owmeta-core Documentation

Release 0.14.0.dev0

Mark Watts

Feb 05, 2023

CONTENTS

Our main README is available online on Github.[1] This documentation contains additional materials beyond what is covered there.

Contents:

[1] http://github.com/openworm/owmeta-core

CHAPTER

ONE

OWMETA_CORE

1.1 owmeta_core package

1.1.1 owmeta_core

owmeta-core is a platform for sharing relational data over the internet.

exception owmeta_core.**ConnectionFailError**(*cause*, **args*)

Bases: Exception

Thrown when a connection fails

class owmeta_core.**Connection**(*configFile=None*, *conf=None*, *mapper=None*)

Bases: object

Connection to an owmeta_core database. Essentially, wraps a *Data* object.

Load desired configuration and open the database

Parameters

configFile
[str, optional] The configuration file for owmeta_core.

conf
[dict, *Configuration*, *Data*, optional] A configuration object for the connection. Takes precedence over configFile

mapper
[*owmeta_core.mapper.Mapper*] Provides the mapper for this connection

Returns

Connection
connection wrapping the configuration

disconnect()

Close the database and stop listening to module loaders

transaction()

Context manager that executes the enclosed code in a transaction and then closes the connection. Provides the connection for binding with as.

identifier

Identifier for this connection.

Primarily, so that this Connection can be passed to contextualize for a Context

property transaction_manager

TransactionManager for the connection

owmeta_core.**connect**

alias of *Connection*

owmeta_core.**disconnect**(*c=None*)

Close the connection.

Deprecated: Just calls disconnect on the given connection

owmeta_core.**OWMETA_PROFILE_DIR = '~/.owmeta'**

Base directory in the user's profile for owmeta (e.g., shared configuration, bundle cache)

1.1.2 Subpackages

owmeta_core.bundle package

class owmeta_core.bundle.**AccessorConfig**

Bases: object

Configuration for accessing a *Remote*. Loaders are added to a remote according to which accessors are available

class owmeta_core.bundle.**Bundle**(*ident*, *bundles_directory='~/.owmeta/bundles'*, *version=None*, *conf=None*, *remotes=None*, *remotes_directory='~/.owmeta/remotes'*, *transaction_manager=None*)

Bases: object

Main entry point for using bundles

Typical usage is something like this:

```
>>> with Bundle('example/bundleId', version=42) as bnd:
...     for aDataObject in bnd(DataObject)().load():
...         # Do something with `aDataObject`
...         print(aDataObject)
DataObject(<http://example.org/entities#aDataObject>)
```

Note: Paths, bundles_directory and remotes_directory, will have symbolic links, environment variables, and "~" (for the current user's home directory) expanded when the *Bundle* is initialized. To reflect changes to symbolic links or home directories, the bundles_directory or remotes_directory attributes must be updated directly or a new instance must be created.

Parameters

ident
[str] Bundle ID

bundles_directory
[str, optional] Path to the bundles directory. Defaults to *DEFAULT_BUNDLES_DIRECTORY*

version
[int, optional] Bundle version to access. By default, the latest version will be used.

conf
[*Configuration* or dict, optional] Configuration to add to the one created for the bundle automatically. Values for the default imports context (*IMPORTS_CONTEXT_KEY*), the default context (*DEFAULT_CONTEXT_KEY*) and store (`'rdf.store'`, `'rdf.source'`, and, `'rdf.store_conf'`) will be ignored and overwritten.

remotes
[iterable of *Remote* or str, optional] A subset of remotes and additional remotes to fetch from. See *Fetcher.fetch*

remotes_directory
[str, optional] The directory to load *Remotes* from in case a bundle is not in the bundle cache. Defaults to *DEFAULT_REMOTES_DIRECTORY*

transaction_manager
[transaction.TransactionManager, optional] Transaction manager

initdb()

Initialize the bundle's conf *Data* instance

load_dependencies()

Load direct dependencies of this bundle

Yields

Bundle
A direct dependency of this bundle

load_dependencies_transitive()

Load dependencies from this bundle transitively

Yields

Bundle
A direct or indirect dependency of this bundle

connection

The owmeta_core connection to the bundle's indexed database

property contexts

List of str. Context IDs in this bundle

class owmeta_core.bundle.**BundleDependencyManager**(*dependencies*, ***common_bundle_arguments*)

Bases: object

Finds the bundle in which a context is defined.

For a given bundle graph, that there is *one* Bundle that "owns" a given context. Although multiple bundles may provide that context, the one closest to the root of the graph which provides some statements in that context is called the owner. Note that this does not mean that bundles on which the owner depends do not also be queried; however, the exact behavior is up to the component that uses this component.

Parameters

dependencies
[function] Function that returns a sequence of dependency descriptors

load_dependencies_transitive()

Load dependencies from this bundle transitively.

Any given version of a bundle will be yielded at most once regardless of how many times that version of the bundle appears in the dependency graph. Dependencies will yielded in topological sort order, so every dependency a Bundle declares will be yielded before any of its transitive dependencies.

Yields

Bundle
A direct or indirect dependency of this bundle

class owmeta_core.bundle.**BundleDependentStoreConfigBuilder**(*bundles_directory=None*, *remotes_directory=None*, *remotes=None*, *read_only=True*, *transaction_manager=None*)

Bases: object

Builds an RDFLib store configuration that depends on bundles.

The process of building the store configuration requires traversing the graph of dependencies so that duplicate dependencies in the graph can be omitted. To support this process, this builder will fetch bundles as needed to resolve transitive dependencies

build(*indexed_db_path*, *dependencies*, *bundle_directory=None*)

Builds the store configuration

Parameters

indexed_db_path
[str] Path to the indexed database of the store that depends on the listed dependenices

dependencies
[list of dict] List of dependencies' info, each entry including at least keys for 'id' and 'version'

bundle_directory
[str, optional] Path to the bundle directory for the dependent store, if the dependent store is a bundle. Used for information in an exceptional path, but not otherwise used

Returns

str
The type of the store. This is the name used to look up the RDFLib store plugin

object
The configuration for the store. This is the object that will be passed to rdflib.store.Store.open to configure the store.

class owmeta_core.bundle.**BundleTransactionManager**(*explicit=False*)

Bases: TransactionManager

Marker class useful in debugging to identify which txn manager we're using

class owmeta_core.bundle.**Cache**(*bundles_directory*)

Bases: object

Cache of bundles

Parameters

bundles_directory
[str] The where bundles are stored

`list`()

Returns a generator of summary bundle info

`class` owmeta_core.bundle.**Deployer**(*remotes=()*, ***kwargs*)

Bases: `_RemoteHandlerMixin`

Deploys bundles to *Remotes*.

A deployer takes a bundle directory tree or bundle archive and uploads it to a remote. *Fetcher* is, functionally, the dual of this class.

Deployer is responsible for selecting remotes and corresponding uploaders among a set of options. `Uploaders` are responsible for actually doing the upload.

deploy(*bundle_path*, *remotes=None*)

Deploy a bundle to *all* remotes that are configured to accept uploads

Parameters

bundle_path
[`str`] Path to a bundle directory tree or archive

remotes
[iterable `of` *Remote* or `str`] A subset of remotes to deploy to and additional remotes to deploy to

Raises

NoAcceptableUploaders
Thrown when none of the selected uploaders could upload the bundle

`class` owmeta_core.bundle.**Descriptor**(*ident*, ***kwargs*)

Bases: `object`

Descriptor for a bundle.

The descriptor is sufficient to build a distributable bundle directory tree from a `ConjunctiveGraph` and a set of files (see *Installer*).

dump(*output*)

Save a descriptor to a file as a YAML record

Parameters

output
[file object] The file to save the descriptor to

`classmethod` **load**(*descriptor_source*)

Load a descriptor from a YAML record

Parameters

descriptor_source
[`str` or file object] The descriptor source. Handled by yaml.safe_load

Raises

NotADescriptor
Thrown when the object loaded from `descriptor_source` isn't a `dict`

`classmethod` **make**(*obj*)

Makes a descriptor from the given object.

Parameters

obj
[a dict-like object] An object with parameters for the Descriptor. Typically a dict

Returns

Descriptor
The created descriptor

class owmeta_core.bundle.**Fetcher**(*bundles_root*, *remotes*, *transaction_manager=None*, ***kwargs*)

Bases: _RemoteHandlerMixin

Fetches bundles from *Remotes*

A fetcher takes a list of remotes, a bundle ID, and, optionally, a version number and downloads the bundle to a local directory. *Deployer* is, functionally, the dual of this class.

Parameters

bundles_root
[str] The root directory of the bundle cache

remotes
[list of *Remote* or str] List of pre-configured remotes used in calls to *fetch*

transaction_manager
[transaction.TransactionManager] Transaction manager to use when populating the indexed database after fetching

fetch(*bundle_id*, *bundle_version=None*, *remotes=None*, *progress_reporter=None*, *triples_progress_reporter=None*)

Retrieve a bundle by name from a remote and put it in the local bundle cache.

The first remote that can retrieve the bundle will be tried. Each remote will be tried in succession until one downloads the bundle.

Parameters

bundle_id
[str] The id of the bundle to retrieve

bundle_version
[int, optional] The version of the bundle to retrieve. If not provided, attempt to fetch the latest version available

remotes
[iterable of *Remote* or str] A subset of remotes and additional remotes to fetch from. If an entry in the iterable is a string, then it will be looked for amongst the remotes passed in initially.

progress_reporter
[tqdm.tqdm-like object, optional] Receives updates of progress in fetching and installing locally

triples_progress_reporter
[tqdm.tqdm-like object, optional] Receives updates of progress for adding triples for an individual graph

Returns

str
returns the directory where the bundle has been placed

Raises

exceptions.NoBundleLoader
Thrown when none of the loaders are able to download the bundle

FetchTargetIsNotEmpty
Thrown when the requested bundle is already in the cache

class owmeta_core.bundle.**FilesDescriptor**

Bases: object

Descriptor for files

class owmeta_core.bundle.**Installer**(*source_directory*, *bundles_directory*, *graph*, *imports_ctx=None*, *default_ctx=None*, *class_registry_ctx=None*, *installer_id=None*, *remotes=()*, *remotes_directory=None*)

Bases: object

Installs a bundle locally

Parameters

source_directory
[str] Directory where files come from. All files for a bundle must be below this directory

bundles_directory
[str] Directory where the bundles files go. Usually this is the bundle cache directory

graph
[rdflib.graph.ConjunctiveGraph] The graph from which we source contexts for this bundle

default_ctx
[str, optional] The ID of the default context – the target of a query when not otherwise specified.

imports_ctx
[str, optional] The ID of the imports context this installer should use. Imports relationships are selected from this graph according to the included contexts.

class_registry_ctx
[str, optional] The ID of the class registry context this installer should use. Class registry entries are retrieved from this graph.

installer_id
[iterable of *Remote* or str, optional] Name of this installer for purposes of mutual exclusion

remotes
[iterable of *Remote*, optional] Remotes to be used for retrieving dependencies when needed during installation. If not provided, the remotes will be collected from remotes_directory

remotes_directory
[str, optional] The directory to load *Remotes* from in case a bundle is not in the bundle cache. Defaults to *DEFAULT_REMOTES_DIRECTORY*

install(*descriptor*, *progress_reporter=None*)

Given a descriptor, install a bundle

Parameters

descriptor
[*Descriptor*] The descriptor for the bundle

progress_reporter
[tqdm.tqdm-like `object`] Used for reporting progress during installation. optional

Returns

`str`
The directory where the bundle is installed

Raises

`TargetIsNotEmpty`
Thrown when the target directory for installation is not empty.

class owmeta_core.bundle.**Remote**(*name*, *accessor_configs=()*)

Bases: `object`

A place where bundles come from and go to

Parameters

name
[`str`] The name of the remote

accessor_configs
[iterable `of` *`AccessorConfig`*] Configs for how you access the remote

add_config(*accessor_config*)

Add the given accessor config to this remote

Parameters

accessor_config
[*`AccessorConfig`*] The config to add

Returns

bool
`True` if the accessor config was added (meaning there's no equivalent one already set for this remote). Otherwise, `False`.

generate_loaders()

Generate the bundle loaders for this remote.

Loaders are generated from *`accessor_configs`* and `LOADER_CLASSES` according with which type of *`Loader`* can load a type of accessor

generate_uploaders()

Generate the bundle uploaders for this remote

classmethod read(*inp*)

Read a serialized *`Remote`*

Parameters

inp
[file object] File-like object containing the serialized *`Remote`*

write(*out*)

Serialize the *`Remote`* and write to `out`

Parameters

out
[file object] Target for writing the remote

accessor_configs

Configs for how you access the remote.

One might configure mirrors or replicas for a given bundle repository as multiple accessor configs

file_name

If read from a file, the remote should have this attribute set to its source file's path

name

Name of the remote

class owmeta_core.bundle.**URLConfig**(*url*)

Bases: *AccessorConfig*

Configuration for accessing a remote with just a URL.

Note that URLConfigs should be pickle-able since they are written to a YAML file as part of the *Remote* they're apart of.

owmeta_core.bundle.**build_indexed_database**(*dest*, *bundle_directory*, *transaction_manager*, *progress=None*, *trip_prog=None*)

Build the indexed database from a bundle directory

owmeta_core.bundle.**retrieve_remotes**(*remotes_dir*, *load_entry_points=True*)

Retrieve remotes from a project directory or user remotes directory

Parameters

owmdir
[str] path to the project directory

load_entry_points
[bool, optional] if True, then the entry points for *Loader* and *Uploader* implementations that have been added as entry points

owmeta_core.bundle.**DEFAULT_BUNDLES_DIRECTORY = '~/.owmeta/bundles'**

Default directory for the bundle cache

owmeta_core.bundle.**DEFAULT_REMOTES_DIRECTORY = '~/.owmeta/remotes'**

Default directory for descriptors of user-level remotes as opposed to project-specific remotes

owmeta_core.bundle.**URL_CONFIG_MAP = {'file': <class 'owmeta_core.bundle.loaders.local.FileURLConfig'>, 'http': <class 'owmeta_core.bundle.loaders.http.HTTPURLConfig'>, 'https': <class 'owmeta_core.bundle.loaders.http.HTTPSURLConfig'>}**

URLConfigs by scheme. Can be populated by pkg_resources entry points

Subpackages

owmeta_core.bundle.loaders package

Package for uploaders and downloaders of bundles

exception owmeta_core.bundle.loaders.**LoadFailed**(*bundle_id*, *loader*, **args*)

Bases: Exception

Thrown when a bundle could not be downloaded

Parameters

bundle_id
[str] ID of the bundle on which a download was attempted

loader
[*Loader*] The loader that attempted to download the bundle

args[0]
[str] Explanation of why the download failed

***args[1:]**
Passed on to Exception

class owmeta_core.bundle.loaders.**Loader**

Bases: object

Downloads bundles into the local index and caches them

Note that a *Loader* is transient: it will be created when needed to download *one* bundle and then discarded. Any state that should be carried from request to request should be attached to an AccessorConfig

Attributes

base_directory
[str] The path where the bundle archive should be unpacked

bundle_versions(*bundle_id*)

List the versions available for the bundle.

This is a required part of the *Loader* interface.

Parameters

bundle_id
[str] ID of the bundle for which versions are requested

Returns

A list of int
Each entry is a version of the bundle available via this loader

can_load(*bundle_id*, *bundle_version=None*)

Returns True if the bundle named bundle_id is available.

This method is for loaders to determine that they probably can or cannot load the bundle, such as by checking repository metadata. Other loaders that return True from *can_load* should be tried if a given loader fails, but a warning should be recorded for the loader that failed.

classmethod can_load_from(*accessor_config*)

Returns True if the given accessor_config is a valid config for this loader

Parameters

accessor_config
[AccessorConfig] The config which we may be able to load from

load(*bundle_id*, *bundle_version=None*)

Load the bundle into the local index

Parameters

bundle_id
[str] ID of the bundle to load

bundle_version
[int] Version of the bundle to load. Defaults to the latest available. optional

Raises

LoadFailed
Raised when the bundle cannot be loaded

class owmeta_core.bundle.loaders.**Uploader**

Bases: object

Uploads bundles to remotes

can_upload(*bundle_path*)

Returns True if this uploader can upload this bundle

Parameters

bundle_path
[str] The file path to the bundle to upload

classmethod can_upload_to(*accessor_config*)

Returns True if this uploader can upload with the given accessor configuration

Parameters

accessor_config
[AccessorConfig]

upload(*bundle_path*)

Upload a bundle

Parameters

bundle_path
[str] The file path to the bundle to upload

Submodules

owmeta_core.bundle.loaders.http module

exception owmeta_core.bundle.loaders.http.**IndexLoadFailed**(*response*)

Bases: Exception

Thrown when the HTTP bundle loader cannot get its index

class owmeta_core.bundle.loaders.http.**HTTPBundleLoader**(*index_url*, *cachedir=None*, *hash_preference=('sha224',)*, ***kwargs*)

Bases: *Loader*

Loads bundles from HTTP(S) resources listed in an index file

Parameters

index_url
[str or *owmeta_core.bundle.URLConfig*] URL for the index file pointing to the bundle archives

cachedir
[str, optional] Directory where the index and any downloaded bundle archive should be cached. If provided, the index and bundle archive is cached in the given directory. If not provided, the index will be cached in memory and the bundle will not be cached.

hash_preference
[tuple of str] Preference ordering of hashes to use for checking integrity of files. If none match in the preference ordering, then the first one

****kwargs**
Passed on to *Loader*

`can_load`(*bundle_id*, *bundle_version=None*)

Check the index for an entry for the bundle.

- If a version is given and the index has an entry for the bundle at that version and that entry gives a URL for the bundle, then we return `True`.
- If no version is given and the index has an entry for the bundle at any version and that entry gives a URL for the bundle, then we return `True`.
- Otherwise, we return `False`

Parameters

bundle_id
[str] ID of the bundle to look for

bundle_version
[int, optional] Version number of the bundle to look for. If not provided, then any version is deemed acceptable

Returns

bool
`True` if the bundle can be loaded; otherwise, `False`

`classmethod can_load_from`(*ac*)

Returns `True` for `http://` or `https://` URLConfigs

Parameters

ac
[AccessorConfig] The config which we may be able to load from

`class owmeta_core.bundle.loaders.http.HTTPBundleUploader`(*upload_url*, *ssl_context=None*, *max_retries=1*)

Bases: *Uploader*

Uploads bundles by sending bundle archives in HTTP POST requests

Parameters

upload_url
[str or URLConfig] URL string or accessor config

ssl_context
[ssl.SSLContext, optional] SSL/TLS context to use for the connection. Overrides any context provided in `upload_url`

max_retries
[int, optional] Maximum number of times to retry the upload after a failure.

upload(*bundle_path*)

Attempt to upload the bundle. Retries will be attempted when `BrokenPipeError` is thrown by the http client

class owmeta_core.bundle.loaders.http.**HTTPSURLConfig**(**args*, *ssl_context_provider=None*, *ssl_context=None*, ***kwargs*)

Bases: *HTTPURLConfig*

HTTPS URL configuration

Parameters

***args**
Passed on to HTTPURLConfig

ssl_context_provider
[`str`] Path to a callable that provides a `ssl.SSLContext`. See *https_remote*

ssl_context
[`ssl.SSLContext`] The SSL/TLS context to use for uploading with this accessor

****kwargs**
Passed on to HTTPURLConfig

class owmeta_core.bundle.loaders.http.**HTTPURLConfig**(**args*, *session_file_name=None*, *session_provider=None*, *cache_dir=None*, *mem_cache=False*, ***kwargs*)

Bases: *URLConfig*

HTTP URL configuration

Parameters

***args**
Passed on to URLConfig

session_file_name
[`str`, `optional`] Session file name

session_provider
[`str`, `optional`] Provider path for a callable that returns a session

cache_dir
[`str`, `optional`] HTTP cache directory. Supersedes `mem_cache`

mem_cache
[bool, `optional`] Whether to use an in-memory cache. Superseded by `cache_dir`

****kwargs**
Passed on to URLConfig

init_session()

Initialize the HTTP session. Typically you won't call this, but will just access *session*

property session

A `requests.Session`

This will be loaded from `session_file_name` if a value is set for that. Otherwise, the session will either be obtained from the `session_provider` or a default session will be created; in either case, any response caching configuration will be applied.

owmeta_core.bundle.loaders.http.**http_remote**(*self*, *, *cache=None*, *session_provider=None*, *session_file_name=None*)

Provide additional parameters for HTTP remote accessors

Parameters

cache
[str] Either the string "mem" or a file path to a cache directory

session_provider
[str] Path to a callable that provides a requests.Session. The format is similar to that for setuptools entry points: path.to.module:path.to.provider.callable. Notably, there's no name and "extras" are not supported. optional.

session_file_name
[str] Path to a file where the HTTP session can be stored

owmeta_core.bundle.loaders.http.**https_remote**(*self*, *, *ssl_context_provider=None*, *cache=None*, *session_provider=None*, *session_file_name=None*)

Provide additional parameters for HTTPS remote accessors

Parameters

ssl_context_provider
[str] Path to a callable that provides a ssl.SSLContext used for bundle uploads. The format is similar to that for setuptools entry points: path.to.module:path.to.provider.callable. Notably, there's no name and "extras" are not supported. optional.

cache
[str] Either the string "mem" or a file path to a cache directory

session_provider
[str] Path to a callable that provides a requests.Session. The format is similar to that for setuptools entry points: path.to.module:path.to.provider.callable. Notably, there's no name and "extras" are not supported. optional.

session_file_name
[str] Path to a file where the HTTP session can be stored

owmeta_core.bundle.loaders.local module

class owmeta_core.bundle.loaders.local.**FileBundleLoader**(*source_bundles_dir*)

Bases: *Loader*

Copies bundles from a local directory structure identical to the local bundle cache typically stored under ~/.owmeta/bundles.

