  
`get_subject_with_local_validation()` (in module *d1\_common.cert.jwt*), 112  
`get_subject_with_remote_validation()` (in module *d1\_common.cert.jwt*), 113  
`get_subject_without_validation()` (in module *d1\_common.cert.jwt*), 113  
`get_subjects_with_equal_or_higher_perm()` (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
`get_subjects_with_equal_or_higher_perm()` (in module *d1\_common.wrap.access\_policy*), 144  
`get_supported_algorithms()` (in module *d1\_common.checksum*), 152  
`get_val_list()` (in module *d1\_common.cert.x509*), 124  
`get_val_str()` (in module *d1\_common.cert.x509*), 124  
`get_version_tag()` (in module *d1\_common.type\_conversions*), 173  
`get_version_tag_by_d1_client()` (in module *d1\_client.d1client*), 214  
`get_version_tag_by_pyxb_binding()` (in module *d1\_common.type\_conversions*), 173  
`get_xml()` (*d1\_common.wrap.simple\_xml.SimpleXMLWrapper* method), 146  
`get_xml_below_element()` (*d1\_common.wrap.simple\_xml.SimpleXMLWrapper* method), 146  
`getAggregatedPids()` (*d1\_common.resource\_map.ResourceMap* method), 168  
`getAggregatedScienceDataPids()` (*d1\_common.resource\_map.ResourceMap* method), 168  
`getAggregatedScienceMetadataPids()` (*d1\_common.resource\_map.ResourceMap* method), 168  
`getAggregation()` (*d1\_common.resource\_map.ResourceMap* method), 166  
`getAllPredicates()` (*d1\_common.resource\_map.ResourceMap* method), 167  
`getAllTriples()` (*d1\_common.resource\_map.ResourceMap* method), 167  
`getCapabilities()` (*d1\_client.mnclient.MemberNodeClient* method), 214  
`getCapabilitiesResponse()` (*d1\_client.mnclient.MemberNodeClient* method), 214  
`getChecksum()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 203  
`getChecksum()` (*d1\_client.mnclient.MemberNodeClient* method), 214  
`getChecksumResponse()`

(*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 202  
 getChecksumResponse ()  
 (*d1\_client.mnclient.MemberNodeClient*  
*method*), 214  
 getFormat () (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 200  
 getFormatResponse ()  
 (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 200  
 getLogRecords () (*d1\_client.baseclient.DataONEBaseClient*  
*method*), 196  
 getLogRecordsResponse ()  
 (*d1\_client.baseclient.DataONEBaseClient*  
*method*), 196  
 getObjectByPid () (*d1\_common.resource\_map.ResourceMap*  
*method*), 166  
 getPackage () (*d1\_client.mnclient\_1\_2.MemberNodeClient\_1\_2*  
*method*), 216  
 getPackageResponse ()  
 (*d1\_client.mnclient\_1\_2.MemberNodeClient\_1\_2*  
*method*), 216  
 getQueryEngineDescription ()  
 (*d1\_client.baseclient\_1\_1.DataONEBaseClient\_1\_1*  
*method*), 198  
 getQueryEngineDescription ()  
 (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 204  
 getQueryEngineDescriptionResponse ()  
 (*d1\_client.baseclient\_1\_1.DataONEBaseClient\_1\_1*  
*method*), 198  
 getQueryEngineDescriptionResponse ()  
 (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 204  
 getReplica () (*d1\_client.mnclient.MemberNodeClient*  
*method*), 215  
 getReplicaResponse ()  
 (*d1\_client.mnclient.MemberNodeClient*  
*method*), 215  
 getResourceMapPid ()  
 (*d1\_common.resource\_map.ResourceMap*  
*method*), 167  
 getResponse () (*d1\_client.baseclient.DataONEBaseClient*  
*method*), 196  
 getSubjectFromCertFile () (in module  
*d1\_test.utilities.my\_subject*), 232  
 getSubjectFromName () (in module  
*d1\_test.utilities.my\_subject*), 232  
 getSubjectInfo () (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 206  
 getSubjectInfoFromCert () (in module  
*d1\_test.utilities.my\_subject*), 232  
 getSubjectInfoResponse ()  
 (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 206  
 getSubjectObjectsByPredicate ()  
 (*d1\_common.resource\_map.ResourceMap*  
*method*), 168  
 getSystemMetadata ()  
 (*d1\_client.baseclient.DataONEBaseClient*  
*method*), 197  
 getSystemMetadata ()  
 (*d1\_test.slender\_node\_test\_client.SlenderNodeTestClient*  
*method*), 234  
 getSystemMetadataResponse ()  
 (*d1\_client.baseclient.DataONEBaseClient*  
*method*), 197  
 GMN, 32  
 H  
 Handler (class in *d1\_test.replication\_tester.replication\_server*),  
 229  
 has\_replication\_policy () (in module  
*d1\_common.replication\_policy*), 161  
 has\_tz () (in module *d1\_common.date\_time*), 154  
 hasReservation () (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 202  
 hasReservationResponse ()  
 (*d1\_client.cnclient.CoordinatingNodeClient*  
*method*), 202  
 HEAD () (*d1\_client.session.Session* *method*), 217  
 http\_datetime\_str\_from\_dt () (in module  
*d1\_common.date\_time*), 155  
 I  
 IdentifierNotUnique, 133  
 Identity Provider, 34  
 initialize () (*d1\_common.resource\_map.ResourceMap*  
*method*), 165  
 InsufficientResources, 133  
 InvalidCredentials, 133  
 InvalidRequest, 133  
 InvalidSystemMetadata, 133  
 InvalidToken, 134  
 Investigator Toolkit (ITK), 32  
 is\_blocked () (in module  
*d1\_common.replication\_policy*), 162  
 is\_empty () (*d1\_common.wrap.access\_policy.AccessPolicyWrapper*  
*method*), 142  
 is\_empty () (in module  
*d1\_common.wrap.access\_policy*), 144  
 is\_leaf (*d1\_common.cert.subject\_info.SubjectInfoNode*  
 attribute), 120  
 is\_multipart () (in module *d1\_common.multipart*),  
 160  
 is\_preferred () (in module  
*d1\_common.replication\_policy*), 162