Note, there is no corresponding bundle uploader: if you want that, you should instead *fetch* the bundle into the target bundle cache directory.

can_load(*bundle_id*, *bundle_version=None*)

Check if the bundle is available under the base directory given at init

classmethod can_load_from(*ac*)

Returns True for file:// URLConfigs

Parameters

ac
[AccessorConfig] The config which we may be able to load from

class owmeta_core.bundle.loaders.local.**FileURLConfig**(*url*)

Bases: *URLConfig*

URL config for local files.

Local file paths, in general, are not especially portable, but this accessor config may be useful for bundle directories on shared file systems like NFS or Samba.

owmeta_core.bundle.loaders.sftp module

Submodules

owmeta_core.bundle.archive module

exception owmeta_core.bundle.archive.**ArchiveTargetPathDoesNotExist**

Bases: Exception

Thrown when the *Archiver* target path does not exist

exception owmeta_core.bundle.archive.**TargetDirectoryMismatch**(*target_directory*, *expected_target_directory*)

Bases: *UnarchiveFailed*

Thrown when the target path doesn't agree with the bundle manifest

exception owmeta_core.bundle.archive.**UnarchiveFailed**

Bases: Exception

Thrown when an *Unarchiver* fails for some reason not covered by other

class owmeta_core.bundle.archive.**ArchiveExtractor**(*targetdir*, *tarfile*)

Bases: object

Extracts tarfile archives

Parameters

targetdir
[str] The directory to which the archive will be extracted

tarfile
[tarfile.TarFile] The file to extract

extract()

Extract the tarfile to the target directory

class owmeta_core.bundle.archive.**Archiver**(*target_directory*, *bundles_directory=None*)

Bases: object

Archives a bundle directory tree

Parameters

target_directory
[str] Where to place archives.

bundles_directory
[str, optional] Where the bundles are. If not provided, then this archiver can only pack bundles when given a specific bundle's directory

pack(*bundle_id=None*, *version=None*, ***, *bundle_directory=None*, *target_file_name=None*)

Pack an installed bundle into an archive file

Parameters

bundle_id
[str, optional] ID of the bundle to pack. If omitted, the bundle_directory must be provided

version
[int, optional] Bundle version

bundle_directory
[str, optional] Bundle directory. If omitted, bundle_id must be provided. If provided, bundle_id and version are ignored

target_file_name
[str, optional] Name of the archive file. If not provided, the name will be 'bundle.tar.xz' and will placed in the target_directory. Relative paths are relative to target_directory

Raises

BundleNotFound
Thrown when the bundle with the given ID cannot be found, or cannot be found at the demanded version

ArchiveTargetPathDoesNotExist
Thrown when the path to the desired target file does not exist

class owmeta_core.bundle.archive.**Unarchiver**(*bundles_directory=None*)

Bases: object

Unpacks an archive file (e.g., a tar.xz) of a bundle

Parameters

bundles_directory
[str, optional] The directory under which bundles should be unpacked. Typically the bundle cache directory.

classmethod manifest(*bundle_tarfile*, *input_file=None*)

Get the manifest file from a bundle archive

Parameters

bundle_tarfile
[tarfile.TarFile] Tarfile, ostensibly containing bundle data

input_file
[file object or str, optional] Name of the tar file. Will attempt to extract it from the tarfile if not given

unpack(*input_file*, *target_directory=None*)

Unpack the archive file

If target_directory is provided, and bundles_directory is provided at initialization, then if the bundle manifest doesn't match the expected archive path, then an exception is raised.

Parameters

input_file
[str or file object] The archive file

target_directory
[str, optional] The path where the archive should be unpacked. If this argument is not provided, then the target directory is derived from `bundles_directory` (see `fmt_bundle_directory`)

Raises

NotABundlePath
Thrown in one of these conditions:

- If the `input_file` is not in an expected format (lzma-zipped TAR file)
- If the `input_file` does not have a "manifest" file
- If the `input_file` manifest file is invalid or is not a regular file (see `validate_manifest` for further details)
- If the `input_file` is a file path and the corresponding file is not found

TargetDirectoryMismatch
Thrown when both a `bundles_directory` has been set at initialization and a `target_directory` is passed to this method and the path under `bundles_directory` indicated by the manifest in the `input_file` does not agree with `target_directory`

owmeta_core.bundle.archive.**ensure_archive**(*bundle_path*)

Produce an archive path from a bundle path whether the given path is an archive or not

Parameters

bundle_path
[str] The path to a bundle directory or archive

owmeta_core.bundle.common module

owmeta_core.bundle.common.**bundle_tree_filter**(*path*, *fullpath*)

Returns true for file names that are to be included in a bundle for deployment or fetching.

Parameters

path
[str] The relative path of the file to check

fillpath
[str] The full path of the file to check (usable for deeper inspection)

owmeta_core.bundle.common.**fmt_bundle_directory**(*bundles_directory*, *ident*, *version=None*)

Get the directory for the given bundle identifier and version

Parameters

ident
[str] Bundle identifier

version
[int] Version number. If not provided, returns the directory containing all of the versions

owmeta_core.bundle.common.**validate_manifest**(*bundle_path*, *manifest_data*)

Validate manifest data in a dict

Parameters

bundle_path
[str] The path to the bundle directory or archive. Used in the exception message if the manifest data is invalid

manifest_data
[dict] The data from a manifest file

Raises

NotABundlePath
Thrown in one of these conditions:

- manifest_data lacks a manifest_version
- manifest_data has a manifest_version > BUNDLE_MANIFEST_VERSION
- manifest_data has a manifest_version <= 0
- manifest_data lacks a version
- manifest_data lacks an id

owmeta_core.bundle.common.**BUNDLE_ARCHIVE_MIME_TYPE = 'application/x-gtar'**
MIME type for bundle archive files

owmeta_core.bundle.common.**BUNDLE_INDEXED_DB_NAME = 'owm.db'**
Base name of the indexed database that gets built in a bundle directory during installation

owmeta_core.bundle.common.**BUNDLE_MANIFEST_FILE_NAME = 'manifest'**
Name of the manifest file in a bundle directory or archive

owmeta_core.bundle.common.**BUNDLE_MANIFEST_VERSION = 1**
Current version number of the bundle manifest. Written by Installer and anticipated by Deployer and Fetcher.

owmeta_core.bundle.exceptions module

exception owmeta_core.bundle.exceptions.**BundleNotFound**(*bundle_id*, *msg=None*, *version=None*)
Bases: Exception

Thrown when a bundle cannot be found on a local or remote resource with the given parameters.

Parameters

bundle_id
[str] ID of the bundle that was sought

msg
[str, optional] An explanation of why the bundle could not be found

version
[int, optional] Version number of the bundle

exception owmeta_core.bundle.exceptions.**CircularDependencyDetected**
Bases: Exception

Thrown when a circular dependency is detected in the bundle dependency graph

exception owmeta_core.bundle.exceptions.**DeployFailed**
Bases: Exception

Thrown when bundle deployment fails for an apparently valid bundle

exception owmeta_core.bundle.exceptions.**FetchFailed**

Bases: Exception

Generic message for when a fetch fails

exception owmeta_core.bundle.exceptions.**FetchTargetIsNotEmpty**(*target*)

Bases: *FetchFailed*

Thrown when the target directory of a fetch is not empty

exception owmeta_core.bundle.exceptions.**InstallFailed**

Bases: Exception

Thrown when a bundle installation fails to complete.

You can assume that any intermediate bundle files have been cleaned up from the bundle cache

exception owmeta_core.bundle.exceptions.**MalformedBundle**(*path*, *explanation*)

Bases: *NotABundlePath*

Thrown when a given path does points to a bundle directory or archive is malformed

exception owmeta_core.bundle.exceptions.**NoAcceptableUploaders**(*bundle_path*)

Bases: *DeployFailed*

Thrown when, for all selected `Remotes`, no `Uploaders` report that they can upload a given bundle

exception owmeta_core.bundle.exceptions.**NoBundleLoader**(*bundle_id*, *bundle_version=None*, *message=None*)

Bases: *FetchFailed*

Thrown when a loader can't be found for a bundle

exception owmeta_core.bundle.exceptions.**NoRemoteAvailable**

Bases: Exception

Thrown when we need a remote and we don't have one

exception owmeta_core.bundle.exceptions.**NotABundlePath**(*path*, *explanation*)

Bases: Exception

Thrown when a given path does not point to a valid bundle directory tree or bundle archive

exception owmeta_core.bundle.exceptions.**NotADescriptor**

Bases: Exception

Thrown when a given file, string, or other object is offered as a descriptor, but does not represent a `Descriptor`

exception owmeta_core.bundle.exceptions.**TargetIsNotEmpty**(*target*)

Bases: *InstallFailed*

Thrown when the target directory of an installation is not empty

exception owmeta_core.bundle.exceptions.**UncoveredImports**(*imports*)

Bases: *InstallFailed*

Thrown when a bundle to be installed has declared imports but is missing dependencies to cover those imports

Parameters

imports
[`list of URIRef`] List of imports declared for a bundle which are not covered by any of the bundle's dependencies

owmeta_core.commands package

Various commands of the same kind as *OWM*, mostly intended as sub-commands of `OWM`.

Submodules

owmeta_core.commands.bundle module

Bundle commands

exception owmeta_core.commands.bundle.**BundleNotFound**(*bundle_id*, *bundle_version=None*)

Bases: *GenericUserError*

Thrown when a bundle cannot be found with the requested ID and version

exception owmeta_core.commands.bundle.**NoBundleLoader**(*bundle_id*, *bundle_version=None*)

Bases: *GenericUserError*

Thrown when a loader can't be found for a bundle

class owmeta_core.commands.bundle.**OWMBundle**(*parent*)

Bases: `object`

Bundle commands

checkout(*bundle_id*)

Switch to the named bundle

Parameters

bundle_id
[`str`] ID of the bundle to switch to

deploy(*bundle_id*, *version=None*, *remotes=None*)

Deploys a bundle to a remote. The target remotes come from project and user settings or, if provided, the `remotes` parameter

Parameters

bundle_id
[`str`] ID of the bundle to deploy

version
[`int`] Version of the bundle to deploy. optional.

remotes
[`str`] Names of the remotes to deploy to. optional.

deregister(*bundle_id*)

Remove a bundle from the project

Parameters

bundle_id
[`str`] The id of the bundle to deregister

fetch(*bundle_id*, *bundle_version=None*, *bundles_directory=None*)

Retrieve a bundle by id from a remote and put it in the local bundle index and cache

Parameters

bundle_id
[str] The id of the bundle to retrieve.

bundle_version
[int] The version of the bundle to retrieve. optional

bundles_directory
[str] Root directory of the bundles cache. optional: uses the default bundle cache in the user's home directory if not provided

`install`(*bundle*)

Install the bundle to the local bundle repository for use across projects on the same machine

Parameters

bundle
[str] ID of the bundle to install or path to the bundle descriptor

`list`()

List registered bundles in the current project.

To list bundles within the local repo or a remote repo, use the `cache list` sub-command.

`load`(*input_file_name*)

Load a bundle from a file and register it into the project

Parameters

input_file_name
[str] The source file of the bundle

`register`(*descriptor*)

Register a bundle within the project

Registering a bundle adds it to project configuration and records where the descriptor file is within the project's working tree. If the descriptor file moves it must be re-registered at the new location.

Parameters

descriptor
[str] Descriptor file for the bundle

`save`(*bundle_id*, *output*, *bundle_version=None*)

Write an installed bundle to a file

Writing the bundle to a file means writing the bundle manifest, constituent graphs, and attached files to an archive. The bundle can be in the local bundle repository, a remote, or registered in the project.

Parameters

bundle_id
[str] The bundle to save

output
[str] The target file

bundle_version
[int] Version of the bundle to write. optional: defaults to the latest installed bundle

`cache`

OWMBundleCache: Bundle cache commands

remote

OWMBundleRemote: Commands for dealing with bundle remotes

class owmeta_core.commands.bundle.**OWMBundleCache**(*parent*)

Bases: object

Bundle cache commands

list()

List bundles in the cache

class owmeta_core.commands.bundle.**OWMBundleRemote**(*parent*)

Bases: object

Commands for dealing with bundle remotes

list()

List remotes

remove(*name*)

Remove the remote

Parameters

name
[str] Name of the remote

show(*name*)

Show details about a remote

Parameters

name
[str] Name of the remote

add

OWMBundleRemoteAdd: Add a remote and, optionally, an accessor to that remote.

Remotes contain zero or more "accessor configurations" which describe how to upload to and download from a remote. Sub-commands allow for specifying additional parameters specific to a type of accessor.

update

OWMBundleRemoteUpdate: Update a remote accessor

Remotes contain zero or more "accessor configurations" which describe how to upload to and download from a remote. Sub-commands allow for specifying additional parameters specific to a type of accessor.

user

If this option is provided, then remotes in the user profile directory are used rather than those in the project directory.

class owmeta_core.commands.bundle.**OWMBundleRemoteAdd**(*parent*)

Bases: _OWMBundleRemoteAddUpdate

Add a remote and, optionally, an accessor to that remote.

Remotes contain zero or more "accessor configurations" which describe how to upload to and download from a remote. Sub-commands allow for specifying additional parameters specific to a type of accessor.

class owmeta_core.commands.bundle.**OWMBundleRemoteUpdate**(*parent*)

Bases: _OWMBundleRemoteAddUpdate

Update a remote accessor

Remotes contain zero or more "accessor configurations" which describe how to upload to and download from a remote. Sub-commands allow for specifying additional parameters specific to a type of accessor.

owmeta_core.data_trans package

Data translators

Some *DataSource* and *DataTranslator* types. Some deal with generic file types (e.g., comma-separated values) while others are specific to the format of a kind of file housed in owmeta.

Submodules

owmeta_core.data_trans.common_data module

Variables common to several *DataSource* and *DataTranslator* implementations

owmeta_core.data_trans.common_data.**DS_DATA_NS** = **Namespace('http://data.openworm.org/data_sources/')**

Namespace for data sources in owmeta-core. Not for use by packages downstream of owmeta-core

owmeta_core.data_trans.common_data.**DS_NS** = **Namespace('http://schema.openworm.org/2020/07/data_sources/')**

Namespace for data sources in owmeta-core. Not for use by packages downstream of owmeta-core

owmeta_core.data_trans.common_data.**TRANS_NS** = **Namespace('http://schema.openworm.org/2020/07/translators/')**

Namespace for translators in owmeta-core. Not for use by packages downstream of owmeta-core

owmeta_core.data_trans.context_datasource module

class owmeta_core.data_trans.context_datasource.**VariableIdentifierContext**(**args*, ***kwargs*)

Bases: *VariableIdentifierMixin*, *Context*

A Context that gets its identifier and its configuration from its 'maker' passed in at initialization

Parameters

maker
[object] An object with an `identifier` attribute

maker.identifier
[rdflib.term.URIRef] A URI that will serve as the identifier for the *VariableIdentifierMixin*

class owmeta_core.data_trans.context_datasource.**VariableIdentifierContextDataObject**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *VariableIdentifierMixin*, *ContextDataObject*

A ContextDataObject that gets its identifier and its configuration from its 'maker' passed in at initialization

Parameters

maker
[object] An object with an `identifier` attribute

maker.identifier
[rdflib.term.URIRef] A URI that will serve as the identifier for the *VariableIdentifierMixin*

class owmeta_core.data_trans.context_datasource.**VariableIdentifierMixin**(*maker=None*, ***kwargs*)

Bases: object

A mix-in class that takes its identifier from its 'maker' passed in at initialization.

Parameters

maker
[object] An object with an `identifier` attribute

maker.identifier
[rdflib.term.URIRef] A URI that will serve as the identifier for the *VariableIdentifierMixin*

owmeta_core.data_trans.csv_ds module

class owmeta_core.data_trans.csv_ds.**CSVDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *LocalFileDataSource*

A CSV file data source

Parameters

commit_op
[CommitOp, optional] The operation to use for commiting the file changes. The default is `COPY`

csv_field_delimiter

"CSV field delimiter", a *DatatypeProperty*

Default value: ','

csv_file_name

"CSV file name", a *DatatypeProperty*

csv_header

"Header column names", a *DatatypeProperty*

class owmeta_core.data_trans.csv_ds.**CSVDataTranslator**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataTranslator*

A data translator which handles CSV files

make_reader(*source*, *skipheader=True*, *dict_reader=False*, *skiplines=0*, ***kwargs*)

Make a CSV reader

Parameters

source
[*CSVDataSource*] The data source to read from

skipheader
[bool] If true, the first line read of the CSV file after the reader is created will not be returned from the reader

dict_reader
[bool] If true, the reader will be a `DictReader`

skiplines
[int] A number of lines to skip before creating the reader. Useful if the CSV file contains some commentary or other 'front matter'

****kwargs**
Remaining arguments passed on to `reader` or `DictReader`

reader(*source*, *skipheader=True*, *dict_reader=False*, *skiplines=0*, ***kwargs*)

Alias to *make_reader*

class owmeta_core.data_trans.csv_ds.**CSVHTTPFileDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *HTTPFileDataSource*

A CSV file retrieved over HTTP

csv_field_delimiter

"CSV field delimiter", a *DatatypeProperty*

Default value: ','

csv_header

"Header column names", a *DatatypeProperty*

owmeta_core.data_trans.excel_ds module

class owmeta_core.data_trans.excel_ds.**XLSXHTTPFileDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *HTTPFileDataSource*

URL
[*DatatypeProperty*] Attribute: `url`

MD5 hash
[*DatatypeProperty*] Attribute: `md5`

SHA-256 hash
[*DatatypeProperty*] Attribute: `sha256`

SHA-512 hash
[*DatatypeProperty*] Attribute: `sha512`

Input source
[*ObjectProperty*] Attribute: `source`

The data source that was translated into this one

Transformation
[*ObjectProperty*] Attribute: `transformation`

Information about the transformation process that created this object

Translation
[*ObjectProperty*] Attribute: `translation`

Information about the translation process that created this object

Description
[*DatatypeProperty*] Attribute: `description`

Free-text describing the data source

owmeta_core.data_trans.file_ds module

class owmeta_core.data_trans.file_ds.**FileDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataSource*

This DataSource represents a "file", essentially a sequence of bytes with a name

Attributes

source_file_path
[path-like object] The file to commit for this datasource

file_contents()

Returns a file object for reading data from the file

update_hash(*algorithm*)

Set a message digest property for the file

Parameters

algorithm
[str] The name of the property and algorithm to update

md5

"MD5 hash", a *DatatypeProperty*

sha256

"SHA-256 hash", a *DatatypeProperty*

sha512

"SHA-512 hash", a *DatatypeProperty*

owmeta_core.data_trans.http_ds module

class owmeta_core.data_trans.http_ds.**HTTPFileDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *FileDataSource*

URL
[*DatatypeProperty*] Attribute: *url*

MD5 hash
[*DatatypeProperty*] Attribute: `md5`

SHA-256 hash
[*DatatypeProperty*] Attribute: `sha256`

SHA-512 hash
[*DatatypeProperty*] Attribute: `sha512`

Input source
[*ObjectProperty*] Attribute: `source`

The data source that was translated into this one

Transformation
[*ObjectProperty*] Attribute: `transformation`

Information about the transformation process that created this object

Translation
[*ObjectProperty*] Attribute: `translation`

Information about the translation process that created this object

Description
[*DatatypeProperty*] Attribute: `description`

Free-text describing the data source

url

"URL", a *DatatypeProperty*

owmeta_core.data_trans.local_file_ds module

class owmeta_core.data_trans.local_file_ds.**CommitOp**(*value*)

Bases: Enum

Indicates which operation to perform for "commiting" a local file. See *LocalFileDataSource*.