`is_private()` (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
`is_private()` (in module *d1\_common.wrap.access\_policy*), 144  
`is_public()` (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 142  
`is_public()` (in module *d1\_common.wrap.access\_policy*), 144  
`is_pyxb()` (in module *d1\_common.type\_conversions*), 175  
`is_pyxb_d1_type()` (in module *d1\_common.type\_conversions*), 175  
`is_pyxb_d1_type_name()` (in module *d1\_common.type\_conversions*), 175  
`is_supported_algorithm()` (in module *d1\_common.checksum*), 152  
`is_sysmeta_pyxb()` (in module *d1\_common.system\_metadata*), 171  
`is_urls_equivalent()` (in module *d1\_common.url*), 179  
`is_utc()` (in module *d1\_common.date\_time*), 154  
`is_valid_iso8601()` (in module *d1\_common.date\_time*), 153  
`is_valid_utf8()` (in module *d1\_common.xml*), 186  
`isAuthorized()` (*d1\_client.baseclient.DataONEBaseClient* method), 197  
`isAuthorizedResponse()` (*d1\_client.baseclient.DataONEBaseClient* method), 197  
`isHttpOrHttps()` (in module *d1\_common.url*), 178  
`isNodeAuthorized()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 211  
`isNodeAuthorizedResponse()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 211  
ISO8601, 36  
**J**  
`joinPathElements()` (in module *d1\_common.url*), 179  
JwtException, 115  
**L**  
`leaf_node_gen()` (*d1\_common.cert.subject\_info.SubjectInfoNode* attribute), 120  
`listChecksumAlgorithms()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 201  
`listChecksumAlgorithmsResponse()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 201  
`listFormats()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 200  
`listNodes()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 200  
`listNodesResponse()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 201  
`listObjects()` (*d1\_client.baseclient.DataONEBaseClient* method), 197  
`listObjectsResponse()` (*d1\_client.baseclient.DataONEBaseClient* method), 197  
`listSubjects()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 207  
`listSubjectsResponse()` (*d1\_client.cnclient.CoordinatingNodeClient* method), 206  
`listViews()` (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 214  
`listViews()` (*d1\_client.mnclient\_1\_2.MemberNodeClient\_1\_2* method), 216  
`listViewsResponse()` (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 214  
`listViewsResponse()` (*d1\_client.mnclient\_1\_2.MemberNodeClient\_1\_2* method), 216  
LOA, 36  
`load_bin()` (in module *d1\_test.test\_files*), 234  
`load_cert()` (in module *d1\_test.test\_files*), 234  
`load_json()` (in module *d1\_common.util*), 181  
`load_json()` (in module *d1\_test.test\_files*), 234  
`load_jwt()` (in module *d1\_test.test\_files*), 234  
`load_utf8_to_str()` (in module *d1\_test.test\_files*), 234  
`load_xml_to_pyxb()` (in module *d1\_test.test\_files*), 234  
`load_xml_to_str()` (in module *d1\_test.test\_files*), 234  
`local_now()` (in module *d1\_common.date\_time*), 158  
`local_now_iso()` (in module *d1\_common.date\_time*), 158  
`log_and_count()` (*d1\_common.util.EventCounter* method), 181  
`log_cert_info()` (in module *d1\_common.cert.x509*), 124  
`log_jwt_bu64_info()` (in module *d1\_common.cert.jwt*), 114  
`log_jwt_dict_info()` (in module *d1\_common.cert.jwt*), 114  
`log_setup()` (in module *d1\_common.util*), 179  
LoggingContext (class in *d1\_common.logging\_context*), 159