COPY = 2

copy the source file contents to the target file

HARDLINK = 4

create a hard-link to the file. This will not be valid in case the source and target file are on different file systems.

RENAME = 1

rename the source file to the target file

SYMLINK = 3

create a symbolic link to the file. This may not be allowed for unprivileged users on Windows machines

class owmeta_core.data_trans.local_file_ds.**LocalFileDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *CapableConfigurable*, *FileDataSource*

File paths should be relative – in general, path names on a given machine are not portable

Attributes

commit_op
[*CommitOp*] The operation to use for commiting the file changes

Parameters

commit_op
[*CommitOp*, `optional`] The operation to use for commiting the file changes. The default is *COPY*

after_transform()

"Commits" the file by applying the operation indicated by `commit_op` to `source_file_path` so that it is accessible at *full_path*

file_contents()

Returns an open file to be read from at <full_path>/<file_name>

This file should be closed when you are done with it. It may be used as a context manager

file_output()

Returns an open file to be written to at <full_path>/<file_name>

This file should be closed when you are done with it. It may be used as a context manager

full_output_path()

Returns the full output path to the file

full_path()

Returns the full path to the file

file_name

"File name", a *DatatypeProperty*

torrent_file_name

"Torrent file name", a *DatatypeProperty*

1.1.3 Submodules

owmeta_core.agg_store module

exception owmeta_core.agg_store.**UnsupportedAggregateOperation**

Bases: Exception

Thrown for operations which modify a graph and hence are inappropriate for *AggregateStore*

class owmeta_core.agg_store.**AggregateStore**(*configuration=None*, *identifier=None*, *graph_aware=None*)

Bases: Store

A read-only aggregate of RDFLib stores

open(*configuration*, *create=True*)

Creates and opens all of the stores specified in the configuration

Also checks for all aggregated stores to be *context_aware*

context_aware = True

Specified by RDFLib. Required to be True for ConjunctiveGraph stores.

Aggregated stores MUST be context-aware. This is enforced by *open()*.

graph_aware = True

Specified by RDFLib. Required to be True for Dataset stores.

The first store must be graph-aware. This is enforced by *open()*.

owmeta_core.bittorrent module

owmeta_core.bundle_dependency_store module

class owmeta_core.bundle_dependency_store.**BundleDependencyStore**(*wrapped=None*, *excludes=()*)

Bases: Store

A read-only RDFLib Store that supports the extra stuff we need from dependencies

open(*configuration*)

Creates and opens the configured store.

Also verifies that the provided store is context-aware

context_aware = True

Specified by RDFLib. Required to be True for ConjunctiveGraph stores.

Wrapped store MUST be context-aware. This is enforced by *open()*.

class owmeta_core.bundle_dependency_store.**StoreCache**

Bases: object

Cache of stores previously cached by a *BDS*.

We don't want to keep hold of a store if there's no BDS using it, so we only reference the stores weakly.

owmeta_core.capabilities module

class owmeta_core.capabilities.**CacheDirectoryCapability**(**args*, ***kwargs*)

Bases: *Capability*

Capability that provides a cache directory.

The provider of this capability must be capable of persisting effectively distinct directories for each Capable which needs this capability. The provider must permit depositing files in the directory by the current effective user.

class owmeta_core.capabilities.**CacheDirectoryProvider**

Bases: *Provider*

Provides the *CacheDirectoryCapability*

cache_directory(*cache_key*)

Return the cache directory path

Parameters

cache_key
[str] The key for the cache entry

Returns

str
The cache directory

clear(*cache_key*)

Clear the cache directory for the Capable.

Should remove the directory itself, if possible.

class owmeta_core.capabilities.**FilePathCapability**(**args*, ***kwargs*)

Bases: *Capability*

Provides a file path where named files can be retrieved.

This capability may be needed when files are referred to that aren't necessarily stored on the local machine, or which on the local machine, but only in non-portable locations (e.g., a home directory).

class owmeta_core.capabilities.**FilePathProvider**

Bases: *Provider*

Provides the *FilePathCapability*

file_path()

The needed file path

class owmeta_core.capabilities.**OutputFilePathCapability**(**args*, ***kwargs*)

Bases: *Capability*

Provides a file path where named files can be put

class owmeta_core.capabilities.**OutputFilePathProvider**

Bases: *Provider*

Provides the *OutputFilePathCapability*

output_file_path()

The needed file path

class owmeta_core.capabilities.**TemporaryDirectoryCapability**(**args*, ***kwargs*)

Bases: *Capability*

Provides new, empty temporary directories

class owmeta_core.capabilities.**TemporaryDirectoryProvider**

Bases: *Provider*

Provides the *TemporaryDirectoryCapability*

temporary_directory()

Return the path of a new, empty temporary directory. The receiver of the temporary directory should delete the directory when they're done with it.

Returns

str
The temporary directory path

owmeta_core.capability module

Defines 'capabilities', pieces of functionality that an object needs which must be injected. The receiver of the capability is called a *capable*.

A given capability can be provided by more than one capability provider, but, for a given set of providers, only one will be bound at a time. Logically, each provider that provides the capability is asked, in a user-provided preference order, whether it can provide the capability for the *specific* capable and the first one which can provide the capability is bound to the object.

The core idea is dependency injection: a capability does not modify the capable: the capable receives the provider and a reference to the capability provided, but how the capable uses the provider is up to the capable. This is important

because the user of the capable should not condition its behavior on the particular capability provider used, although it may change its behavior based on which capabilities the capable has.

Note, that there may be some providers that lose their ability to provide a capability after they have been bound to a capable. This loss should be communicated with a *CannotProvideCapability* exception when the relevant methods are called on the provider. This *may* allow certain operations to be retried with a provider lower on the capability order, *but* a provider that throws *CannotProvideCapability* may validly be asked if it can provide the capability again – if it *still* cannot provide the capability, it should communicate that by returning `None` from its `provides_to` method.

Providers may keep state between calls to provide a capability but their correctness must not depend on any ordering of method calls except that, of course, their `__init__` is called first. For instance, a provider can retain an index that it downloads to answer `provides_to`, but if that index can expire, the provider should check for that and retrieve an updated index if necessary.

exception owmeta_core.capability.**CannotProvideCapability**(*cap*, *provider*)

Bases: `Exception`

Thrown by a *provider* when it cannot provide the capability during the object's execution

Parameters

cap
[*Capability*] the capabiilty

provider
[*Provider*] the provider which failed to provide `cap`

exception owmeta_core.capability.**NoProviderAvailable**(*cap*, *receiver*, *providers*)

Bases: `Exception`

Thrown when there is no provider available for a capabiilty

Attributes

cap
[*Capability*] The capability that was sought

receiver
[*Capable*] The object for which the capability was sought

Parameters

cap
[*Capability*] The capability that was sought

receiver
[*Capable*] The object for which the capability was sought

providers
[`list of` *Provider*] Providers that were tried for the capability

exception owmeta_core.capability.**NoProviderGiven**(*cap*, *receiver=None*)

Bases: `Exception`

Thrown by a *Capable* when a *Capability* is needed, but none has been provided by a call to *accept_capability_provider*

Parameters

cap
[*Capability*] The capability that was sought

receiver
[*Capable*] The object for which a capability was needed

exception owmeta_core.capability.**UnwantedCapability**

Bases: Exception

Thrown by a *Capable* when *accept_capability_provider* is offered a provider for a capability that it does not "want", meaning it doesn't have the code to use it. This can happen when a sub-class of a Capable declares a needed capability without overriding *accept_capability_provider* to accept that capability.

class owmeta_core.capability.**Capability**(**args*, ***kwargs*)

Bases: object

A capability.

class owmeta_core.capability.**Capable**

Bases: object

An object which can have capabilities

accept_capability_provider(*cap*, *provider*)

The *Capable* should replace any previously accepted provider with the one given.

The capability *should* be checked to determine which capability is being provided, even if only one is declared on the class, so that if a sub-class defines a capability without defining how to accept it, then the wrong actions won't be taken. In case the capability isn't recognized, it is generally better to pass it to the super() implementation rather than failing to allow for cooperative multiple inheritance.

Parameters

cap
[*Capability*] the capabiilty

provider
[*Provider*] the provider which provides `cap`

property needed_capabilities

The list of needed capabilities. These should be treated as though they are required for any of the object's methods.

property wanted_capabilities

The list of wanted capabilities. These should be treated as though they are optional. The *Capable* subclass must determine how to deal with the provider not being available.

class owmeta_core.capability.**Provider**

Bases: object

A capability provider.

In general, providers should do any general setup in their initializer, and setup for any source passed into *provides_to* method if, in fact, the provider does provide the needed capabilities

provides(*cap*, *obj*)

Returns a provider of the given capability if it's one this provider provides; otherwise, returns None.

Parameters

cap
[*Capability*] The capability to provide

obj
[*Capable*] The object to provide the capability to

Returns

Provider **or** None

provides_to(*obj*, *cap*)

Returns a *Provider* if the provider provides a capability to the given object; otherwise, returns None.

The default implementation always returns None. Implementers of *Provider* should check they can actually provide the capability for the given object rather than just that they *might* be able to.

It's best to do setup for providing the capability before exiting this method rather than, for instance, in the methods of the returned provider when the *Capable* is trying to use it.

Parameters

obj
[*Capable*] The object needing/wanting the capability

cap
[*Capability*] The capability needed/wanted

Returns

Provider **or** None

owmeta_core.capability.**get_provider**(*ob*, *cap*, *provs*)

Get provider for a capabilty that can provide to the given object

Parameters

ob
[*Capable*] Object needing the capability

cap
[*Capability*] Capability needed

provs
[list of *Provider*] All providers available

Returns

Provider
A provider of the given capability or None

owmeta_core.capability.**get_providers**(*cap*, *provs*, *ob*)

Get providers for a capabilty

Parameters

cap
[*Capability*] Capability needed

provs
[list of *Provider*] All providers available

Yields

Provider
A Provider that provides the given capability

owmeta_core.capability.**is_capable**(*ob*)

Returns true if the given object can accept capability providers

Parameters

ob
[object] An object which may be a *Capable*

Returns

bool
True if the given object accepts capability providers of some kind. Otherwise, false.

owmeta_core.capability.**provide**(*ob*, *provs*)

Provide capabilities to ob out of provs

Parameters

ob
[object] An object which may need capabilities

provs
[list of *Provider*] The providers available

Raises

NoProviderAvailable
when there is no provider available

owmeta_core.capability_providers module

Classes for managing things in the owmeta-core project directory, typically named .owm

class owmeta_core.capability_providers.**SimpleCacheDirectoryProvider**(*cache_directory*, ***kwargs*)

Bases: *CacheDirectoryProvider*

Provides a directory for caching remote resources as local files

class owmeta_core.capability_providers.**SimpleDataSourceDirProvider**(*basedir*)

Bases: *OutputFilePathProvider*

Provides a directory under the provided base directory

class owmeta_core.capability_providers.**SimpleTemporaryDirectoryProvider**(*base_directory*, *suffix=None*, *prefix=None*, ***kwargs*)

Bases: *TemporaryDirectoryProvider*

Provides temporary directories under a given base directory

class owmeta_core.capability_providers.**TDSDPHelper**(*basedir*, *key*, *transaction_manager*)

Bases: *FilePathProvider*, *OutputFilePathProvider*

This provider relies on the transaction library's machinery to manage the transaction.

Consistency is NOT guaranteed in all cases: in particular, this provider uses a file-based locking mechanism with a "lock file" in the given base directory which, if it's deleted during the two-phase commit process, removes the isolation of the changes made in the directory.

sortKey()

See also:

transaction.interfaces.IDataManager

class owmeta_core.capability_providers.**TransactionalDataSourceDirProvider**(*basedir*, *transaction_manager*)

Bases: *OutputFilePathProvider*, *FilePathProvider*

Provides a DataSourceDirectoryProvider with transactional semantics.

Provides a *TDSDPHelper* for *DataSource* objects, indexed by the `DataSource` identifier. If asked to provide a *FilePathCapability* (i.e, a directory for input), and the `DataSource` is a *LocalFileDataSource*, then we'll check that a file named with the value of *file_name* is in the provided directory.

class owmeta_core.capability_providers.**WorkingDirectoryProvider**(*cwd=None*, ***kwargs*)

Bases: *FilePathProvider*

Provides file paths from the current working directory for *data_trans.local_file_ds.LocalFileDataSource* instances.

Parameters

cwd
[`str` or `pathlib.Path`, `optional`] The working directory to use. The default is what `os.getcwd` returns

owmeta_core.capability_providers.**getrandbits**(*k*) → x. Generates an int with k random bits.

owmeta_core.capable_configurable module

class owmeta_core.capable_configurable.**CapableConfigurable**(**args*, ***kwargs*)

Bases: *Capable*, *Configurable*

Helper class for *Capable* objects that are also *Configurable*

Takes the providers from the *capability.providers* configuration value and calls `provide` with the resulting providers. If the value is unset or empty, then `provide` will not be called.

capability.providers

a list of *"provider path"* strings or *Providers*

Raises

NoProviderAvailable
if any of the needed capabilities cannot be provided

owmeta_core.cli module

owmeta_core.cli.**additional_args**(*parser*)

Add some additional options specific to CLI

owmeta_core.cli.**main**(**args*)

Entry point for the command line interface.

Additional sub-commands can be added by specifying them in an entry point in your package's setup.py like this:

```
'owmeta_core.commands': [
    'subcommand_name = module.path.for:TheSubCommand',
    'sub.sub.subcommand_name = module.path.for:TheSubSubSubCommand',
],
```

Where, `subcommand_name` will be the name of the sub-command under the top-level `owm` command and `module.path.for.TheSubCommand` will be the class implementing the command. To add to existing sub-commands one indicate the place in the command hierarchy as in `sub.sub.subcommand_name`: `TheSubSubSubCommand` would be available under the (hypothetical) existing `owm sub sub` command as `owm sub sub subcommand_name`

So-called "hints" can affect the way command implementations are interepreted such as indicating whether a method argument should be read in as a positional argument or an option and what a command-line option should be named (as opposed to deriving it from a parameter name or member variable). There is a set of hints which are a part of owmeta-core (see *CLI_HINTS*), but these can be augmented by specifying entry points like this:

```
'owmeta_core.cli_hints': 'hints = module.path.for:CLI_HINTS',
```

If `module.path.for.CLI_HINTS` is a dictionary, it will get added to the hints, potentially affecting any sub-commands without hints already available. The entry point name (`hints` in the example) is only used for error-reporting by this module. Although this is not strictly enforced, adding hints for sub-commands published by other modules, including owmeta-core, should be avoided to ensure consistent behavior across installations. See *owmeta_core.cli_hints* source for the format of hints.

See `CLICommandWrapper` for more details on how the command line options are constructed.

Parameters

***args**
Arguments to the command. Used instead of `sys.argv`

owmeta_core.cli_command_wrapper module

exception owmeta_core.cli_command_wrapper.**CLIUserError**

Bases: `Exception`

An error which the user would have to correct.

Typically caused by invalid user input

class owmeta_core.cli_command_wrapper.**CLIAppendAction**(*mapper*, *key*, *index=-1*, *mapped_name=None*, **args*, ***kwargs*)

Bases: *CLIStoreAction*

Extends CLIStoreAction to append to a set of accumulated values

Used for recording a `dict`

Parameters

mapper
[*CLIArgMapper*] CLI argument to Python mapper

key
[`str`] Indicates what kind of argument is being mapped. One of *INSTANCE_ATTRIBUTE*, *METHOD_NAMED_ARG*, *METHOD_KWARGS*, *METHOD_NARGS*

index
[`int`] Argument index. Used for maintaining the order of arguments when passed to the runner

mapped_name
[`str`] The name to map to. optional.

***args**
passed to `Action`

****kwargs**
passed to `Action`

class owmeta_core.cli_command_wrapper.**CLIArgMapper**

Bases: `object`

Stores mappings for arguments and maps them back to the part of the object they come from

apply(*runner*)

Applies the collected arguments to the runner by calling methods and traversing the object attributes as required

Parameters

runner
[`object`] Target of the command and source of argument and method names

See also:

CLICommandWrapper
accepts a runner argument in its `__init__` method

runners

Mapping from subcommand names to functions which run for them

class owmeta_core.cli_command_wrapper.**CLICommandWrapper**(*runner*, *mapper=None*, *hints=None*, *hints_map=None*, *program_name=None*)

Bases: `object`

Wraps an object such that it can be used in a command line interface

Parameters

runner
[`object`] An object that provides the methods to be invoked

mapper
[*CLIArgMapper*] Stores the arguments and associated runners for the command. A mapper is created if none is provided. optional

hints
[`dict`] A multi-level dict describing how certain command line arguments get turned into attributes and method arguments. If `hints` is not provided, the hints are looked up by the runner's fully-qualified class name in `hints_map`. optional

hints_map
[`dict`] A multi-level dict describing how certain command line arguments get turned into attributes and method arguments. Defaults to *CLI_HINTS*. optional

program_name
[`str`] The name of the top-level program. Uses `sys.argv[0]` if not provided. optional

extract_args(*val*)

Extract arguments from the method or class docstring

In the return value (see below), the `summary` is a `str` used in listing out sub-commands. The `detail` is for the sub-command usage information and should, generally, include the `summary`. The `params` are a list *ParamInfo* objects describing the parameters.

Parameters

val
[object] The object with the documentation

Returns

tuple
a triple, (summary, detail, params)

main(*args=None*, *argument_callback=None*, *argument_namespace_callback=None*)

Runs in a manner suitable for being the 'main' method for a command line interface: parses arguments (as would be done with the result of *parser*) from sys.argv or the provided args list and executes the commands specified therein

Parameters

args
[list] the argument list to parse. optional

argument_callback
[callable()] a callback to add additional arguments to the command line. optional

argument_namespace_callback
[callable()] a callback to handle the parsed arguments to the command line. optional

parser(*parser=None*)

Generates the argument parser's arguments

Parameters

parser
[argparse.ArgumentParser] The parser to add the arguments to. optional: will create a parser if none is given

class owmeta_core.cli_command_wrapper.**CLIStoreAction**(*mapper*, *key*, *index=-1*, *mapped_name=None*, **args*, ***kwargs*)

Bases: Action

Interacts with the CLIArgMapper

Parameters

mapper
[*CLIArgMapper*] CLI argument to Python mapper

key
[str] Indicates what kind of argument is being mapped. One of *INSTANCE_ATTRIBUTE*, *METHOD_NAMED_ARG*, *METHOD_KWARGS*, *METHOD_NARGS*

index
[int] Argument index. Used for maintaining the order of arguments when passed to the runner

mapped_name
[str] The name to map to. optional.

***args**
passed to Action

****kwargs**
passed to Action

class owmeta_core.cli_command_wrapper.**CLIStoreTrueAction**(**args*, ***kwargs*)

Bases: *CLIStoreAction*

Action for storing `True` when a given option is provided

Parameters

***args**
passed to *CLIStoreAction*

****kwargs**
passed to *CLIStoreAction*

class owmeta_core.cli_command_wrapper.**CLISubCommandAction**(*mapper*, **args*, ***kwargs*)

Bases: `_SubParsersAction`

Action for sub-commands

Extends the normal action for sub-parsers to record the subparser name in a mapper

Parameters

mapper
[*CLIArgMapper*] CLI argument to Python mapper

***args**
Passed on to `argparse._SubParsersAction`

****kwargs**
Passed on to `argparse._SubParsersAction`

owmeta_core.cli_command_wrapper.**ARGUMENT_TYPES = {'int': <class 'int'>}**

Map from parameter types to type constructors for parsing arguments

owmeta_core.cli_common module

owmeta_core.cli_common.**INSTANCE_ATTRIBUTE = 'INSTANCE_ATTRIBUTE'**

Indicates an option that corresponds to a command object's instance attribute

owmeta_core.cli_common.**METHOD_KWARGS = 'METHOD_KWARGS'**

Indicates an option that corresponds to the keyword argument consumer of a method (e.g. `**kwargs`)

owmeta_core.cli_common.**METHOD_NAMED_ARG = 'METHOD_NAMED_ARG'**

Indicates an option that corresponds to a method's named parameter

owmeta_core.cli_common.**METHOD_NARGS = 'METHOD_NARGS'**

Indicates an option that corresponds to the variadic argument consumer of a method (e.g. `*args`)

owmeta_core.cli_hints module

Hints for the CLI wrapper that help mapping from the Python methods to command line arguments.