- LogRecordIterator (class in *normalize\_datetime\_to\_utc()* (in module *d1\_client.iter.logrecord*), 189  
*d1\_common.date\_time*), 156
- LogRecordIteratorMulti (class in *normalize\_in\_place()* (in module *d1\_client.iter.logrecord\_multi*), 190  
*d1\_common.system\_metadata*), 171
- M**
- macfuse, 34
- main() (in module *d1\_test.replication\_tester.replication\_tester*), 179
- main() (in module *d1\_test.utilities.create\_from\_file*), 229
- main() (in module *d1\_test.utilities.generate\_test\_subject\_certs*), 230
- main() (in module *d1\_test.utilities.populate\_mn*), 232
- makeCNBaseURL() (in module *d1\_test.utilities.create\_from\_file*), 230
- makeMNBaseURL() (in module *d1\_test.utilities.populate\_mn*), 232
- mapIdentity() (*d1\_client.cnclient.CoordinatingNodeClient* method), 207
- mapIdentityResponse() (*d1\_client.cnclient.CoordinatingNodeClient* method), 207
- MemberNodeClient (class in *d1\_client.mnclient*), 214
- MemberNodeClient\_1\_1 (class in *d1\_client.mnclient\_1\_1*), 215
- MemberNodeClient\_1\_2 (class in *d1\_client.mnclient\_1\_2*), 215
- MemberNodeClient\_2\_0 (class in *d1\_client.mnclient\_2\_0*), 216
- Metacat, 32
- method\_obj() (in module *d1\_common.wrap.access\_policy*), 144
- minixsv, 35
- mk\_func() (in module *d1\_common.wrap.access\_policy*), 144
- MN, 33
- mod\_ssl, 35
- mod\_wsgi, 35
- MPM, 35
- MySQL, 35
- N**
- name (*d1\_common.types.exceptions.DataONEException* attribute), 132
- nested\_update() (in module *d1\_common.util*), 180
- Node, 33
- node\_gen(*d1\_common.cert.subject\_info.SubjectInfoNode* attribute), 120
- NodeListIterator (class in *d1\_client.iter.node*), 191
- normalize() (in module *d1\_common.multipart*), 160
- normalize() (in module *d1\_common.replication\_policy*), 162
- normalize\_datetime\_to\_utc()* (in module *d1\_common.date\_time*), 156
- normalize\_in\_place()* (in module *d1\_common.system\_metadata*), 171
- normalize\_request\_response\_dump()* (in module *d1\_client.util*), 221
- normalizeTarget()* (in module *d1\_common.url*), 179
- NotAuthorized, 134
- NotFound, 134
- NotImplemented, 134
- O**
- OAI-ORE Resource Map, 36
- ObjectListIterator (class in *d1\_client.iter.objectlist*), 192
- ObjectListIteratorMulti (class in *d1\_client.iter.objectlist\_multi*), 193
- OpenSSL, 34, 36
- OPTIONS() (*d1\_client.session.Session* method), 218
- Oracle, 35
- P**
- pack\_echo\_header() (in module *d1\_test.mock\_api.create*), 226
- parent\_gen(*d1\_common.cert.subject\_info.SubjectInfoNode* attribute), 120
- parse\_media\_range() (in module *d1\_common.ext.mimeparser*), 126
- parse\_mime\_type() (in module *d1\_common.ext.mimeparser*), 125
- parse\_response() (in module *d1\_common.multipart*), 159
- parse\_str() (in module *d1\_common.multipart*), 159
- parse\_xml() (*d1\_common.wrap.simple\_xml.SimpleXMLWrapper* method), 146
- parseDoc() (*d1\_common.resource\_map.ResourceMap* method), 169
- parseUrl() (in module *d1\_common.url*), 178
- path\_generator() (in module *d1\_common.iter.path*), 127
- pem\_in\_http\_header\_to\_pem\_in\_string() (in module *d1\_test.utilities.pem\_in\_http\_header*), 232
- pem\_in\_string\_to\_pem\_in\_http\_header() (in module *d1\_test.utilities.pem\_in\_http\_header*), 232
- PID, 33
- ping() (*d1\_client.baseclient.DataONEBaseClient* method), 196
- pingResponse() (*d1\_client.baseclient.DataONEBaseClient* method), 196
- POST() (*d1\_client.session.Session* method), 217
- PostgreSQL, 35

print\_logging() (in module *d1\_common.util*), 181  
 process\_input() (in module *d1\_test.utilities.urlencode*), 233  
 process\_row() (*d1\_client.solr\_client.SolrSearchResponseIterator* *d1\_test.instance\_generator.random\_data*),  
 method), 220  
 ProgressLogger (class in *random\_bool\_factor*() (in module  
*d1\_common.utils.progress\_logger*), 137  
 Psycpg2, 35  
 PUT() (*d1\_client.session.Session* method), 217  
 Python, 35  
 python-dateutil, 35  
 python-iso8601, 36  
 python-setuptools, 36  
 PyXB, 35  
 pyxb\_binding(*d1\_client.baseclient.DataONEBaseClient* *random\_checksum\_algorithm*() (in module  
*attribute*), 196  
 pyxb\_get\_namespace\_name() (in module *random\_choice\_pop*() (in module  
*d1\_common.type\_conversions*), 175  
 pyxb\_get\_type\_name() (in module *d1\_test.instance\_generator.random\_data*),  
*d1\_common.type\_conversions*), 175  
 pyxb\_is\_dataone\_exception() (in module *random\_cn*() (in module  
*d1\_common.types.exceptions*), 131  
 pyxb\_is\_v1() (in module *random\_email*() (in module  
*d1\_common.type\_conversions*), 176  
 pyxb\_is\_v2() (in module *random\_lower\_ascii*() (in module  
*d1\_common.type\_conversions*), 176  
 pyxb\_to\_dict() (in module *d1\_common.node*), 160  
 pyxb\_to\_dict() (in module *random\_mn*() (in module  
*d1\_common.replication\_policy*), 163  
 pyxb\_to\_etree() (in module *d1\_test.instance\_generator.random\_data*),  
*d1\_common.type\_conversions*), 177  
 pyxb\_to\_str() (in module *random\_names*() (in module  
*d1\_common.type\_conversions*), 177  
 pyxb\_to\_v1\_str() (in module *random\_sized\_sample*() (in module  
*d1\_common.type\_conversions*), 174  
 pyxb\_to\_v2\_str() (in module *random\_sized\_sample\_pop*() (in module  
*d1\_common.type\_conversions*), 174  
 random\_checksum\_algorithm() (in module  
*d1\_test.instance\_generator.random\_data*), 222  
 random\_choice\_pop() (in module  
*d1\_test.instance\_generator.random\_data*), 225  
 random\_cn() (in module  
*d1\_test.instance\_generator.random\_data*), 223  
 random\_email() (in module  
*d1\_test.instance\_generator.random\_data*), 224  
 random\_lower\_ascii() (in module  
*d1\_test.instance\_generator.random\_data*), 223  
 random\_mn() (in module  
*d1\_test.instance\_generator.random\_data*), 223  
 random\_names() (in module  
*d1\_test.instance\_generator.names*), 223  
 random\_sized\_sample() (in module  
*d1\_test.instance\_generator.random\_data*), 224  
 random\_sized\_sample\_pop() (in module  
*d1\_test.instance\_generator.random\_data*), 225  
 random\_subj() (in module  
*d1\_test.instance\_generator.random\_data*), 223  
 random\_unicode\_char() (in module  
*d1\_test.instance\_generator.random\_data*), 224  
 random\_unicode\_char\_no\_whitespace() (in  
 module *d1\_test.instance\_generator.random\_data*),  
 224  
 random\_unicode\_name() (in module  
*d1\_test.instance\_generator.random\_data*), 224  
 random\_unicode\_name\_list() (in module  
*d1\_test.instance\_generator.random\_data*), 224  
 random\_unicode\_name\_unique\_list() (in  
 module *d1\_test.instance\_generator.random\_data*), 224  
 random\_3\_words() (in module