CLI_HINTS
hints accepted by *CLICommandWrapper*

owmeta_core.collections module

class owmeta_core.collections.**Bag**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *Container*

A convenience class for working with a rdf:Bag

class owmeta_core.collections.**Container**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataObject*

Base class for rdfs:Containers

Example (*Bag*, `Alt`, and `Seq` have the same operations):

```
>>> nums = Bag(ident="http://example.org/fav-numbers")
>>> nums[1] = 42
>>> nums.set_member(2, 415)
owmeta_core.statement.Statement(...)
>>> nums._3(15)
owmeta_core.statement.Statement(...)
>>> nums._2.index
2
>>> nums._1()
42
>>> nums[2]
415
>>> nums._2(6)
owmeta_core.statement.Statement(...)
>>> nums[2]
6
```

Note that because the set of entries in `rdfs:Container` is not bounded, iteration over *Containers* is not bounded. To iterate over a *Container*, it is recommended to add some external bound with `itertools.islice` or something like `zip(range(bound), container)`. Where values have not been set, `None` will be returned.

set_member(*index*, *item*)

Set a member at the given index.

If an existing value is set at the given index, then it will be replaced. Note that, as described in the RDF Primer, there is no well-formedness guarantee: in particular, some other instance of a container may declare a different value at the same index.

class owmeta_core.collections.**ContainerMembershipProperty**(**args*, ***kwargs*)

Bases: `UnionProperty`

Base class for container membership properties like `rdf:_1`, `rdf:_2`, ...

owner_type

alias of *BaseDataObject*

owmeta_core.command module

This module defines the root of a high-level interface for owmeta_core, refered to as "OWM" (for the *main class* in the interface), "owm" (for the command line that wraps the interface), or "the command interface" in the documentation. Additional "sub-commands" may be defined which provide additional functionality.

If there is a suitable method in the high-level interface, it should generally be preferred to the lower-level interfaces for stability.

exception owmeta_core.command.**AlreadyDisconnected**(*owm*)

Bases: Exception

Thrown when OWM is already disconnected but a request is made to disconnect again

exception owmeta_core.command.**ConfigMissingException**(*key*)

Bases: *GenericUserError*

Thrown when a configuration key is missing

exception owmeta_core.command.**DirtyProjectRepository**

Bases: Exception

Thrown when we're about to commit, but the project repository has changes to the graphs such that it's not safe to just re-serialize the indexed database over the graphs.

exception owmeta_core.command.**InvalidGraphException**

Bases: *GenericUserError*

Thrown when a graph cannot be translated due to formatting errors

exception owmeta_core.command.**NoConfigFileError**(*config_file_path*)

Bases: *GenericUserError*

Thrown when a project config file (e.g., '.owm/owm.conf') cannot be found

exception owmeta_core.command.**OWMDirMissingException**

Bases: *GenericUserError*

Thrown when the .owm directory is needed, but cannot be found

exception owmeta_core.command.**StatementValidationError**(*statements*)

Bases: *GenericUserError*

Thrown in the case that a set of statements fails to validate

exception owmeta_core.command.**UnreadableGraphException**

Bases: *GenericUserError*

Thrown when a graph cannot be read due to it being missing, the active user lacking permissions, etc.

class owmeta_core.command.**NullContextRecord**(*node_index*, *statement*)

Bases: _NullContextRecord

Stored when the identifier for the context of an object we're saving is None

Create new instance of _NullContextRecord(node_index, statement)

class owmeta_core.command.**OWM**(*owmdir=None*, *non_interactive=False*)

Bases: object

High-level commands for working with owmeta data

Attributes

cleanup_manager
[atexit-like] An object to which functions can be registered and unregistered. To handle cleaning up connections that were not closed more directly (e.g., by calling *disconnect*)

progress_reporter
[tqdm-like] A callable that presents some kind of progress to a user. Interface is a subset of the tqdm.tqdm object: the reporter must accept unit, miniters, file, and leave options, although what it does with those is unspecified. Additionally, for reporting progress on cloning a project, an *optional interface* is required.

add_graph(*url=None*, *context=None*, *include_imports=True*)

Fetch a graph and add it to the local store.

Parameters

url
[str] The URL of the graph to fetch

context
[rdflib.term.URIRef] If provided, only this context and, optionally, its imported graphs will be added.

include_imports
[bool] If True, imports of the named context will be included. Has no effect if context is None.

clone(*url=None*, *update_existing_config=False*, *branch=None*)

Clone a data store

Parameters

url
[str] URL of the data store to clone

update_existing_config
[bool] If True, updates the existing config file to point to the given file for the store configuration

branch
[str] Branch to checkout after cloning

commit(*message*, *skip_serialization=False*)

Write the graph and configuration changes to the local repository

Parameters

message
[str] commit message

skip_serialization
[bool] If set, then skip graph serialization. Useful if you have manually changed the graph serialization or just want to commit changes to project configuration

connect(*read_only=False*, *expect_cleanup=False*)

Create a connection to the project database.

Most commands will create their own connections where needed, but for multiple commands you'll want to create one connection at the start. Multiple calls to this method can be made without calling *disconnect* on the resulting connection object, but only if read_only has the same value for all calls.

Read-only connections can only be made with the default stores: if you have configured your own store and you want the connection to be read-only, you must change the configuration to make it read-only before calling *connect*.

Parameters

read_only
[bool] if True, the resulting connection will be read-only

expect_cleanup
[bool] if False, a warning will be issued if the `cleanup_manager` has to disconnect the connection

Returns

ProjectConnection
Usable as a `context manager`

`declare`(*python_type*, *attributes=()*, *id=None*)

Create a new data object or update an existing one

Parameters

python_type
[`str`] The path to the Python type for the object. Formatted like "full.module.path:ClassName"

attributes
[`str`] Attributes to set on the object before saving

id
[`str`] The identifier for the object

`diff`(*color=False*)

Show differences between what's in the working context set and what's in the serializations

Parameters

color
[bool] If set, then ANSI color escape codes will be incorporated into diff output. Default is to output without color.

`disconnect`()

Destroy a connection to the project database

Should not be called if there is no active connection

`fetch_graph`(*url*)

Fetch a graph

Parameters

url
[`str`] URL for the graph

`get_default_context`()

Read the current target context for the repository

`git`(**args*)

Runs git commmands in the ".owm" directory

Parameters

***args**
arguments to git

`imports_context`(*context=None*, *user=False*)

Read or set current target imports context for the repository

Parameters

context
[str] The context to set

user
[bool] If set, set the context only for the current user. Has no effect for retrieving the context

`init`(*update_existing_config=False*, *default_context_id=None*)

Makes a new graph store.

The configuration file will be created if it does not exist. If it *does* exist, the location of the database store will, by default, not be changed in that file

If not provided, some values will be prompted for, unless batch (non-interactive) mode is enabled. If batch mode is enabled, either an error will be returned or a default value will be used for missing options. Values which are required either in a prompt or as options are indicated as "Required" below.

Parameters

update_existing_config
[bool] If True, updates the existing config file to point to the given file for the store configuration

default_context_id
[str] URI for the default context. Required

`list_contexts`()

List contexts

`regendb`()

Regenerates the indexed database from graph serializations.

Note that any uncommitted contents in the indexed database will be deleted.

`retract`(*subject*, *property*, *object*)

Remove one or more statements

Parameters

subject
[str] The object which you want to say something about. optional

property
[str] The type of statement to make. optional

object
[str] The other object you want to say something about. optional

`save`(*module*, *provider=None*, *context=None*)

Save the data in the given context

Saves the "mapped" classes declared in a module and saves the objects declared by the "provider" (see the argument's description)

Parameters

module
[str] Name of the module housing the provider

provider
[str] Name of the provider, a callble that accepts a context object and adds statements to it. Can be a "dotted" name indicating attribute accesses. Default is *DEFAULT_SAVE_CALLABLE_NAME*

context
[str] The target context. The default context is used

say(*subject*, *property*, *object*)

Make a statement

Parameters

subject
[str] The object which you want to say something about

property
[str] The type of statement to make

object
[str] The other object you want to say something about

set_default_context(*context*, *user=False*)

Set current default context for the repository

Parameters

context
[str] The context to set

user
[bool] If set, set the context only for the current user. Has no effect for retrieving the context

translate(*translator*, *output_key=None*, *output_identifier=None*, *data_sources=()*, *named_data_sources=None*)

Do a translation with the named translator and inputs

Parameters

translator
[str] Translator identifier

output_key
[str] Output key. Used for generating the output's identifier. Exclusive with output_identifier

output_identifier
[str] Output identifier. Exclusive with output_key

data_sources
[list of str] Input data sources

named_data_sources
[dict] Named input data sources

basedir

The base directory. owmdir is resolved against this base

bundle

OWMBundle: Bundle commands

config

OWMConfig: Config file commands.

Without any sub-command, prints the configuration parameters

config_file

The config file name

context

Context to use instead of the default context. Commands that work with other contexts (e.g., `owm contexts rm-import`) will continue to use those other contexts unless otherwise indicated

contexts

OWMContexts: Commands for working with contexts

graph_accessor_finder

Finds an RDFLib graph from the given URL

namespace

OWMNamespace: RDF namespace commands

namespace_manager_store_name

The file name of the namespace database store

non_interactive

If this option is provided, then interactive prompts are not allowed

owmdir

The base directory for owmeta files. The repository provider's files also go under here

registry

OWMRegistry: Commands for dealing with the class registry, a mapping of RDF types to constructs in programming languages

Although it is called the "*class* registry", the registry can map RDF types to constructs other than classes in the target programming language, particularly in languages that don't have classes (e.g., C) or where the use of classes is not preferred in that language.

repository_provider

The provider of the repository logic (cloning, initializing, committing, checkouts)

source

OWMSource: Commands for working with DataSource objects

store_name

The file name of the database store

temporary_directory

The base temporary directory for any operations that need one

property transaction_manager

The `transaction.TransactionManager` for the current connection

translator

OWMTranslator: Data source translator commands

type

OWMTypes: Commands for dealing with Python classes and RDF types

userdir

Root directory for user-specific configuration

class owmeta_core.command.**OWMConfig**(*parent*)

Bases: object

Config file commands.

Without any sub-command, prints the configuration parameters

delete(*key*)

Deletes a config value

Parameters

key
[str] The configuration key

get(*key*)

Read a config value

Parameters

key
[str] The configuration key

set(*key*, *value*)

Set a config value

Parameters

key
[str] The configuration key

value
[str] The value to set

user

If set, configs are only for the user; otherwise, they would be committed to the repository

user_config_file

The user config file name

class owmeta_core.command.**OWMContexts**(*parent*)

Bases: object

Commands for working with contexts

add_import(*importer*, *imported*)

Add an import to the imports graph

Parameters

importer
[str] The importing context

imported
[list str] The imported context

bundle(*context*)

Show the closest bundle that defines this context

Parameters

context
[str] The context to lookup

edit(*context=None*, *format=None*, *editor=None*, *list_formats=False*)

Edit a provided context or the current default context.

The file name of the serialization will be passed as the sole argument to the editor. If the editor argument is not provided, will use the EDITOR environment variable. If EDITOR is also not defined, will try a few known editors until one is found. The editor must write back to the file.

Parameters

context
[str] The context to edit

format
[str] Serialization format (ex, 'n3', 'nquads'). Default 'n3'

editor
[str] The program which will be used to edit the context serialization.

list_formats
[bool] List the formats available for editing (I.O.W., formats that we can both read and write)

list(*include_dependencies=False*, *include_default=False*)

List the set of contexts in the graph

Parameters

include_dependencies
[bool] If set, then contexts from dependencies will be included

include_default
[bool] If set, then include the default graph in the results as well

list_changed()

Return the set of contexts which differ from the serialization on disk

list_importers(*context*)

List the contexts that import the given context

Parameters

context
[str] The context to list importers for

list_imports(*context*)

List the contexts that the given context imports

Parameters

context
[str] The context to list imports for

rm(**context*)

Remove a context

Parameters

***context**
[str] Context to remove

rm_import(*importer*, *imported*)

Remove an import statement

Parameters

importer
[str] The importing context

imported
[list of str] An imported context

serialize(*context=None*, *destination=None*, *format='nquads'*, *include_imports=False*, *whole_graph=False*)

Serialize the current default context or the one provided

Parameters

context
[str] The context to save

destination
[file or str] A file-like object to write the file to or a file name. If not provided, messages the result.

format
[str] Serialization format (ex, 'n3', 'nquads')

include_imports
[bool] If true, then include contexts imported by the provided context in the result. The default is not to include imported contexts.

whole_graph
[bool] Serialize all contexts from all graphs (this probably isn't what you want)

class owmeta_core.command.**OWMNamespace**(*parent*)

Bases: object

RDF namespace commands

bind(*prefix*, *uri*)

Bind a prefix to a namespace URI

Parameters

prefix
[str] Prefix to bind to a namespace URI

uri
[str] Namespace URI to bind to a prefix

list()

List namespace prefixes and URIs in the project

class owmeta_core.command.**OWMRegistry**(*parent*)

Bases: object

Commands for dealing with the class registry, a mapping of RDF types to constructs in programming languages

Although it is called the "*class* registry", the registry can map RDF types to constructs other than classes in the target programming language, particularly in languages that don't have classes (e.g., C) or where the use of classes is not preferred in that language.

list(*module=None*, *rdf_type=None*, *class_name=None*)

List registered classes

Parameters

module
[str] If provided, limits the registry entries returned to those that have the given module name. Optional.

rdf_type
[str] If provided, limits the registry entries returned to those that have the given RDF type. Optional.

class_name
[str] If provided, limits the registry entries returned to those that have the given class name. Optional.

rm(**registry_entry*)

Remove a registry entry

Parameters

***registry_entry**
[str] Registry entry to remove

show(**registry_entry*)

Show registry entries

Parameters

***registry_entry**
[str] Registry entry to show

module_access

OWMRegistryModuleAccess: Commands for manipulating software module access in the class registry

class owmeta_core.command.**OWMRegistryModuleAccess**(*parent*)

Bases: object

Commands for manipulating software module access in the class registry

list(*registry_entry=None*)

List module accessors

Parameters

registry_entry
[str] Registry entry ID. Optional

Returns

sequence of ModuleAccessor

declare

OWMRegistryModuleAccessDeclare: Commands for module access declarations

show

OWMRegistryModuleAccessShow: Show module accessor description

class owmeta_core.command.**OWMRegistryModuleAccessDeclare**(*parent*)

Bases: object

Commands for module access declarations

python_pip(*package_name*, *package_version=None*, *index=None*, *module_names=None*, *module_id=None*)

Declare access with a Python pip package

The given module should already have been defined in the class registry. This may be achieved by the "owm save" command.

Parameters

package_name
[str] Name of the package

package_version
[str] Version of the package. If not provided, will attempt to find the active version in package metadata

index
[str] The index to get the package from. Optional

module_names
[list of str] Name of the module. If not provided, will attempt to find the modules from package metadata. Multiple module names can be provided

module_id
[str] URI identifier of the module. Cannot be specified along with module_name

class owmeta_core.command.**OWMRegistryModuleAccessShow**(*parent*)

Bases: object

Show module accessor description

class owmeta_core.command.**OWMSource**(*parent*)

Bases: object

Commands for working with DataSource objects

derivs(*data_source*)

List data sources derived from the one given

Parameters

data_source
[str] The ID of the data source to find derivatives of

list(*context=None*, *kind=None*, *full=False*)

List known sources

Parameters

kind
[str] Only list sources of this kind

context
[str] The context to query for sources

full
[bool] Whether to (attempt to) shorten the source URIs by using the namespace manager

`list_kinds`(*full=False*)

List kinds of DataSources available in the current context.

Note that *only* DataSource types which are reachable from the current context will be listed. So if, for instance, you have just saved some types (e.g., with `owm save`) but have not added an import of the contexts for those types, you may not see any results from this command.

Parameters

full
[bool] Whether to (attempt to) shorten the source URIs by using the namespace manager

`rm`(**data_source*)

Remove a `DataSource`

Parameters

***data_source**
[str] ID of the source to remove

`show`(**data_source*)

Parameters

***data_source**
[str] The ID of the data source to show

`class owmeta_core.command.OWMTranslator`(*parent*)

Bases: `object`

Data source translator commands

`create`(*translator_type*)

Creates an instance of the given translator class and adds it to the graph

Parameters

translator_type
[str] RDF type for the translator class

`list`(*context=None*, *full=False*)

List translators

Parameters

context
[str] The root context to search

full
[bool] Whether to (attempt to) shorten the source URIs by using the namespace manager

`list_kinds`(*full=False*)

List kinds of DataTranslators

Note that *only* DataTranslator types which are reachable from the current context will be listed. So if, for instance, you have just saved some types (e.g., with `owm save`) but have not added an import of the contexts for those types, you may not see any results from this command.

Parameters

full
[bool] Whether to (attempt to) shorten the translator URIs by using the namespace manager

rm(**translator*)

Remove a DataTranslator

Parameters

***translator**
[str] ID of the source to remove

show(*translator*)

Show a translator

Parameters

translator
[str] The translator to show

class owmeta_core.command.**OWMTypes**(*parent*)

Bases: object

Commands for dealing with Python classes and RDF types

rm(**type*)

Removes info about the given types, like rdfs:subClassOf statements, and removes the corresponding registry entries as well

Parameters

***type**
[str] Types to remove

class owmeta_core.command.**ProjectConnection**(*owm*, *connection*, *connections*, ***, *expect_cleanup=True*)

Bases: object

Connection to the project database

transaction()

Context manager that executes the enclosed code in a transaction and then closes the connection. Provides the connection for binding with as.

class owmeta_core.command.**SaveValidationFailureRecord**(*user_module*, *stack*, *validation_record*)

Bases: _SaveValidationFailureRecord

Record of a validation failure in *OWM.save*

Create new instance of _SaveValidationFailureRecord(user_module, stack, validation_record)

class owmeta_core.command.**UnimportedContextRecord**(*importer*, *context*, *node_index*, *statement*)

Bases: _UnimportedContextRecord

Stored when statements include a reference to an object but do not include the context of that object in the callback passed to *OWM.save*. For example, if we had a callback like this:

```
def owm_data(ns):
    ctxA = ns.new_context(ident='http://example.org/just-pizza-stuff')
    ctxB = ns.new_context(ident='http://example.org/stuff-sam-likes')
    sam = ctxB(Person)('sam')
```

(continues on next page)

(continued from previous page)

```
    pizza = ctxA(Thing)('pizza')
    sam.likes(pizza)
```

it would generate this error because `ctxB` does not declare an import for `ctxA`

Create new instance of _UnimportedContextRecord(importer, context, node_index, statement)

owmeta_core.command.**DEFAULT_SAVE_CALLABLE_NAME = 'owm_data'**

Default name for the provider in the arguments to *OWM.save*

owmeta_core.command.**DSD_DIRKEY = 'owmeta_core.command.OWMDirDataSourceDirLoader'**

Key used for data source directory loader and file path provider

owmeta_core.command_util module

Utilities for making objects that work with the `CLICommandWrapper`

exception owmeta_core.command_util.**GenericUserError**

Bases: `Exception`

An error which should be reported to the user. Not necessarily an error that is the user's fault

class owmeta_core.command_util.**IVar**(*default_value=None*, *doc=None*, *value_type=<class 'str'>*, *name=None*)

Bases: `object`

A descriptor for instance variables amended to provide some attributes like default values, value types, etc.

classmethod property(*wrapped=None*, **args*, ***kwargs*)

Creates a *PropertyIVar* from a method

Typically, this will be used as a decorator for a method

Parameters

wrapped
[`types.FunctionType` or `types.MethodType`] The function to wrap. optional: if omitted, returns a function that can be invoked later to create the *PropertyIVar*

class owmeta_core.command_util.**PropertyIVar**(**args*, ***kwargs*)

Bases: *IVar*

An *IVar* that functions similarly to a `property`

Typically a PropertyIVar will be created by using *IVar.property* as a decorator for a method like:

```
class A(object):

    @IVar.property('default_value')
    def prop(self):
        return 'value'
```

__get__(*target*, *objecttype=None*)

Executes the provided getter

When the getter is first called, and when a setter is also defined, the setter will be called with the default value before the getter is called for the first time. _Even if the default_value is not set explicitly, the setter will still be called with 'None'.

setter(*fset*)

Decorator for the setter that goes along with this property.