## Q

quality() (in module *d1\_common.ext.mimeparser*), 126  
 quality\_parsed() (in module  
*d1\_common.ext.mimeparser*), 126  
 query() (*d1\_client.baseclient\_1\_1.DataONEBaseClient\_1\_1* *random\_unicode\_name*() (in module  
 method), 198  
 query() (*d1\_client.cnclient.CoordinatingNodeClient* *random\_unicode\_name\_list*() (in module  
 method), 203  
 queryResponse() (*d1\_client.baseclient\_1\_1.DataONEBaseClient\_1\_1* *random\_unicode\_name\_unique\_list*() (in  
 method), 198  
 queryResponse() (*d1\_client.cnclient.CoordinatingNodeClient* *random\_unicode\_name\_unique\_list*() (in  
 method), 203

## R

random\_3\_words() (in module

`module d1_test.instance_generator.random_data),`  
`replace_by_xml()` (`d1_common.wrap.simple_xml.SimpleXMLWrapper`  
`method`), 149  
`random_unicode_str()` (`in module`  
`d1_test.instance_generator.random_data`),  
`replace_namespace_with_prefix()` (`in mod-`  
`ule d1_common.type_conversions`), 177  
`random_within_range()` (`in module`  
`d1_test.instance_generator.random_data`),  
`replicate()` (`d1_client.mnclient.MemberNodeClient`  
`method`), 215  
`random_word()` (`in module`  
`d1_test.instance_generator.random_data`),  
`replicateResponse()`  
`(d1_client.mnclient.MemberNodeClient`  
`method)`, 215  
`random_word_list()` (`in module`  
`d1_test.instance_generator.random_data`),  
`Replication target`, 32  
`random_word_unique_list()` (`in module`  
`d1_test.instance_generator.random_data`),  
`ReplicationTester` (`class in`  
`d1_test.replication_tester.replication_tester`),  
`229`  
`rdn_escape()` (`in module d1_common.cert.x509`),  
`122`  
`requestMapIdentity()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 208  
`reformat_to_pretty_xml()` (`in module`  
`d1_common.xml`), 184  
`requestMapIdentityResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 208  
`register()` (`d1_client.cnclient.CoordinatingNodeClient`  
`method`), 212  
`reserveIdentifier()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 200  
`registerAccount()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 205  
`reserveIdentifierResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 200  
`registerAccountResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 205  
`resolve()` (`d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 202  
`registerResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 212  
`resolveResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 202  
`remove_children()`  
`(d1_common.wrap.simple_xml.SimpleXMLWrapper`  
`method)`, 149  
`revision_list_to_obsoleted_by_dict()` (`in`  
`module d1_common.revision`), 170  
`remove_perm()` (`d1_common.wrap.access_policy.AccessPolicyWrapper`  
`method)`, 143  
`revision_list_to_obsoletes_dict()` (`in`  
`module d1_common.revision`), 170  
`remove_perm()` (`in module`  
`d1_common.wrap.access_policy`), 144  
`round_to_nearest()` (`in module`  
`d1_common.date_time`), 158  
`remove_subj()` (`d1_common.wrap.access_policy.AccessPolicyWrapper`  
`method)`, 144  
`run()` (`d1_test.replication_tester.replication_server.TestHTTPServer`  
`method)`, 229  
`remove_subj()` (`in module`  
`d1_common.wrap.access_policy`), 144  
**S**  
`removeMapIdentity()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 207  
`save_json()` (`in module d1_common.util`), 181  
`removeMapIdentityResponse()`  
`(d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 207  
`Science Data`, 33  
`Science Metadata`, 33  
`search()` (`d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 203  
`search()` (`d1_client.solr_client.SolrClient` `method`),  
`219`  
`searchResponse()` (`d1_client.cnclient.CoordinatingNodeClient`  
`method)`, 203  
`replace_by_etree()`  
`(d1_common.wrap.simple_xml.SimpleXMLWrapper`  
`method)`, 149  
`Self signed certificate`, 34

<code>serialize_cert_to_der()</code>	(in module <code>d1_common.cert.x509</code> ), 125	<code>setObsoletedBy()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 201
<code>serialize_cert_to_pem()</code>	(in module <code>d1_common.cert.x509</code> ), 125	<code>setObsoletedByResponse()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 201
<code>serialize_for_transport()</code>	(in module <code>d1_common.xml</code> ), 183	<code>setReplicationPolicy()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 210
<code>serialize_gen()</code>	(in module <code>d1_common.xml</code> ), 183	<code>setReplicationPolicyResponse()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 210
<code>serialize_to_display()</code>	( <code>d1_common.resource_map.ResourceMap</code> method), 166	<code>setReplicationStatus()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 209
<code>serialize_to_display()</code>	( <code>d1_common.types.exceptions.DataONEException</code> method), 132	<code>setReplicationStatusResponse()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 209
<code>serialize_to_headers()</code>	( <code>d1_common.types.exceptions.DataONEException</code> method), 132	<code>setRightsHolder()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 204
<code>serialize_to_normalized_compact_json()</code>	(in module <code>d1_common.util</code> ), 182	<code>setRightsHolderResponse()</code> ( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 204
<code>serialize_to_normalized_pretty_json()</code>	(in module <code>d1_common.util</code> ), 182	SID, 33
<code>serialize_to_transport()</code>	( <code>d1_common.resource_map.ResourceMap</code> method), 165	<code>SimpleXMLWrapper</code> (class in <code>d1_common.wrap.simple_xml</code> ), 145
<code>serialize_to_transport()</code>	( <code>d1_common.types.exceptions.DataONEException</code> method), 132	<code>SimpleXMLWrapperException</code> , 149
<code>serialize_to_xml_str()</code>	(in module <code>d1_common.xml</code> ), 184	<code>size</code> ( <code>d1_common.iter.bytes.BytesIterator</code> attribute), 127
Server key, 33		<code>size</code> ( <code>d1_common.iter.file.FileIterator</code> attribute), 127
Server Side Authentication, 34		<code>size</code> ( <code>d1_common.iter.file.FileLikeObjectIterator</code> attribute), 127
Server side certificate, 34		<code>size</code> ( <code>d1_common.iter.string.StringIterator</code> attribute), 127
ServiceFailure, 134		<code>SlenderNodeTestClient</code> (class in <code>d1_test.slender_node_test_client</code> ), 233
Session (class in <code>d1_client.session</code> ), 216		Solr, 36
<code>set_attr_text()</code> ( <code>d1_common.wrap.simple_xml.SimpleXMLWrapper</code> method), 148		<code>SolrArrayResponseIterator</code> (class in <code>d1_client.solr_client</code> ), 220
<code>set_element_dt()</code> ( <code>d1_common.wrap.simple_xml.SimpleXMLWrapper</code> method), 148		<code>SolrArrayTransformer</code> (class in <code>d1_client.solr_client</code> ), 220
<code>set_element_text()</code>	( <code>d1_common.wrap.simple_xml.SimpleXMLWrapper</code> method), 147	<code>SolrClient</code> (class in <code>d1_client.solr_client</code> ), 218
<code>set_element_text_by_attr_key()</code>	( <code>d1_common.wrap.simple_xml.SimpleXMLWrapper</code> method), 147	<code>SolrRecordTransformerBase</code> (class in <code>d1_client.solr_client</code> ), 219
<code>setAccessPolicy()</code>	( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 205	<code>SolrSearchResponseIterator</code> (class in <code>d1_client.solr_client</code> ), 220
<code>setAccessPolicyResponse()</code>	( <code>d1_client.cnclient.CoordinatingNodeClient</code> method), 204	<code>SolrSubsampleResponseIterator</code> (class in <code>d1_client.solr_client</code> ), 220
<code>setDocumentedBy()</code>	( <code>d1_common.resource_map.ResourceMap</code> method), 167	<code>SolrValuesResponseIterator</code> (class in <code>d1_client.solr_client</code> ), 220
<code>setDocuments()</code> ( <code>d1_common.resource_map.ResourceMap</code> method), 167		<code>sort()</code> ( <code>d1_test.xml_normalize.StableNode</code> method), 235
		<code>sort_elements_by_child_values()</code> (in module <code>d1_common.xml</code> ), 185