See also:

property

class owmeta_core.command_util.**SubCommand**(*cmd*)

Bases: object

A descriptor that wraps objects which function as sub-commands to *OWM* or to other sub-commands

owmeta_core.configure module

This module defines a generic configuration dictionary with a few extra features.

A list of all documented configuration values can be found under "configuration values" in the index.

exception owmeta_core.configure.**BadConf**

Bases: Exception

Special exception subclass for alerting the user to a bad configuration

class owmeta_core.configure.**ConfigValue**

Bases: object

A value to be configured. Base class intended to be subclassed, as its only method is not implemented

class owmeta_core.configure.**Configurable**(*conf=None*, ***kwargs*)

Bases: object

An object which can accept configuration. A base class intended to be subclassed.

get(*pname*, *default=None*)

Gets a config value from this *Configurable*'s conf

See also:

Configuration.get

class owmeta_core.configure.**Configuration**(***initial_values*)

Bases: object

A simple configuration object. Enables setting and getting key-value pairs

Unlike a dict, Configuration objects will execute a function when retrieving values to enable deferred computation of seldom-used configuration values. In addition, entries in a *Configuration* can be aliased to one another.

copy(*other*)

Copy configuration from another object into this one

Parameters

other
[dict or *Configuration*] Configuration to copy from

Returns

Configuration
self

get(*pname*, *default=NO_DEFAULT*)

Get some parameter value out by asking for a key. Note that unlike `dict`, if you don't specify a default, then a `KeyError` is raised

Parameters

pname
[`str`] they key of the value you want to return.

default
[`object`] The default value to return if there's no entry for `pname`

Returns

`object`
The value corresponding to the key

link(**names*)

Call link() with the names of configuration values that should always be the same to link them together

classmethod open(*file_name*)

Open a configuration file and read it to build the internal state.

Sets *configure.file_location* to the given file_name

configure.file_location

The location where a *Configuration* was loaded from. This may be set by any function that loads the configuration – not just *Configuration.open*. Generally, this value is suitable for finding files in locations relative to the config file, but not for much else.

Parameters

file_name
[`str`] configuration file encoded as JSON

Returns

Configuration
returns an instance of this class with the configuration taken from the JSON file

See also:

process_config

classmethod process_config(*config_dict*, *variables=None*)

Resolves variables in config values and creates an instance of this class

Parameters

config_dict
[`dict`] The source for the resulting config

Returns

Configuration
config populated with variables

owmeta_core.context module

class owmeta_core.context.**Context**(**args*, ***kwargs*)

Bases: ContextualizableDataUserMixin

A context. Analogous to an RDF context, with some special sauce

__call__(*o=None*, **args*, ***kwargs*)

Contextualize an object

Parameters

o
[object] The object to contexualize

__bool__()

Always returns True. Prevents a context with zero statements from testing false since that's not typically a useful branching condition.

add_import(*context*)

Add an imported context

add_statement(*stmt*)

Add a statement to the context. Typically, statements will be added by contextualizing a *DataObject* and making a statement thereon. For instance, if a class A has a property p, then for the context ctx:

```
ctx(A)(ident='http://example.org').p('val')
```

would add a statement to ctx like:

```
(A(ident='http://example.org'), A.p.link, rdflib.term.Literal('val'))
```

Parameters

stmt
[owmeta_core.statement.Statement] Statement to add

clear()

Clear declared statements

contents()

Returns statements added to this context

Returns

generator

contents_triples()

Returns, as triples, the statements staged in this context

Yields

tuple
A triple of RDFLib Identifiers

declare_imports(*context=None*, *transitive=False*)

Declare imports statements in the given context

Parameters

context
[*Context*, optional] The context in which to declare statements. If not provided, one will be created with `self.conf[IMPORTS_CONTEXT_KEY]` as the identifier

Returns

Context
The context in which the statements were declared

`load_graph_from_configured_store()`

Create an RDFLib graph for accessing statements in this context, *including* imported contexts. The "configured" graph is the one at `self.rdf`.

Returns

rdflib.graph.ConjunctiveGraph

`load_mixed_graph()`

Create a graph for accessing statements both staged (see *load_staged_graph*) and stored (see *load_graph_from_configured_store*). No effort is made to either deduplicate, smush blank nodes, or logically reconcile statements between staged and stored graphs.

Returns

rdflib.graph.ConjunctiveGraph

`load_own_graph_from_configured_store()`

Create a RDFLib graph for accessing statements in this context, *excluding* imported contexts. The "configured" graph is the one at `self.conf['rdf.graph']`.

Returns

rdflib.graph.ConjunctiveGraph

`load_staged_graph()`

Create a graph for accessing statements declared in this specific instance of this context. This statements may not have been written to disk; therefore, they are "staged".

Returns

rdflib.graph.ConjunctiveGraph

`rdf_graph()`

Return the principal graph for this context. For a regular *Context* this will be the "staged" graph.

Returns

rdflib.graph.ConjunctiveGraph

See also:

staged
Has the "staged" principal graph.

mixed
Has the "mixed" principal graph.

stored
Has the "stored" graph, including imports.

own_stored
Has the "stored" graph, excluding imports.

`remove_statement`(*stmt*)

Remove a statement from the context

Parameters

stmt
[`tuple`] Statement to remove

`save`(*graph=None*, *inline_imports=False*, *autocommit=True*, *saved_contexts=None*)

Alias to save_context

`save_context`(*graph=None*, *inline_imports=False*, *autocommit=True*, *saved_contexts=None*)

Adds the staged statements in the context to a graph

Parameters

graph
[`rdflib.graph.Graph` or `set`, `optional`] the destination graph. Defaults to `self.rdf`

inline_imports
[bool, `optional`] if `True`, imported contexts will also be written added to the graph

autocommit
[bool, `optional`] if `True`, `graph.commit` is invoked after adding statements to the graph (including any imported contexts if `inline_imports` is `True`)

saved_contexts
[`set`, `optional`] a collection of identifiers for previously saved contexts. Note that `id` is used to get an identifier: the return value of `id` can be repeated after an object is deleted.

`save_imports`(*context=None*, **args*, *transitive=True*, ***kwargs*)

Add the *`imports`* on this context to a graph

Parameters

context
[*`Context`*, `optional`] The context to add statements to. This context's configured graph will ultimately receive the triples. By default, a context will be created with `self.conf[IMPORTS_CONTEXT_KEY]` as the identifier

transitive
[bool, `optional`] If `True`, call imported imported contexts to save their imports as well

`transitive_imports`()

Return imports on this context and on imported contexts

Yields

`Context`

`property imports`

Return imports on this context

Yields

`Context`

`property mixed`

A read-only context whose principal graph is the "mixed" graph.

Returns

`QueryContext`

See also:

rdf_graph
load_mixed_graph
Defines the principal graph for this context

property own_stored

A read-only context whose principal graph is the "stored" graph, excluding imported contexts.

Returns

QueryContext

See also:

rdf_graph
load_own_graph_from_configured_store
Defines the principal graph for this context

property rdf_object

Returns a dataobject for this context

Returns

owmeta_core.dataobject.DataObject

property staged

A read-only context whose principal graph is the "staged" graph.

Returns

QueryContext

See also:

rdf_graph
load_staged_graph
Defines the principal graph for this context

property stored

A read-only context whose principal graph is the "stored" graph, including imported contexts.

Returns

QueryContext

See also:

rdf_graph
load_graph_from_configured_store
Defines the principal graph for this context

property triples_saved

The number of triples saved in the most recent call to *save_context*

class owmeta_core.context.**ContextContextManager**(*ctx*, *to_import*)

Bases: object

The context manager created when Context::__call__ is passed a dict

class owmeta_core.context.**QueryContext**(**args*, ***kwargs*)

Bases: *Context*

A read-only context.

owmeta_core.context.**DEFAULT_CONTEXT_KEY = 'default_context_id'**

Configuration file key for the URI of a default RDF graph context.

This is the URI of the default graph in a project or bundle.

owmeta_core.context.**IMPORTS_CONTEXT_KEY = 'imports_context_id'**

Configuration file key for the URI of an imports RDF graph context.

The imports context holds the relationships between contexts, especially the imports relationship

owmeta_core.context_common module

owmeta_core.context_common.**CONTEXT_IMPORTS = rdflib.term.URIRef('http://schema.openworm.org/2020/07/Context/imports')**

URI for the `Context` imports predicate

owmeta_core.context_dataobject module

class owmeta_core.context_dataobject.**ContextDataObject**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Represents a context

owmeta_core.context_mapped_class_util module

owmeta_core.context_store module

class owmeta_core.context_store.**ContextStore**(*context=None*, *include_stored=False*, *imports_graph=None*, ***kwargs*)

Bases: `Store`

A store specific to a *Context*

A *ContextStore* may have triples

Parameters

context
[*Context*] The context to which this store belongs

include_stored
[bool] If `True`, the backing store will be queried as well as the staged triples in `context`

imports_graph
[`Store` or `Graph`] The graph to query for imports relationships between contexts

****kwargs**
Passed on to `Store`

contexts(*triple=None*)

Generator over all contexts in the graph. If triple is specified, a generator over all contexts the triple is in.

if store is graph_aware, may also return empty contexts

Returns
a generator over Nodes

owmeta_core.contextualize module

class owmeta_core.contextualize.**AbstractBaseContextualizable**

Bases: ABC

Abstract base class for contextualizables

Any class with an attribute `contextualize` with a Function value is recognized as a subclass

class owmeta_core.contextualize.**BaseContextualizable**(**args*, ***kwargs*)

Bases: object

Helper base-class for contextualizable objects. Caches contextualized objects returned from *contextualize_augment*

add_contextualization(*context*, *contextualization*)

Manually add a contextualized object to the cache

Parameters

context
[Context] The context of the object

contextualization
[object] The contextualized version of the object

contextualize(*context*)

Return an object with the given context. If the provided `context` is None, then `self` MUST be returned unmodified. Prefer to override `contextualize_agument` which will be called from this method.

It is generally not correct to set a field on the object and return the same object as this would change the context for other users of the object. Also, returning a copy of the object is usually inappropriate for mutable objects. Immutable objects may maintain a 'context' property and return a copy of themselves with that property set to the provided `context` argument.

contextualize_augment(*context*)

For sub-classes to override: Return an object with the given context. If the provided `context` is None, then `self` MUST be returned unmodified.

Returns

object
the contextualized object

decontextualize()

Return the object with all contexts removed. Sub-classes should override.

class owmeta_core.contextualize.**Contextualizable**(**args*, ***kwargs*)

Bases: *BaseContextualizable*

A *BaseContextualizable* with the addition of a default behavior of setting the context from the class's 'context' attribute. This generally requires that for the metaclass of the Contextualizable that a 'context' data property is defined. For example:

```
>>> class AMeta(ContextualizableClass):
...     @property
...     def context(self):
...         return self.__context
...
...     @context.setter
...     def context(self, ctx):
...         self.__context = ctx

>>> class A(six.with_metaclass(Contextualizable)):
...     pass
```

class owmeta_core.contextualize.**ContextualizableClass**(*name*, *typ*, *dct*)

Bases: type

A super-type for contextualizable classes

Attributes

context_carries
[tuple of str] When defining a specialized contextualizable class, you may want to define some attribute on the class that is only set if it's declared directly in the class body (e.g., by using property and name mangling). However, by default, contextualization creates a subclass and you may want your property to be "carried" into the new context. You can achieve this by declaring context_carries with the names of attributes that should be carried through a contextualization.

owmeta_core.contextualize.**contextualize_helper**(*context*, *obj*, *noneok=False*)

Does some extra stuff to make access to the type of a ContextualizingProxy work more-or-less like access to the the wrapped object

owmeta_core.contextualize.**decontextualize_helper**(*obj*)

Removes contexts from a ContextualizingProxy

owmeta_core.custom_dataobject_property module

class owmeta_core.custom_dataobject_property.**CustomProperty**(**args*, ***kwargs*)

Bases: *Contextualizable*, *DataUser*

Store a value associated with a DataObject

Properties can be be accessed like methods. A method call like:

```
a.P()
```

for a property P will return values appropriate to that property for a, the owner of the property.

Parameters

owner
[*owmeta_core.dataobject.DataObject*] The owner of this property

name
[str] The name of this property. Can be accessed as an attribute like:

```
owner.name
```

get(**args*)

Get the things which are on the other side of this property

The return value must be iterable. For a `get` that just returns a single value, an easy way to make an iterable is to wrap the value in a tuple like `(value,)`.

Derived classes must override.

get_terms(**args*)

Get the things which are on the other side of this property

The return value must be iterable. For a `get` that just returns a single value, an easy way to make an iterable is to wrap the value in a tuple like `(value,)`.

Derived classes must override.

has_value()

Returns true if the *CustomProperty* has any values set on it.

This may be defined differently for each property

one()

Returns a single value for the *CustomProperty* whether or not it is multivalued.

set(**args*, ***kwargs*)

Set the value of this property

Derived classes must override.

owmeta_core.data module

class owmeta_core.data.**Data**(*conf=None*, ***kwargs*)

Bases: *Configuration*

Provides configuration for access to the database.

Usually doesn't need to be accessed directly

rdf.graph

An RDFLib `ConjunctiveGraph`, possibly a `Dataset`. Configured according to *rdf.source* and any other variables used by the *RDFSource* corresponding

rdf.namespace

Default namespace bound to an empty string in the the namespace manager, *rdf.namespace_manager*

rdf.namespace_manager

RDFLib Namespace Manager. Typically, this is generated automatically during a call to *init*

rdf.namespace_manager.store

RDFLib store name specific to namespaces

rdf.namespace_manager.store_conf

Configuration for RDFLib store specified with *rdf.namespace_manager.store*

transaction_manager.provider

A *provider* for a transaction manager. Provider must resolve to a `callable` that accepts a *Data* instance.

transaction_manager

Transaction manager for RDFLib stores. Provided by *transaction_manager.provider* if that's defined. Should be passed to `IDataManager` instances within the scope of a given *Data* instance.

rdf.source

A string corresponding to a key in *SOURCES*

Parameters

conf
[*Configuration*] The base configuration from which this configuration will be built. This configuration will be copied into this one, but no direct reference will be retained

close()

Close a the configured database

closeDatabase()

Close a the configured database

destroy()

Close a the configured database

init()

Open the configured database

init_database()

Open the configured database

classmethod load(*file_name*)

Load a file into a new Data instance storing configuration in a JSON format

classmethod open(*file_name*)

Load a file into a new Data instance storing configuration in a JSON format

classmethod process_config(*config_dict*, ***kwargs*)

Load a file into a new Data instance storing configuration in a JSON format

class owmeta_core.data.**DataUser**(**args*, ***kwargs*)

Bases: *Configurable*

A convenience wrapper for users of the database

Classes which use the database should inherit from DataUser.

add_reference(*g*, *reference_iri*)

Add a citation to a set of statements in the database

Parameters
triples – A set of triples to annotate

add_statements(*graph*)

Add a set of statements to the database. Annotates the addition with uploader name, etc

Parameters
graph – An iterable of triples

infer()

Fire FuXi rule engine to infer triples

retract_statements(*graph*)

Remove a set of statements from the database.

Parameters
graph – An iterable of triples

class owmeta_core.data.**DefaultSource**(***kwargs*)

Bases: *RDFSource*

Reads from and queries against a configured database.

The default configuration.

The database store is configured with:

```
"rdf.source" = "default"
"rdf.store" = <your rdflib store name here>
"rdf.store_conf" = <your rdflib store configuration here>
```

Leaving unconfigured simply gives an in-memory data store.

class owmeta_core.data.**RDFSource**(***kwargs*)

Bases: *Configurable*, *ConfigValue*

Base class for data sources.

Alternative sources should derive from this class

open()

Called on `owmeta_core.connect()` to set up and return the rdflib graph. Must be overridden by subclasses.

class owmeta_core.data.**SPARQLSource**(***kwargs*)

Bases: *RDFSource*

Reads from and queries against a remote data store

```
"rdf.source" = "sparql_endpoint"
```

class owmeta_core.data.**SleepyCatSource**(***kwargs*)

Bases: *RDFSource*

Reads from and queries against a local Sleepycat database

The database can be configured like:

```
"rdf.source" = "Sleepycat"
"rdf.store_conf" = <your database location here>
```

class owmeta_core.data.**ZODBSource**(**args*, ***kwargs*)

Bases: *RDFSource*

Reads from and queries against a configured Zope Object Database.

If the configured database does not exist, it is created.

The database store is configured with:

```
"rdf.source" = "ZODB"
"rdf.store_conf" = <location of your ZODB database>
```

Leaving unconfigured simply gives an in-memory data store.

owmeta_core.data.**NAMESPACE_MANAGER_KEY = 'rdf.namespace_manager'**

Constant for *rdf.namespace_manager*

owmeta_core.data.**NAMESPACE_MANAGER_STORE_CONF_KEY = 'rdf.namespace_manager.store_conf'**

Constant for *rdf.namespace_manager.store_conf*

owmeta_core.data.**NAMESPACE_MANAGER_STORE_KEY = 'rdf.namespace_manager.store'**

Constant for *rdf.namespace_manager.store*

owmeta_core.data.**SOURCES = {'default': <class 'owmeta_core.data.DefaultSource'>, 'mysql': <class 'owmeta_core.data.MySQLSource'>, 'postgresql': <class 'owmeta_core.data.PostgreSQLSource'>, 'sleepycat': <class 'owmeta_core.data.SleepyCatSource'>, 'sparql_endpoint': <class 'owmeta_core.data.SPARQLSource'>, 'sqlite': <class 'owmeta_core.data.SQLiteSource'>, 'zodb': <class 'owmeta_core.data.ZODBSource'>}**

Table of possible sources for *rdf.source*

owmeta_core.data.**TRANSACTION_MANAGER_KEY = 'transaction_manager'**

Constant for *transaction_manager*

owmeta_core.data.**TRANSACTION_MANAGER_PROVIDER_KEY = 'transaction_manager.provider'**

Constant for *transaction_manager.provider*

owmeta_core.dataobject module

exception owmeta_core.dataobject.**ClassResolutionFailed**

Bases: Exception

Thrown when a *PythonClassDescription* can't resolve its class

exception owmeta_core.dataobject.**ModuleResolutionFailed**

Bases: Exception

Thrown when a *PythonModule* can't resolve its module

class owmeta_core.dataobject.**Alias**(*target*)

Bases: object

Used to declare that a descriptor is an alias to some other Property

Example usage:

```
class Person(DataObject):
    child = DatatypeProperty()
    offspring = Alias(child)
```

Parameters

target
[dataobject_property.Property] The property to alias

class owmeta_core.dataobject.**BaseDataObject**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *IdMixin*, *GraphObject*, ContextualizableDataUserMixin

An object which can be mapped to an RDF graph

Attributes

rdf_type
[rdflib.term.URIRef] The RDF type URI for objects of this type

rdf_namespace
[rdflib.namespace.Namespace] The rdflib namespace (prefix for URIs) for instances of this class

schema_namespace
[rdflib.namespace.Namespace] The rdflib namespace (prefix for URIs) for types that are part of this class' schema

properties
[list of *owmeta_core.dataobject_property.Property* or *owmeta_core.custom_dataobject_property.CustomProperty*] Properties belonging to this object

owner_properties
[list of *owmeta_core.dataobject_property.Property* or *owmeta_core.custom_dataobject_property.CustomProperty*] Properties belonging to parents of this object

properties_are_init_args
[bool] If true, then properties defined in the class body can be passed as keyword arguments to __init__. For example:

```
>>> class A(DataObject):
...     p = DatatypeProperty()

>>> A(p=5)
```

If the arguments are written explicitly into the __init__ method definition, then no special processing is done.

`classmethod DatatypeProperty`(**args*, ***kwargs*)

Attach a, possibly new, property to this class that has a simple type (string, number, etc) for its values

Parameters

linkName
[str] The name of this property.

owner
[*owmeta_core.dataobject.BaseDataObject*] The owner of this property.