sort\_value\_list\_pyxb() (in module *d1\_common.xml*), 185  
 SQLite3, 35  
 SSL, 34  
 SSL handshake, 34  
 StableNode (class in *d1\_test.xml\_normalize*), 235  
 StableTree (in module *d1\_test.xml\_normalize*), 235  
 start\_task() (*d1\_common.utils.progress\_logger.ProgressLogger* attribute), 119  
     method), 138  
 start\_task\_type() (in module *d1\_common.utils.progress\_logger.ProgressLogger*), 137  
 stop() (*d1\_test.replication\_tester.replication\_server.TestHTTPServer*), 229  
     method), 229  
 str\_is\_error() (in module *d1\_common.type\_conversions*), 176  
 str\_is\_identifier() (in module *d1\_common.type\_conversions*), 176  
 str\_is\_objectList() (in module *d1\_common.type\_conversions*), 176  
 str\_is\_v1() (in module *d1\_common.type\_conversions*), 175  
 str\_is\_v2() (in module *d1\_common.type\_conversions*), 175  
 str\_is\_well\_formed() (in module *d1\_common.type\_conversions*), 176  
 str\_to\_etree() (in module *d1\_common.type\_conversions*), 176  
 str\_to\_pyxb() (in module *d1\_common.type\_conversions*), 176  
 str\_to\_v1\_pyxb() (in module *d1\_common.type\_conversions*), 174  
 str\_to\_v1\_str() (in module *d1\_common.type\_conversions*), 174  
 str\_to\_v2\_pyxb() (in module *d1\_common.type\_conversions*), 175  
 str\_to\_v2\_str() (in module *d1\_common.type\_conversions*), 174  
 StringIterator (class in *d1\_common.iter.string*), 129  
 strip\_log() (in module *d1\_common.type\_conversions*), 178  
 strip\_logEntry() (in module *d1\_common.type\_conversions*), 178  
 strip\_node() (in module *d1\_common.type\_conversions*), 178  
 strip\_node\_list() (in module *d1\_common.type\_conversions*), 178  
 strip\_system\_metadata() (in module *d1\_common.type\_conversions*), 178  
 strip\_timezone() (in module *d1\_common.date\_time*), 157  
 strip\_v2\_elements() (in module *d1\_common.type\_conversions*), 178  
 stripElementSlashes() (in module *d1\_common.url*), 179  
 subj\_has\_perm() (*d1\_common.wrap.access\_policy.AccessPolicyWrapper* method), 143  
 subj\_has\_perm() (in module *d1\_common.wrap.access\_policy*), 144  
 SUBJECT\_NODE\_TAG (*d1\_common.cert.subject\_info.SubjectInfoNode*), 119  
 SubjectInfoNode (class in *d1\_common.cert.subject\_info*), 119  
 SubjectInfoTree (in module *d1\_common.cert.subject\_info*), 121  
 SynchronizationFailed, 135  
 synchronizationFailed() (in module *d1\_client.mnclient.MemberNodeClient* method), 215  
 synchronizationFailedResponse() (in module *d1\_client.mnclient.MemberNodeClient* method), 215  
 synchronize() (*d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 213  
 synchronizeResponse() (in module *d1\_client.cnclient\_2\_0.CoordinatingNodeClient\_2\_0* method), 213  
 sysmeta\_add\_blocked() (in module *d1\_common.replication\_policy*), 161  
 sysmeta\_add\_preferred() (in module *d1\_common.replication\_policy*), 161  
 sysmeta\_set\_default\_rp() (in module *d1\_common.replication\_policy*), 161  
 System Metadata, 33  
 systemMetadataChanged() (in module *d1\_client.mnclient.MemberNodeClient* method), 215  
 systemMetadataChangedResponse() (in module *d1\_client.mnclient.MemberNodeClient* method), 215  
 SystemMetadataIteratorMulti (class in *d1\_client.iter.sysmeta\_multi*), 193  
**T**  
 test\_dst\_mn() (*d1\_test.replication\_tester.replication\_tester.ReplicationTester* method), 230  
 test\_src\_mn() (*d1\_test.replication\_tester.replication\_tester.ReplicationTester* method), 230  
 TestHTTPServer (class in *d1\_test.replication\_tester.replication\_server*), 229  
 TLS, 34  
 to\_iso8601\_utc() (in module *d1\_common.date\_time*), 158  
 ToJsonCompatibleTypes (class in *d1\_common.util*), 182