`classmethod ObjectProperty`(**args*, ***kwargs*)

Attach a, possibly new, property to this class that has a *BaseDataObject* for its values

Parameters

linkName
[str] The name of this property.

owner
[*owmeta_core.dataobject.BaseDataObject*] The owner of this property.

value_type
[type] The type of BaseDataObject for values of this property

classmethod UnionProperty(**args*, ***kwargs*)

Attach a, possibly new, property to this class that has a simple type (string,number,etc) or *BaseDataObject* for its values

Parameters

linkName
[str] The name of this property.

owner
[*owmeta_core.dataobject.BaseDataObject*] The owner of this property.

attach_property(*prop_cls*, *name=None*, *ephemeral=False*, ***kwargs*)

Parameters

prop_cls
[type] The property class to attach to this dataobject

name
[str, optional] The name to use for attaching to this dataobject

ephemeral
[bool, optional] If True, the property will not be set as an attribute on the object

****kwargs**
Arguments to pass to the initializer of the property class

contextualize_augment(*context*)

For MappedClass, rdf_type and rdf_namespace have special behavior where they can be auto-generated based on the class name and base_namespace. We have to pass through these values to our "proxy" to avoid this behavior

get_owners(*property_class_name*)

Return a generator of owners along a property pointing to this object

graph_pattern(*shorten=False*, *show_namespaces=True*, ***kwargs*)

Get the graph pattern for this object.

It should be as simple as converting the result of triples() into a BGP

Parameters

shorten
[bool] Indicates whether to shorten the URLs with the namespace manager attached to the `self`

hashfun()

Returns a md5 hash object; optionally initialized with a string

id_is_variable()

Is the identifier a variable?

load(*graph=None*)

Loads *DataObjects* by matching between the object graph and the RDF graph

Parameters

graph
[rdflib.graph.ConjunctiveGraph] the RDF graph to load from

load_one(*graph=None*)

Load a single *DataObject*

load_terms(*graph=None*)

Loads URIs by matching between the object graph and the RDF graph

Parameters

graph
[rdflib.graph.ConjunctiveGraph] the RDF graph to load from

make_key_from_properties(*names*)

Creates key from properties

retract()

Remove this object from the data store.

save()

Write in-memory data to the database. Derived classes should call this to update the store.

property expr

Create a query expression rooted at this object

property rdf

Returns either the configured RDF graph or the `Context.rdf_graph` of its context

property rdfs_comment

Corresponds to the rdfs:comment predicate

property rdfs_label

Corresponds to the rdfs:label predicate

property rdfs_member

Corresponds to the rdfs:member predicate

class owmeta_core.dataobject.**ClassDescription**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Describes a class in the programming language.

Note that, in other languages, there may not actually be classes per se. In such cases, the *ClassDescription* may instead indicate a function. The conventions for how that function accepts a URI for the sake of creating an "instance" of is up to the associated software module.

property module

The module the class belongs to

class owmeta_core.dataobject.**ContextMappedClass**(*name*, *typ*, *dct*)

Bases: *MappedClass*, *ContextualizableClass*

The metaclass for a *BaseDataObject*.

augment_rdf_type_object(*rdf_type_object*)

Runs after initialization of the rdf_type_object

contextualize_class_augment(*context*)

For MappedClass, rdf_type and rdf_namespace have special behavior where they can be auto-generated based on the class name and base_namespace. We have to pass through these values to our "proxy" to avoid this behavior

property definition_context

Unlike self.context, definition_context isn't meant to be overriden

property query

Creates a proxy that changes how some things behave for purposes of querying

class owmeta_core.dataobject.**ContextualizableList**(**args*, ***kwargs*)

Bases: *Contextualizable*, list

A Contextualizable list

class owmeta_core.dataobject.**DataObject**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataObject*

An object that can be mapped to an RDF graph

class owmeta_core.dataobject.**Module**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Represents a module of code

Most modern programming languages organize code into importable modules of one kind or another. This is basically the nearest level above a *class* in the language.

Modules are accessible by one or more *ModuleAccessor*

property accessor

Describes a way to get the module

property package

Package that provides the module

class owmeta_core.dataobject.**ModuleAccessor**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Describes how to access a module.

Module access is how a person or automated system brings the module to where it can be imported/included, possibly in a subsequent

help_str()

Format a string to show how to access the module by installing it or requiring it or whatever.

Default implementation just returns an empty string

class owmeta_core.dataobject.**OptionalKeyValue**(*prop*)

Bases: object

An optional key value to use in key_properties

class owmeta_core.dataobject.**PIPInstall**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *ModuleAccessor*

Describes a pip install command line

property index_url

URL of the index from which the package should be retrieved

class owmeta_core.dataobject.**Package**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Describes an idealized software package identifiable by a name and version number

property name

The standard name of the package

property version

The version of the package

class owmeta_core.dataobject.**PythonClassDescription**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *ClassDescription*

Description for a Python class

resolve_class()

Load the class described by this object

Returns

type
The class described by this object

Raises

ClassResolutionFailed
Raised if the class can't be resolved for whatever reason

property module

The module the class belongs to

property name

Local name of the class (i.e., relative to the module name)

class owmeta_core.dataobject.**PythonModule**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *Module*

A Python module

resolve_module()

Load the module referenced by this object

Returns

types.ModuleType
The module referenced by this object

Raises

ModuleResolutionFailed
Raised if the class can't be resolved for whatever reason

property name

The full name of the module

class owmeta_core.dataobject.**PythonPackage**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *Package*

A Python package

class owmeta_core.dataobject.**RDFProperty**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataObject*

The *DataObject* corresponding to rdf:Property

property rdfs_subpropertyof

Corresponds to the rdfs:subPropertyOf predidcate

class owmeta_core.dataobject.**RDFSClass**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataObject*

The GraphObject corresponding to rdfs:Class

property rdfs_subclassof_property

Corresponds to the rdfs:subClassOf predidcate

class owmeta_core.dataobject.**RDFSCommentProperty**(**args*, ***kwargs*)

Bases: DatatypeProperty

Corresponds to the rdfs:comment predicate

Parameters

resolver
[RDFTypeResolver] Resolves RDF identifiers returned from get() into objects

owner_type

alias of *BaseDataObject*

class owmeta_core.dataobject.**RDFSLabelProperty**(**args*, ***kwargs*)

Bases: DatatypeProperty

Corresponds to the rdfs:label predicate

Parameters

resolver
[RDFTypeResolver] Resolves RDF identifiers returned from get() into objects

owner_type

alias of *BaseDataObject*

class owmeta_core.dataobject.**RDFSMemberProperty**(**args*, ***kwargs*)

Bases: UnionProperty

Corresponds to the rdfs:member predicate

Parameters

resolver
[RDFTypeResolver] Resolves RDF identifiers into objects returned from get()

owner_type

alias of *BaseDataObject*

class owmeta_core.dataobject.**RDFSSubClassOfProperty**(**args*, ***kwargs*)

Bases: ObjectProperty

Corresponds to the rdfs:subClassOf predidcate

owner_type

alias of *RDFSClass*

value_type

alias of *RDFSClass*

class owmeta_core.dataobject.**RDFSSubPropertyOfProperty**(**args*, ***kwargs*)

Bases: ObjectProperty

Corresponds to the rdfs:subPropertyOf predidcate

owner_type

alias of *RDFProperty*

value_type

alias of *RDFProperty*

class owmeta_core.dataobject.**RDFTypeProperty**(**args*, ***kwargs*)

Bases: ObjectProperty

Corresponds to the rdf:type predidcate

owner_type

alias of *BaseDataObject*

class owmeta_core.dataobject.**RegistryEntry**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

A mapping from a class in the programming language to an RDF class.

Objects of this type are utilized in the resolution of classes from the RDF graph

property class_description

The description of the class

property rdf_class

The RDF (Resource Description Framework) type for the class

We use rdf_type for the type of a *DataObject* (RegistryEntry.rdf_type in this case), so we call this *rdf_class* to avoid the conflict

owmeta_core.dataobject.**DatatypeProperty**(**args*, ***kwargs*)

Used in a *DataObject* implementation to designate a property whose values are not *DataObjects*.

An example *DatatypeProperty* use:

```
class Person(DataObject):
    name = DatatypeProperty()
    age = DatatypeProperty()

Person(name='Abioye', age=34)
```

owmeta_core.dataobject.**ObjectProperty**(**args*, ***kwargs*)

Used in a *DataObject* implementation to designate a property whose values are other *DataObjects*.

An example *ObjectProperty* use:

```
class Person(DataObject):
    name = DatatypeProperty()
    friend = ObjectProperty()

Person(name='Abioye', friend=Person(name='Baako'))
```

owmeta_core.dataobject.**UnionProperty**(**args*, ***kwargs*)

Used in a *DataObject* implementation to designate a property whose values are either other *DataObjects* or literals (e.g., str, int).

An example *UnionProperty* use:

```
class Address(DataObject):
    street = DatatypeProperty()
    number = DatatypeProperty()
    city = DatatypeProperty()
    state = DatatypeProperty()
    zip = DatatypeProperty()

class Person(DataObject):
    name = DatatypeProperty()
    address = UnionProperty()

Person(name='Umoja', address='38 West 88th Street, Manhattan NY 10024 , New York,␣
↪USA')
Person(name='Umoja', address=Address(number=38,
                                     street='West 88th Street',
                                     city='New York',
                                     state='NY',
                                     zip=10024))
```

owmeta_core.dataobject.**DATAOBJECT_PROPERTY_NAME_PREFIX = '_owm_'**

Prefix for property attribute names

owmeta_core.dataobject.**This = <object object>**

A reference to be used in class-level property declarations to denote the class currently being defined. For example:

```
>>> class Person(DataObject):
...     parent = ObjectProperty(value_type=This,
...                             inverse_of=(This, 'child'))
...     child = ObjectProperty(value_type=This)
```

owmeta_core.dataobject_property module

class owmeta_core.dataobject_property.**ContextMappedPropertyClass**(*name*, *typ*, *dct*)

Bases: *MappedClass*, *ContextualizableClass*

Meta-class for *Property*.

A few attributes can be specified in the class body which affect how the created type is set up: these are defined in the "Attributes" section.

One aspect in particular is important: a *Property* class can represent a single type of property where all instances have the same URI, or a *Property* can represent an a class of RDF properties where the instances have distinct URIs and correspond to instances of the RDF type. An instance of the latter is demonstrated with *ContainerMembershipProperty*.

Attributes

rdf_type_class
[type] A sub-class of *DataObject* to use as the type. If set, this will be used instead of what *init_rdf_type_object* would create.

rdf_type
[str or URIRef] The RDF type for the *Property*. Must be defined for *init_rdf_type_object* to actually create the rdf type object

rdf_type_object_deferred
[bool] If True, defer calling *init_rdf_type_object* until it's explicitly called rather than during normal class init. Useful for cases where *init_rdf_type_object* uses types that aren't defined at the point where the *Property* is defined.

rdf_object
[RDFProperty] An instance of RDFProperty corresponding to this class. If set, this will be used instead of what init_rdf_object would create.

rdf_object_deferred
[bool] If True, defer calling init_rdf_object until it is explicitly called rather than during normal class init. Useful for cases where init_rdf_object uses types that aren't defined at the point where the *Property* is defined.

contextualize_class_augment(*context*)

For MappedClass, rdf_type and rdf_namespace have special behavior where they can be auto-generated based on the class name and base_namespace. We have to pass through these values to our "proxy" to avoid this behavior

init_rdf_type_object()

Initializes rdf_type_class and thereby initializes the rdf_type_object

Sometimes, we actually use Property sub-classes *as* rdf:Property classes (e.g., rdfs:ContainerMembershipProperty). The rdf_type attribute has to be defined on this class if we're going to use it as an rdf:Property class.

class owmeta_core.dataobject_property.**ExprResultObj**(*expr*, *ident*)

Bases: object

Object returned by *PropertyExpr.to_objects*. Attributes for which *PropertyExpr.to_dict* has been called can be accessed on the object. For example we can print out the b properties of instances of a class A:

```
class B(DataObject):
    v = DatatypeProperty()

class A(DataObject):
    b = ObjectProperty(value_type=B)

a = A().a.expr
a.b.v()
for anA in a.to_objects():
    print(anA.identifier, anA.b)
```

anA is an *ExprResultObj* in the example. The

property(*property_class*)

Return the results object for this sub-expression

Parameters

property_class
[*Property*, *Property* sub-class, URIRef, or str]

property rdf_type

Allias to rdf_type_property

class owmeta_core.dataobject_property.**Property**(**args*, ***kwargs*)

Bases: *DataUser*, *Contextualizable*

A property attached to a *DataObject*.

clear()

Clears values set *in all contexts*

contextualize_augment(*context*)

For MappedClass, rdf_type and rdf_namespace have special behavior where they can be auto-generated based on the class name and base_namespace. We have to pass through these values to our "proxy" to avoid this behavior

get_terms()

Get the Node instances matching this property query

has_defined_value()

Returns True if this property has a value in the current context which is either a GraphObject with defined set to True or a literal value

has_value()

Returns True if there is a value set on this property in the currrent context

one()

Query for a single value from this property.

For a multi-valued property, the returned value is chosen arbitrarily. If there's no value returned from the query, then None is returned.

onedef()

Return a single defined value set on this property in the current context

This does not execute a query, but returns a value which was set on this property.

set(*v*)

Set the value for or add a value to this property

unset(*v*)

Remove a from this property

property defined_values

The "defined" values set on this property in the current context

property expr

An query expression from this property

property identifier

Alias to link

lazy = True

If True, then the property is not attached to an instance until the property is set or queried.

multiple = False

If True, then the property will only maintain a single staged value at a time. No effort is made to check how many values are stored in the RDF graph.

property values

Return all values set on this property in the current context

class owmeta_core.dataobject_property.**PropertyExpr**(*props*, *triples_provider=None*, *terms_provider=None*, *origin=None*)

Bases: object

A property expression

property(*property_class*)

Create a sub-expression with the given property.

Allows for creating expressions with properties that are not necessarily declared for the value_type of this expression's property

to_dict(*multiple=False*)

Return a dict mapping from identifiers for subjects of this expression's property to the objects for that property.

Parameters

multiple
[bool, optional] If False, then only a single object is allowed for each subject in the results. An exception is raised if more than one object is found for a given subject.

to_objects()

Returns a list of *ExprResultObj* that allow for retrieving results in a convenient attribute traversal

to_terms()

Return a list of rdflib.term.Node terms produced by this expression.

property rdf_type

Short-hand for rdf_type_property

owmeta_core.datasource module

exception owmeta_core.datasource.**ExtraSourceFound**

Bases: Exception

Raised by *transform* when more than one source is found in the current context

exception owmeta_core.datasource.**NoSourceFound**

Bases: Exception

Raised by *transform* when a source cannot be found in the current context

exception owmeta_core.datasource.**NoTranslatorFound**

Bases: Exception

Raised by *transform* when a translator cannot be found in the current context

class owmeta_core.datasource.**BaseDataTranslator**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataTransformer*

Input type(s): *DataSource*

Output type(s): *DataSource*

make_transformation(*sources=()*)

Just calls *make_translation* and returns its result.

make_translation(*sources=()*)

It's intended that implementations of *BaseDataTranslator* will override this method to make custom *Translations* according with how different arguments to *translate* are (or are not) distinguished.

The actual properties of a *Translation* subclass must be assigned within the *translate* method

Parameters

sources
[tuple] The sources that go into the translation. Sub-classes may choose to pass these to their superclass' make_translation method or not.

Returns

a description of the translation

transform(**args*, ***kwargs*)

Just calls *translate* and returns its result.

translate(**args*, ***kwargs*)

Notionally, this method takes one or more data sources, and translates them into some other data source that captures essentially the same information, but, possibly, in a different format. Additional sources can be passed in as well for auxiliary information which are not "translated" in their entirety into the output data source. Such auxiliarry data sources should be distinguished from the primary ones in the translation

Parameters

***args**
Input data sources

****kwargs**
Named input data sources

Returns

the output data source

class owmeta_core.datasource.**DataObjectContextDataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataSource*

Input source
[*ObjectProperty*] Attribute: source

The data source that was translated into this one

Transformation
[*ObjectProperty*] Attribute: transformation

Information about the transformation process that created this object

Translation
[*ObjectProperty*] Attribute: translation

Information about the translation process that created this object

Description
[*DatatypeProperty*] Attribute: `description`

Free-text describing the data source

class owmeta_core.datasource.**DataSource**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

A source for data that can get translated into owmeta_core objects.

The value for any field can be passed to `__init__` by name. Additionally, if the sub-class definition of a DataSource assigns a value for that field like:

```
class A(DataSource):
    some_field = 3
```

that value will be used over the default value for the field, but not over any value provided to `__init__`.

after_transform()

Called after `Transformer.transform`.

This method should handle any of the things that should happen for an output data source after `Transformer.transform` (or `Translator.translate`). This can include things like flushing output to files, closing file handles, and writing triples in a Context.

NOTE: Be sure to call this method via super() in sub-classes

identifier_augment()

It doesn't make much sense to have translation and transformation set, so we just take the first of them

description

"Description", a *DatatypeProperty*: Free-text describing the data source

source

"Input source", a *ObjectProperty*: The data source that was translated into this one

transformation

"Transformation", a *ObjectProperty*: Information about the transformation process that created this object

translation

"Translation", a *ObjectProperty*: Information about the translation process that created this object

class owmeta_core.datasource.**DataSourceType**(*name*, *typ*, *dct*)

Bases: *ContextMappedClass*

A type for DataSources

Sets up the graph with things needed for MappedClasses

class owmeta_core.datasource.**DataTransformer**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Transforms one or more *DataSources* to one or more other *DataSources*

Attributes

input_type
[`type` or `tuple` `of` `type`] A source for data that can get translated into owmeta_core objects.

output_type
[`type` or `tuple` `of` `type`] A source for data that can get translated into owmeta_core objects.

transformation_type
[type] Record of the how a *DataSource* was produced and the sources of the transformation that produced it.

output_key
[str] The "key" for outputs from this transformer (see `IdentifierMixin`). Normally only defined during execution of __call__

output_identifier
[str] The identifier for outputs from this transformer. Normally only defined during execution of __call__

input_type

alias of *DataSource*

output_type

alias of *DataSource*

transformation_type

alias of *Transformation*

after_transform()

Called after *transform* runs in `__call__` and after the result *DataSource.after_transform* is called.

make_new_output(*sources*, **args*, ***kwargs*)

Make a new output *DataSource*. Typically called within *transform*.

make_transformation(*sources=()*)

It's intended that implementations of *DataTransformer* will override this method to make custom *Transformations* according with how different arguments to *transform* are (or are not) distinguished.

The actual properties of a *Transformation* subclass must be assigned within the *transform* method

transform(**args*, ***kwargs*)

Notionally, this method takes a data source, which is transformed into some other data source. There doesn't necessarily need to be an input data source.

Parameters

***args**
Input data sources

****kwargs**
Named input data sources

Returns

the output data source

transform_with(*translator_type*, **sources*, *output_key=None*, *output_identifier=None*, ***named_sources*)

Transform with the given *DataTransformer* and sources.

This should be used in a *transform* implementation to compose multiple transformations. An instance of the transformer will be created and contextualized with the *this* transformer's context unless the given transformer already has a context.

class owmeta_core.datasource.**DataTranslator**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataTranslator*

A specialization with the *GenericTranslation* translation type that adds sources for the translation automatically when a new output is made

translation_type

alias of *GenericTranslation*

class owmeta_core.datasource.**GenericTranslation**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *Translation*

A generic translation that just has sources in any order

class owmeta_core.datasource.**OneOrMore**(*source_type*)

Bases: object

Wrapper for *DataTransformer* input *DataSource* types indicating that one or more of the wrapped type must be provided to the translator

class owmeta_core.datasource.**PersonDataTranslator**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *BaseDataTranslator*

A person who was responsible for carrying out the translation of a data source manually

property person

A person responsible for carrying out the translation.

class owmeta_core.datasource.**Transformation**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *DataObject*

Record of the how a *DataSource* was produced and the sources of the transformation that produced it. Unlike the 'source' field attached to DataSources, the Translation may distinguish different kinds of input source to a transformation.

class owmeta_core.datasource.**Translation**(**args*, *no_type_decl=False*, ***kwargs*)

Bases: *Transformation*

A transformation where, notionally, the general character of the input is preserved.

In contrast to just a transformation, a translation wouldn't just pick out, say, one record within an input source containing several, but would have an output source with o

owmeta_core.datasource.**transform**(*transformer*, *output_key=None*, *output_identifier=None*, *data_sources=()*, *named_data_sources=None*)

Do a translation with the named translator and inputs

Parameters

transformer
[*DataTransformer*] transformer to execute

output_key
[str] Output key. Used for generating the output's identifier. Exclusive with output_identifier

output_identifier
[str] Output identifier. Exclusive with output_key

data_sources
[list of *DataSource*] Input data sources

named_data_sources
[dict] Named input data sources

Raises

NoTranslatorFound
when a translator is not found

NoSourceFound
when a source cannot be looked up in the given context

ExtraSourceFound
when a more than one source is found in the given context for the given source identifier

owmeta_core.datasource_loader module

DataSourceLoaders take a DataSource and retrieve the primary data (e.g., CSV files, electrode recordings) from some location (e.g., a file store, via a bittorrent tracker).

Each loader can treat the base_directory given as its own namespace and place directories in there however it wants.

exception owmeta_core.datasource_loader.**LoadFailed**(*data_source*, *loader*, **args*)

Bases: Exception

Thrown when loading fails for a .DataSourceDirLoader

Parameters

data_source
[*DataSource*] The *DataSource* on which loading was attempted

loader
[*DataSourceDirLoader*] The loader that attempted to load the data source

args[0]
[str] Message explaining why loading failed

args[1:]
Passed on to Exception

class owmeta_core.datasource_loader.**DataSourceDirLoader**(*base_directory=None*, *directory_key=None*)

Bases: object

Loads data files for a DataSource

The loader is expected to organize files for each data source within the given base directory.

__call__(*data_source*)

Load the data source. Calls *load*

Parameters

data_source
[*DataSource*] The data source to load files for

Returns

str
A path to the loaded resource

Raises

LoadFailed
If *load*:

- throws an exception
- doesn't return anything
- returns a path that isn't under base_directory

- returns a path that doesn't exist

`can_load`(*data_source*)

Returns true if the *DataSource* can be loaded by this loader

Parameters

data_source
[*DataSource*] The data source to load files for

`load`(*data_source*)

Loads the files for the data source

Parameters

data_source
[*DataSource*] The data source to load files for

Returns

`str`
A path to the loaded resource

owmeta_core.docscrape module

A replpacement for numpydoc's docscrape that doesn't require numpydoc's time-consuming imports

`class` owmeta_core.docscrape.**ParamInfo**(*name*, *val_type*, *desc*)

Bases: `tuple`

Create new instance of ParamInfo(name, val_type, desc)

`property desc`

Alias for field number 2

`property name`

Alias for field number 0

`property val_type`

Alias for field number 1

owmeta_core.file_lock module

`exception` owmeta_core.file_lock.**InvalidLockAccess**

Bases: `Exception`

Raised when attempt to do something improper with a lock like releasing the lock when you haven't yet acquired it.

owmeta_core.file_match module

owmeta_core.file_utils module

owmeta_core.file_utils.**hash_file**(*hsh*, *fname*, *blocksize=None*)

Updates the given hash object with the contents of a file.

The file is read in `blocksize` chunks to avoid eating up too much memory at a time.

Parameters

hsh
[`hashlib.hash`] The hash object to update

fname
[`str`] The filename for the file to hash

blocksize
[`int`, `optional`] The number of bytes to read at a time. If not provided, will use `hsh.block_size` instead.

owmeta_core.git_repo module

class owmeta_core.git_repo.**GitRepoProvider**

Bases: `object`

Provides a project repository for *OWM* backed by a Git repository

clone(*url*, *base*, *progress=None*, ***kwargs*)

Parameters

url
[`str`] URL to clone from

base
[`str`] Directory to clone into

progress
[`tqdm.tqdm`-like] Must support a `progress.update` method accepting the amount to add to total progress (see https://tqdm.github.io/docs/tqdm/#update)

owmeta_core.graph_object module

exception owmeta_core.graph_object.**IdentifierMissingException**(*dataObject='[unspecified object]'*, **args*, ***kwargs*)

Bases: `Exception`

Indicates that an identifier should be available for the object in question, but there is none

class owmeta_core.graph_object.**ComponentTripler**(*start*, *traverse_undefined=False*, *generator=False*)

Bases: `object`

Gets a set of triples that are connected to the given object by objects which have an identifier.

The ComponentTripler does not query against a backing graph, but instead uses the properties attached to the object.

class owmeta_core.graph_object.**DescendantTripler**(*start*, *graph=None*, *transitive=True*)

Bases: object

Gets triples that the object points to, optionally transitively.

Parameters

start
[*GraphObject*] the node to start from

graph
[rdflib.graph.Graph, optional] if given, the graph to draw descedants from. Otherwise the object graph is used

class owmeta_core.graph_object.**GraphObject**(***kwargs*)

Bases: object

An object which can be included in the object graph.

An abstract base class.

variable()

Must return a *Variable* object that identifies this *GraphObject* in queries.

The variable can be randomly generated when the object is created and stored in the object.

property defined

Returns true if an *identifier()* would return an identifier

property identifier

Must return an object representing this object or else raise an Exception.

class owmeta_core.graph_object.**GraphObjectChecker**(*query_object*, *graph*, *sort_first=False*)

Bases: object

Checks the graph of defined GraphObjects for

class owmeta_core.graph_object.**GraphObjectQuerier**(*q*, *graph*, *hop_scorer=None*)

Bases: object

Performs queries for objects in the given graph.

The querier queries for objects at the center of a star graph. In SPARQL, the query has the form:

```
SELECT ?x WHERE {
    ?x  <p1> ?o1 .
    ?o1 <p2> ?o2 .
     ...
    ?on <pn> <a> .

    ?x  <q1> ?n1 .
    ?n1 <q2> ?n2 .
     ...
    ?nn <qn> <b> .
}
```

It is allowed that `<px> == <py>` for `x != y`.

Queries such as:

```
SELECT ?x WHERE {
    ?x  <p1> ?o1 .
     ...
    ?on <pn>  ?y .
}
```

or:

```
SELECT ?x WHERE {
    ?x  <p1> ?o1 .
     ...
    ?on <pn>  ?x .
}
```

or:

```
SELECT ?x WHERE {
    ?x  ?z ?o .
}
```

or:

```
SELECT ?x WHERE {
    ?x  ?z <a> .
}
```

are not supported and will be ignored without error.

Call the GraphObjectQuerier object to perform the query.

Parameters

q
[*GraphObject*] The object which is queried on

graph
[object] The graph from which the objects are queried. Must implement a method `triples()` that takes a triple pattern, `t`, and returns a set of triples matching that pattern. The pattern for `t` is `t[i] = None`, 0 <= i <= 2, indicates that the i'th position can take any value.

The `graph` method can optionally implement the 'range query' 'interface': the graph must have a property `supports_range_queries` equal to `True` and `triples()` must accept an *InRange* object in the object position of the query triple, but only for literals

hop_scorer
[callable()] Returns a score for a hop (a four-tuple, (subject, predicate, object, target)) indicating how selective the query would be for that hop, with lower numbers being more selective. In general the score should only take the given hop into account – it should not take previously given hops into account when calculating a score.

`merge_paths`(*l*)

Combines a list of lists into a multi-level table with the elements of the lists as the keys. For given:

```
[[a, b, c], [a, b, d], [a, e, d]]
```

merge_paths returns:

```
{a: {b: {c: {},
         d: {}},
     e: {d: {}}}}
```

class owmeta_core.graph_object.**LegendFinder**(*start*, *graph=None*)

Bases: object

Gets a list of the objects which can not be deleted freely from the transitive closure.

Essentially, this is the 'mark' phase of the "mark-and-sweep" garbage collection algorithm.

"Heroes get remembered, but legends never die."

class owmeta_core.graph_object.**Variable**

Bases: int

A marker used in *GraphObjectQuerier* for variables in a query

owmeta_core.graph_serialization module

Utilies for graph serialization

owmeta_core.graph_serialization.**write_canonical_to_file**(*graph*, *file_name*)

Write a graph to a file such that the contents would only differ if the set of triples in the graph were different. The serialization format is N-Triples.

Parameters

graph
[rdflib.graph.Graph] The graph to write

file_name
[str] The name of the file to write to

owmeta_core.identifier_mixin module

class owmeta_core.identifier_mixin.**IdMixin**(*ident=None*, *key=None*, **args*, *direct_key=None*, ***kwargs*)

Bases: object

Mixin that provides common identifier logic

Attributes

hashfun
[function] Returns a sha224 hash object; optionally initialized with a string

rdf_namespace
[rdflib.namespace.Namespace] The namespace for identifiers created

direct_key
[bool] Whether to make a key directly, just adding the string onto the namespace or indirectly by hashing the key before joining with the namespace.

defined_augment()

This fuction must return False if *identifier_augment()* would raise an *IdentifierMissingException*. Override it when defining a non-standard identifier for subclasses of DataObjects.

hashfun()

Returns a sha224 hash object; optionally initialized with a string

identifier_augment()

Override this method to define an identifier in lieu of one explicity set.

One must also override *defined_augment()* to return True whenever this method could return a valid identifier. *IdentifierMissingException* should be raised if an identifier cannot be generated by this method.

Raises

IdentifierMissingException

classmethod make_identifier(*data*)

Makes an identifier based on this class' `rdf_namespace` by calling `__str__` on the data and passing to the class' *hashfun*.

If the `__str__` for data's type doesn't function as an identifier, you should use either *make_identifier_direct()* or override *identifier_augment()* and *defined_augment()*

classmethod make_identifier_direct(*string*)

Make identifier by using the `quote`'d value of `key` appended to the `rdf_namespace` value

property identifier

The identifier

owmeta_core.inverse_property module

For declaring inverse properties of GraphObjects

class owmeta_core.inverse_property.**InversePropertyMixin**

Bases: `object`

Mixin for inverse properties.

Augments Property methods to update inverse properties as well

owmeta_core.json_schema module

exception owmeta_core.json_schema.**AssignmentValidationException**

Bases: *ValidationException*

Raised when an attempt is made to assign an inappropriate value with *Creator*

exception owmeta_core.json_schema.**SchemaException**

Bases: `Exception`

Raised for an invalid input given to *TypeCreator*

exception owmeta_core.json_schema.**ValidationException**

Bases: `Exception`

Raised for an invalid input given to *Creator*

class owmeta_core.json_schema.**Creator**(*schema*)

Bases: object

Creates objects based on a JSON schema augmented with type annotations as would be produced by *TypeCreator*

Currently, only annotations for JSON objects are supported. In the future, conversions for all types (arrays, numbers, ints, strings) may be supported.

Takes a schema annotated with '_owm_type' entries indicating which types are expected at each position in the object and produces an instance of the root type described in the schema

Parameters

schema
[dict] The annotated schema

assign(*obj*, *name*, *value*)

Assign the given value to a property with the given name on the object

Parameters

obj
[object] The object to receive the assignment

name
[str] The name on the object to assign to

value
[object] The value to assign

create(*instance*, *ident=None*)

Creates an instance of the root OWM type given a deserialized instance of the type described in our JSON schema.

A context can be passed in and it will be used to contextualize the OWM types

Parameters

instance
[dict] The JSON object to create from

context
[*owmeta_core.context.Context*] The context in which the object should be created

Raises

ValidationException
Raised when there's an error with the given instance compared to the schema

fill_in(*target*, *instance*, *ident=None*)

"Fill-in" an already existing target object with JSON matching a schema

make_instance(*owm_type*)

Make an instance of the given type

Parameters

owm_type
[type] The type for which an instance should be made

class owmeta_core.json_schema.**DataObjectTypeCreator**(**args*, *module*, *context=None*, ***kwargs*)

Bases: *TypeCreator*

Creates DataObject types from a JSON Schema

Attributes

cdict
[dict] Map from paths in the schema to the dictionaries that will be passed into the class definition. The path is the same as passed into create_type

module
[str] The module in which classes will be defined

Parameters

module
[str] The module in which classes will be defined

context
[*owmeta_core.context.Context* or str] The class context in which the various types will be declared

determine_property_type(*path*, *k*, *v*)

Determine the type of property created by proc_prop

select_base_types(*path*, *schema*)

Returns the base types for create_type

Parameters

path
[tuple] The path to the sub-schema

schema
[dict] The sub-schema at the path location

class owmeta_core.json_schema.**DataSourceTypeCreator**(**args*, *module*, *context=None*, ***kwargs*)

Bases: *DataObjectTypeCreator*

Creates DataSource types from a JSON Schema

Parameters

module
[str] The module in which classes will be defined

context
[*owmeta_core.context.Context* or str] The class context in which the various types will be declared

select_base_types(*path*, *schema*)

Returns the base types for create_type

Parameters

path
[tuple] The path to the sub-schema

schema
[dict] The sub-schema at the path location

class owmeta_core.json_schema.**TypeCreator**(*name*, *schema*, *definition_base_name=''*)

Bases: object

Creates OWM types from a JSON schema and produces a copy of the schema annotated with the created types.

Parameters

name
[str] The name of the root class and the base-name for all classes derived from a schema's properties

schema
[dict] A JSON schema as would be returned by json.load()

definition_base_name
[str] The base-name for types defined in the schema's definitions. optional. By default, definitions just take the capitalized form of their key in the "definitions" block

annotate()

Returns the annotated JSON schema

create_type(*path*, *schema*)

Create the OWM type.

At this point, the properties for the schema will already be created.

Parameters

path
[tuple] The path to the type

schema
[dict] The JSON schema that applies to this type

extract_name(*path*)

Generates a class name from the path to the sub-schema

Parameters

path
[tuple] Path to the sub-schema

proc_prop(*path*, *key*, *value*)

Process property named `key` with the given `value`.

The `path` will not include the key but will be the path of the definition that contains the property. For example, in:

```
{"$schema": "http://json-schema.org/schema",
 "title": "Example Schema",
 "type": "object",
 "properties": {"data": {"type": "object",
                         "properties": {
                            "data_data": {"type": "string"}
                         }}}}
```

proc_prop would be called as `.proc_prop((), 'data', {'type': 'object', ...})` for `data`, but for `data_data`, it would be called like `.proc_prop(('properties', 'data'), 'data_data', {'type': 'string'})`

Parameters

path
[tuple] The path to the given property.

key
[str] The name of the property

value
[dict] the definition of the property

classmethod retrieve_type(*annotated_schema*, *pointer=''*)

Look up the type created for the object at the given JSON pointer location

Parameters

annotated_schema
[dict] Annotated schema as returned from *annotate*

pointer
[str, optional] JSON pointer to the schema/sub-schema

Returns

type
The type at the given JSON pointer location

Raises

LookupError
Raised when the pointer has no referent in the given document or there's type associated with the referent

owmeta_core.json_schema.**resolve_fragment**(*document*, *fragment*)

Resolve a `fragment` within the referenced `document`.

Parameters

document
[object] The referent document. Typically a collections.abc.Mapping (e.g., a dict) or collections.abc.Sequence, but if fragment is #, then the document is returned unchanged.

fragment
[str] a URI fragment to resolve within it

Returns

object
The part of the document referred to

owmeta_core.json_schema.**resolve_json_pointer**(*document*, *pointer*)

Resolve a `fragment` within the referenced `document`.

Parameters

document
[object] The referent document. Typically a collections.abc.Mapping (e.g., a dict) or collections.abc.Sequence, but if fragment is #, then the document is returned unchanged.

pointer
[str] a JSON pointer to resolve in the document

Returns

object
The part of the document referred to

owmeta_core.mapped_class module

class owmeta_core.mapped_class.**MappedClass**(*name*, *bases*, *dct*)

Bases: type

A type for MappedClasses

Sets up the graph with things needed for MappedClasses

on_mapper_add_class(*mapper*)

Called by *owmeta_core.mapper.Mapper*

Registers certain properties of the class

register_on_module(*module=None*)

"Registers" this class on a module (typically the one in which the class is defined) such that owmeta-core functions can locate it. This happens automatically when the class is defined unless the 'unmapped' attribute is defined and set to True.

This mechanism necessary in some cases where classes are generated dynamically or in a method and aren't necessarily assigned to attributes on the module where they are defined.

owmeta_core.mapper module

exception owmeta_core.mapper.**ClassRedefinitionAttempt**(*mapper*, *maybe_cls*, *cls*)

Bases: Exception

Thrown when a *Mapper.add_class* is called on a class when a class with the same name has already been added to the mapper

class owmeta_core.mapper.**Mapper**(*name=None*, *class_registry_context=None*, *class_registry_context_list=None*, ***kwargs*)

Bases: *Configurable*

Keeps track of relationships between Python classes and RDF classes

The mapping this object manages may also be written to the RDF graph as `class registry entries`. The entries are written to the "class registry context", which can be specified when the Mapper is created.

Parameters

name
[str, optional] Name of the mapper for diagnostic/debugging purposes

class_registry_context
[*owmeta_core.context.Context* or str, optional] The context where mappings should be saved and/or retrieved from. Either the context object itself or the ID for it. If not provided, then the class registry context ID is looked up from the Mapper's configuration at *CLASS_REGISTRY_CONTEXT_KEY*

class_registry_context_list
[list of *owmeta_core.context.Context* or str, optional] List of contexts or context IDs where registry entries should be retrieved from if the class_registry_context doesn't yield a mapping

****kwargs**
passed to super-classes

add_class(*cls*)

Add a class to the mapper

Parameters

cls
[type] The class to add to the mapper

Raises

ClassRedefinitionAttempt
Thrown when *add_class* is called on a class when a class with the same name has already been added to the mapper

load_module(*module_name*)

Loads the module.

lookup_class(*cname*)

Gets the class corresponding to a fully-qualified class name

resolve_class(*uri*, *context*)

Look up the Python class for the given URI recovered from the given *Context*

Parameters

uri
[rdflib.term.URIRef] The URI to look up

context
[*Context*] The context the URI was found in. May affect which Python class is returned.

property class_registry_context

Context where class registry entries are stored

property class_registry_context_list

Context where class registry entries are retrieved from if *class_registry_context* doesn't contain an appropriate entry

owmeta_core.mapper.**CLASS_REGISTRY_CONTEXT_KEY = 'class_registry_context_id'**

class_registry_context_id

Configuration file key for the URI of the class registry RDF graph context.

The class registry context holds the mappings between RDF types and Python classes for a project or bundle.

owmeta_core.mapper.**CLASS_REGISTRY_CONTEXT_LIST_KEY = 'class_registry_context_list'**

class_registry_context_list

Configuration file key for the list of class registry contexts

If it is specified, then *class_registry_context_id* should be searched first for class registry entries. The class registry list may be built automatically or not defined at all depending on who makes the Configuration, but if it is specified with this property, then it should be respected.

owmeta_core.property_mixins module

class owmeta_core.property_mixins.**UnionPropertyMixin**(*resolver*, ***kwargs*)

Bases: object

A Property that can handle either DataObjects or basic types

Parameters

resolver
[RDFTypeResolver] Resolves RDF identifiers into objects returned from get()

owmeta_core.property_value module

class owmeta_core.property_value.**PropertyValue**(*value*)

Bases: object

Holds a literal value for a property

owmeta_core.quantity module

owmeta_core.ranged_objects module

class owmeta_core.ranged_objects.**InRange**(*minval=None*, *maxval=None*, ***kwargs*)

Bases: object

A range between values

owmeta_core.rdf_query_modifiers module

class owmeta_core.rdf_query_modifiers.**ContainerMembershipIsMemberTQLayer**(*nxt=None*)

Bases: *TQLayer*

Adds a triple into the results for rdfs:subPropertyOf(rdfs:member) relationships for all known ContainerMembershipProperty instances

Parameters

nxt
[*TQLayer* or rdflib.graph.Graph] The "next" or "lower" layer that this layer modifies

class owmeta_core.rdf_query_modifiers.**RangeTQLayer**(*nxt=None*)

Bases: *TQLayer*

A layer that understands ranges in the object position of a triple.

If the next layer has the supports_range_queries attribute set to True, then the range is passed down as-is

Parameters

nxt
[*TQLayer* or rdflib.graph.Graph] The "next" or "lower" layer that this layer modifies

class owmeta_core.rdf_query_modifiers.**TQLayer**(*nxt=None*)

Bases: object

Triple Query Layer. Wraps a graph or another *TQLayer* to do something to the triples and triples_choices query or the result of the query.