`topological_sort()` (in module `d1_common.revision`), 169  
`transform()` (`d1_client.solr_client.SolrArrayTransformer` method), 220  
`transform()` (`d1_client.solr_client.SolrRecordTransformerBase` method), 220  
`trigger_by_header()` (in module `d1_test.mock_api.d1_exception`), 227  
`trigger_by_pid()` (in module `d1_test.mock_api.d1_exception`), 227  
`trigger_by_status_code()` (in module `d1_test.mock_api.d1_exception`), 227  
`ts_from_dt()` (in module `d1_common.date_time`), 154  
`ts_to_dt()` (in module `d1_common.cert.jwt`), 115  
`ts_to_str()` (in module `d1_common.cert.jwt`), 114  
`TYPE_NODE_TAG` (`d1_common.cert.subject_info.SubjectInfoNode` attribute), 119  
`tzname()` (`d1_common.date_time.FixedOffset` method), 153  
`tzname()` (`d1_common.date_time.UTC` method), 153  
`updateReplicationMetadata()` (`d1_client.cnclient.CoordinatingNodeClient` method), 210  
`updateReplicationMetadataResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 210  
`updateResponse()` (`d1_client.mnclient.MemberNodeClient` method), 215  
`updateSystemMetadata()` (`d1_client.baseclient_2_0.DataONEBaseClient_2_0` method), 199  
`updateSystemMetadataResponse()` (`d1_client.baseclient_2_0.DataONEBaseClient_2_0` method), 199  
`urlencode()` (in module `d1_common.url`), 179  
`UTC` (class in `d1_common.date_time`), 153  
`utcnow()` (in module `d1_common.date_time`), 157  
`utcoffset()` (`d1_common.date_time.FixedOffset` method), 153  
`utcoffset()` (`d1_common.date_time.UTC` method), 153

## U

`unpack_echo_header()` (in module `d1_test.mock_api.create`), 226  
`UnsupportedMetadataType`, 134  
`UnsupportedType`, 134  
`update()` (`d1_client.mnclient.MemberNodeClient` method), 215  
`update()` (`d1_common.wrap.access_policy.AccessPolicyWrapper` method), 141  
`update()` (`d1_test.slender_node_test_client.SlenderNodeTestClient` method), 233  
`update()` (in module `d1_common.wrap.access_policy`), 144  
`update_elements()` (in module `d1_common.system_metadata`), 172  
`updateAccount()` (`d1_client.cnclient.CoordinatingNodeClient` method), 205  
`updateAccountResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 205  
`updateGroup()` (`d1_client.cnclient.CoordinatingNodeClient` method), 209  
`updateGroupResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 209  
`updateNodeCapabilities()` (`d1_client.cnclient.CoordinatingNodeClient` method), 212  
`updateNodeCapabilitiesResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 211

## V

`v2_0_tag()` (in module `d1_common.type_conversions`), 178  
`validate_and_decode()` (in module `d1_common.cert.jwt`), 114  
`validate_bagit_file()` (in module `d1_common.bagit`), 149  
`Wrapper` specific extensions, 32  
`verifyAccount()` (`d1_client.cnclient.CoordinatingNodeClient` method), 206  
`verifyAccountResponse()` (`d1_client.cnclient.CoordinatingNodeClient` method), 206  
`VersionMismatch`, 135  
`view()` (`d1_client.cnclient_2_0.CoordinatingNodeClient_2_0` method), 214  
`view()` (`d1_client.mnclient_1_2.MemberNodeClient_1_2` method), 216  
`viewResponse()` (`d1_client.cnclient_2_0.CoordinatingNodeClient_2_0` method), 213  
`viewResponse()` (`d1_client.mnclient_1_2.MemberNodeClient_1_2` method), 215

## W

`Workspace`, 33  
`wrap()` (in module `d1_common.wrap.access_policy`), 141  
`wrap()` (in module `d1_common.wrap.simple_xml`), 145  
`wrap_sysmeta_pyxb()` (in module `d1_common.wrap.access_policy`), 141  
`WSGI`, 35

## X

X.509, [33](#)

`xml_is_dataone_exception()` (in module  
`dl_common.types.exceptions`), [131](#)

`xml_to_stabletree()` (in module  
`dl_test.xml_normalize`), [235](#)

`xsd_datetime_str_from_dt()` (in module  
`dl_common.date_time`), [155](#)