Parameters

nxt
[*TQLayer* or rdflib.graph.Graph] The "next" or "lower" layer that this layer modifies

class owmeta_core.rdf_query_modifiers.**TerminalTQLayer**

Bases: object

A TQLayer that has no "next". May be useful to create a layer that stands in place of a Graph.

owmeta_core.rdf_query_modifiers.**rdfs_subclassof_zom**(*triple*)

Argument to ZeroOrMoreTQLayer. Adds sub-classes to triple queries for an rdf:type

owmeta_core.rdf_query_modifiers.**rdfs_subclassof_zom_creator**(*target_type*)

Creates a function used by ZeroOrMoreTQLayer to determine if a query needs to be augmented to retrieve sub-classes of a *given* RDF type

owmeta_core.rdf_query_modifiers.**rdfs_subpropertyof_zom**(*super_property*)

Argument to ZeroOrMoreTQLayer. Adds sub-properties of the given property to triple queries

owmeta_core.rdf_query_util module

exception owmeta_core.rdf_query_util.**MissingRDFTypeException**

Bases: Exception

Raised when we were looking for an RDF type couldn't find one

owmeta_core.rdf_query_util.**get_most_specific_rdf_type**(*graph*, *types*, *base=None*)

Find the RDF type that isn't a sub-class of any other, constrained to be a sub-class of base if that is provided.

Parameters

graph
[rdflib.graph.Graph] The graph to query rdfs:subClassOf relationships

types
[list of rdflib.term.URIRef] The types to query

base
[rdflib.term.URIRef] The "base" type

See also:

RDFTypeResolver

owmeta_core.rdf_query_util.**load**(*graph*, *start*, *target_type*, **args*)

Loads a set of objects based on the graph starting from start

Parameters

graph
[rdflib.graph.Graph] The graph to query from

start
[*graph_object.GraphObject*] The graph object to start the query from

target_type
[rdflib.term.URIRef] URI of the target type. Any result will be a sub-class of this type

owmeta_core.rdf_query_util.**load_base**(*graph*, *idents*, *target_type*, *context*, *resolver*)

Loads a set of objects from an RDF graph given their identifiers

Parameters

graph
[rdflib.graph.Graph] The graph to query from

idents
[list of rdflib.term.URIRef] A list of identifiers to convert into objects

target_type
[rdflib.term.URIRef] URI of the target type. Any result will be a sub-class of this type

context
[object] Limits the scope of the query to statements within or entailed by this context. Notionally, it's a owmeta_core.context.Context instance

resolver
[*rdf_type_resolver.RDFTypeResolver*] Handles some of the mappings

owmeta_core.rdf_query_util.**load_terms**(*graph*, *start*, *target_type*)

Loads a set of terms based on the object graph starting from `start`

Parameters

graph
[rdflib.graph.Graph] The graph to query from

start
[*graph_object.GraphObject*] The graph object to start the query from

target_type
[rdflib.term.URIRef] URI of the target type. Any result will be a sub-class of this type

owmeta_core.rdf_query_util.**oid**(*identifier_or_rdf_type*, *rdf_type*, *context*, *base_type=None*)

Create an object from its rdf type

Parameters

identifier_or_rdf_type
[rdflib.term.URIRef] If `rdf_type` is provided, then this value is used as the identifier for the newly created object. Otherwise, this value will be the `rdf_type` of the object used to determine the Python type and the object's identifier will be randomly generated.

rdf_type
[rdflib.term.URIRef] If provided, this will be the `rdf_type` of the newly created object.

context
[Context, optional] The context to resolve a class from

base_type
[type] The base type

Returns

The newly created object

owmeta_core.rdf_type_resolver module

class owmeta_core.rdf_type_resolver.**RDFTypeResolver**(*default_type*, *type_resolver*, *id2object_translator*, *deserializer*)

Bases: object

Handles mapping between RDF graphs and Python types

Parameters

default_type
[str, rdflib.term.URIRef] If no type is retrieved from the graph, this will be the type selected

type_resolver
[callable()][(rdflib.graph.Graph, [rdflib.term.URIRef], rdflib.term.URIRef or None) -> rdflib.term.URIRef] This callable (e.g., function) receives a graph, all the types found for an identifier, and the "base" type sought, which constrains the result to be a sub-type of the base, and returns a single identifier for a type that id2object_translator can translate into an object

id2object_translator
[callable()][(rdflib.term.URIRef, rdflib.term.URIRef, *owmeta_core.context.Context*) -> object] This callable (e.g., function) receives an identifier for an object and an identifier for the object's type and returns an object corresponding to the identifier and type

deserializer
[callable()][(rdflib.term.Literal) -> object] This callable (e.g., function) receives a literal and turns it into an object

owmeta_core.rdf_utils module

class owmeta_core.rdf_utils.**BatchAddGraph**(*graph*, *batchsize=1000*, *_parent=None*, **args*, ***kwargs*)

Bases: object

Wrapper around graph that turns calls to 'add' into calls to 'addN'

owmeta_core.rdf_utils.**transitive_lookup**(*graph*, *start*, *predicate*, *context=None*, *direction='down'*, *seen=None*)

Do a transitive lookup over an rdflib.graph.Graph or rdflib.store.Store

In other words, finds all resources which relate to start through zero or more predicate relationships. start itself will be included in the return value.

Loops in the input graph will not cause non-termination.

Parameters

graph
[rdflib.graph.Graph or rdflib.store.Store] The graph to query

start
[rdflib.term.Identifier] The resource in the graph to start from

predicate
[rdflib.term.URIRef] The predicate relating terms in the closure

context
[rdflib.graph.Graph or rdflib.term.URIRef] The context in which the query should run. Optional

direction
[*DOWN* or *UP*] The direction in which to traverse

seen
[set of rdflib.term.Identifier] A set of terms which have already been "seen" by the algorithm. Useful for repeated calls to *transitive_lookup*. Note: if the `start` is in `seen`, queries from `start` will still be done, but any items in the result of *those* queries will not be queried for if in `seen`. Optional

Returns

set of rdflib.term.Identifier
resources in the transitive closure of `predicate` from `start`

owmeta_core.rdf_utils.**transitive_subjects**(*graph*, *start*, *predicate*, *context=None*, *direction='down'*, *seen=None*)

Alias to *transitive_lookup*

owmeta_core.rdf_utils.**DOWN = 'down'**

Subject to Object direction for traversal across triples.

owmeta_core.rdf_utils.**UP = 'up'**

Object to Subject direction for traversal across triples.

owmeta_core.requests_sessions module

A collection of functions that produce requests.Session objects.

A few methods request a "session provider". The functions in here are providers of that kind

owmeta_core.requests_sessions.**caching**()

Provides a requests.Session that puts cached responses in `.owmeta_http_cache`

In absence of explict cache-control headers, uses a heuristic of caching cacheable responses for up to a day.

owmeta_core.statement module

owmeta_core.text_util module

owmeta_core.utils module

Common utilities for translation, massaging data, etc., that don't fit elsewhere in owmeta_core

owmeta_core.utils.**grouper**(*iterable*, *n*, *fillvalue=None*)

Collect data into fixed-length chunks or blocks

owmeta_core.utils.**retrieve_provider**(*provider_path*)

Look up a "provider" specified by a string.

Path to an object that provides something. The format is similar to that for setuptools entry points: `path.to.module:path.to.provider.callable`. Notably, there's no name and "extras" are not supported.

Parameters

provider_path
[str] The path to the provider

Returns

object
The provider

Raises

ValueError
The `provider_path` format doesn't match the expected pattern

AttributeError
Some element in the path is missing

owmeta_core.variable module

class owmeta_core.variable.**Variable**(*name*, ***kwargs*)

Bases: *GraphObject*

A graph object representing a variable. Typically used in property values

CHAPTER

TWO

FOR USERS

2.1 Making data objects

To make a new object type, you just need to make a subclass of *BaseDataObject* with the appropriate properties.

Say, for example, that I want to record some information about drug reactions in dogs. I make `Drug`, `Experiment`, and `Dog` classes to describe drug reactions:

```
>>> from owmeta_core.dataobject import (BaseDataObject,
...                                     DatatypeProperty,
...                                     ObjectProperty,
...                                     Alias)
>>> from owmeta_core.context import Context
>>> from owmeta_core.mapper import Mapper

>>> module_context = 'http://example.com/animals'

>>> class Dog(BaseDataObject):
...     breed = DatatypeProperty()

>>> class Drug(BaseDataObject):
...     name = DatatypeProperty()
...     drug_name = Alias(name)
...     key_property = 'name'
...     direct_key = True

>>> class Experiment(BaseDataObject):
...     drug = ObjectProperty(value_type=Drug)
...     subject = ObjectProperty(value_type=Dog)
...     route_of_entry = DatatypeProperty()
...     reaction = DatatypeProperty()

# Do some accounting stuff to register the classes. Usually happens behind
# the scenes.
>>> m = Mapper()
>>> m.process_classes(Drug, Experiment, Dog)
```

So, we have created I can then make a Drug object for moon rocks and describe an experiment by Aperture Labs:

```
>>> ctx = Context('http://example.org/experiments', mapper=m)
>>> d = ctx(Drug)(name='moon rocks')
```

(continues on next page)

(continued from previous page)

```
>>> e = ctx(Experiment)(key='experiment001')
>>> w = ctx(Dog)(breed='Affenpinscher')
>>> e.subject(w)
owmeta_core.statement.Statement(...Context(.../experiments"))

>>> e.drug(d)
owmeta_core.statement.Statement(...)

>>> e.route_of_entry('ingestion')
owmeta_core.statement.Statement(...)

>>> e.reaction('no reaction')
owmeta_core.statement.Statement(...)
```

and save those statements:

```
>>> ctx.save()
```

For simple objects, this is all we have to do.

You can also add properties to an object after it has been created by calling either ObjectProperty or DatatypeProperty on the class:

```
>>> d = ctx(Drug)(name='moon rocks')
>>> Drug.DatatypeProperty('granularity', owner=d)
__main__.Drug_granularity(owner=Drug(ident=rdflib.term.URIRef('http://data.openworm.org/
↪Drug#moon%20rocks')))

>>> d.granularity('ground up')
owmeta_core.statement.Statement(...Context(.../experiments"))

>>> do = Drug()
```

Properties added in this fashion will not propagate to any other objects:

```
>>> do.granularity
Traceback (most recent call last):
    ...
AttributeError: 'Drug' object has no attribute 'granularity'
```

They will, however, be saved along with the object they are attached to.

2.2 Working with contexts

2.2.1 Background

Contexts were introduced to owmeta-core as a generic tool for grouping statements. We need to group statements to make statements about statements like “Who made these statements?” or “When were these statements made?”. That’s the main usage. Beyond that, we need a way to share statements. Contexts have identifiers by which we can naturally refer to contexts from other contexts.

owmeta-core needs a way to represent contexts with the existing statement form. Other alternatives were considered, such as using Python’s context managers, but I (Mark) also wanted a way to put statements in a context that could also

be carried with the subject of the statement. Using the wrapt package's proxies allows to achieve this while keeping the interface of the wrapped object the same, which is useful since it doesn't require a user of the object to know anything about contexts unless they need to change the context of a statement.

The remainder of this page will go into doing some useful things with contexts.

2.2.2 Classes and contexts

owmeta-core can load classes as well as instances from an RDF graph. The packages which define the classes must already be installed in the Python library path, and a few statements need to be in the graph you are loading from or in a graph imported (transitively) by that graph. The statements you need are these

```
:a_class_desc <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://openworm.org/
↪entities/PythonClassDescription> .
:a_class_desc <http://openworm.org/entities/ClassDescription/module> :a_module .
:a_class_desc <http://openworm.org/entities/PythonClassDescription/name> "AClassName" .

:a_module <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://openworm.org/
↪entities/PythonModule> .
:a_module <http://openworm.org/entities/PythonModule/name> "APackage.and.module.name" .
```

where `:a_class_desc` and `:a_module` are placeholders for objects which will typically be created by owmeta-core on the user's behalf, and `AClassName` is the name of the class available at the top-level of the module `APackage.and.module.name`. These statements will be created in memory by owmeta-core when a module defining a *DataObject*-derived class is first processed by a *Mapper* which will happen after the module is imported.

2.3 `owm` Command Line

The `owm` command line provides a high-level interface for working with owmeta-core-managed data. The central object which `owm` works on is the owmeta-core project, which contains the triple store – a set of files in a binary format. The sub-commands act on important files inside the project or with entities in the database.

To get usage information:

```
owm --help
```

To clone a project:

```
owm clone $database_url
```

This will clone a project into `.owm` in your current working directory. After a successful clone, a binary database usable as a owmeta store will have been created from the serialized graphs (i.e., sets of RDF triples) in the project.

To save changes made to the database, run the `commit` sub-command like this:

```
owm commit -m "Adding records from January-March"
```

To recreate the database from serialized graphs, run the `regendb` sub-command:

```
owm regendb
```

Be careful with `regendb` as it will delete anything you have added to binary database beyond what's in the serialized graphs.

To make a new project:

```
owm init
```

This will create a project in `.owm` in your current working directory.

2.4 Software Versioning

The owmeta-core library follows the semanitc versioning scheme. For the sake of versioning, the software interface consists of:

1. The `owm` command line interface
2. The underlying *owmeta_core.command.OWM* class underlying that CLI
3. All "public" definitions (i.e., those whose names do not begin with '_') in the *owmeta_core* package, sub-packages, and sub-modules
4. The format of RDF data generated by *owmeta_core.dataobject.DataObject* and the subclasses thereof defined in the *owmeta_core* package, sub-packages, and sub-modules
5. The API documentation for the *owmeta_core* package, sub-packages, and sub-modules

In addition, any changes to the packages released on PyPI mandates at least a patch version increment.

For Git, our software version control system, software releases will be represented as tags in the form `v$semantic_version` with all components of the semantic version represented.

2.4.1 Documentation versioning

The documentation will have a distinct version number from the software. The version numbers for the documentation must change at least as often as the software versioning since the relationship of the documentation to the software necessarily changes. However, changes *only* to the non-API documentation will not be a cause for a change to any of the components of the software version number. For documentation releases which coincide with software releases, the documentation version number will simply be the software version number. Any subsequent change to documentation between software releases will compel an increase in the documentation version number by one. The documentation version number for such documentation releases will be represented as `${software_version}+docs${documentation_increment}`.

2.4.2 Mapped Class Versioning

Versioning for mapped classes has special considerations related to how Python classes map to RDF types. RDF types map to a specific class, module, and package version through the "class registry". This class registry is contained within a *bundle* and each bundle is free to have its own registry. The registry, moreover, can be replaced by another registry by the user of the bundle.

We want data created with later versions of a mapped class to be compatible with earlier versions of that class so that we can use pre-existing analysis and transformations (e.g., with a *DataTranslator*). This flexibility allows for pre-existing processes to keep working without change even as the upstream moves on. On the other hand, newer versions of a software package should still have access to the data created with older versions of the corresponding Python classes so that users of that package are not forced to translate or abandon the old data.

The two-way compatibility described above is appropriate in the context of the "open world assumption": the relationships an RDF type participates in are described by the Python class, but that description may be incomplete. We may make the description of an RDF type more complete by adding properties to the Python class or constraining existing properties. When we add properties, however, we should create a new Python class rather than modifying the existing

one: this allows for querying for data created with the earlier version of the Python class while also being able to create instances of the new class. The new class should not, however, have the same RDF type as the old one since the code for resolving types from the class registry only supports mapping to one Python type from any given RDF type[1]. The recommended way to handle this is to include a version number in the URI for the RDF type and, when making the new type, to increment the version number for the new URI. The new type should be declared as a sub-class of the old type, and owmeta-core will add the appropriate sub-class relationships so that querying for the old type will return instances of the new type as well. This split also means that while I use the new software package, I can utilize the data generated with the old Python class without needing to have the old Python package because the new package retains the old class.

2.4.3 Release Notes

Release notes are organized, generally into three sections. 'Features and Enhancements' are changes to the external interface of owmeta-core where there wasn't anything that fulfilled the use case previously (features) or where the previous behavior was sub-optimal or just different (enhancements), but not wrong per se. The second section, 'Fixes', contains corrections to previous behavior. The third section 'Internal/Misc. Changes' contains changes that either don't really change owmeta-core itself, like changes in to project metadata, documentation changes, or changes to build automation. Other sections may be added, like 'Known Issues', which should be self-explanatory when used.

Notes:

2.5 Python Release Compatibility

All Python releases will be supported until they reach their official end-of-life, typically reported as "Release Schedule" PEPs (search "release schedule" on the PEP index) Thereafter, any regressions due to dependencies of owmeta-core dropping support for an EOL Python version, or due to a change in owmeta-core making use of a feature in a still-supported Python release will only be fixed for the sake of OpenWorm projects when requested by an issue on our tracker or for other projects when a compelling case can be made.

This policy is intended to provide support to most well-maintained projects which depend on owmeta-core while not overburdening developers.

2.6 BitTorrent client for P2P filesharing

1. **Download** desired contents:

- A *LocalFileDataSource* created and stored within the local graph store contains a torrent_file_name `Informational`. This refers to the torrent containing the location of the desired contents on the BitTorrent. A `torrent` is used to locate files on the File System [BEP 3]. A *DataSource* defines attributes about the contents that it represents.

  Module `t` describes the `DataSource` attributes:

[1] One alternative to this is returning, for each RDF instance of a type, `N` Python instances for `N` Python classes in the registry mapped to the RDF type.

```
def owm_data(ns):
    ns.context.add_import(ConnectomeCSVDataSource.definition_context)
    ns.context(ConnectomeCSVDataSource)(
    key = '2000_connections',
    csv_file_name = 'connectome.csv',
    torrent_file_name = 'd9da5ce947c6f1c127dfcdc2ede63320.torrent'
 )
```

The `DataSource` can be created and stored on the local graph with:

```
$ owm save t
```

The `DataSource` identifier can be used to see contents stored in the local graph with:

```
$ owm source show ConnectomeCSVDataSource:2000_connections
```

ConnectomeCSVDataSource
CSV file name: 'connectome.csv'

File name: 'connectome.csv'

Torrent file name: 'd9da5ce947c6f1c127dfcdc2ede63320.torrent'

- The BitTorrentDataSourceDirLoader class inherits from the *DataSourceDirLoader* and overrides its `load ()` method. Google Drive stores the `torrents` uploaded by other researchers. `load()` fetches the `torrent` refered to in `torrent_file_name` of the `DataSource`, performs *DataTranslator* from one form to another and then adds the `torrent` to the BitTorrent Client for downloading its contents.

This `BitTorrent Client` is available on PyPI and is included in the owmeta_core setup.

To install separately:

```
$ pip install torrent-client
```

For reference, use the torrent-client repository and its usage information with:

```
$ torrent_cli.py -h
```

The `DataSourceDirLoader` attribute - `base_directory`, which is set in the `BitTorrentDataSourceDirLoader` constructor is where both the `torrent` and its contents are downloaded:

```
content = BitTorrentDataSourceDirLoader("./")
```

- Within the .owm directory we have the credentials.json and token.pickle these are for authentication of the Google Drive. For the purpose of access control the `client_secret` required by `credentials.json` will only be shared by owmeta maintainers.

- The `torrent` file name is the MD5 message digest of its contents. If the hash of the downloaded contents is the same as its `torrent` name the data is unaltered.

Data-Integrity is to be checked after 100% download completion:

```
$ python3 integrity.py 'd9da5ce947c6f1c127dfcdc2ede63320.torrent' 'Merged_
↪Nuclei_Stained_Worm.zip'
```

2. **Upload** your contents:

- On an AWS EC2 instance is running a Nginx WSGI and a Flask Server to accept .zip content file uploads. Visit this Elastic IP address [13.235.204.78] to upload your files by browsing through your filesystem and then clicking the `Submit Query button`.
- This will create a `torrent` and `seed` your contents in parts, to other peers on the BitTorrent network. Content can then be downloaded as described above.

2.7 Querying for data objects

2.7.1 DataObject query form

Sub-classes of *DataObject* have a `query` attribute that provides a modified form of the class which is fit for creating instances used in queries. The query form may do other things later, but, principally, it overrides identifier generation based on attributes (see *IdMixin*).

For example, to query for a `Neuron` object with the name "AVAL" you would instantiate the `Neuron` like this:

```
>>> Neuron.query(name='AVAL')
```

Although it is possible to include instances without the query form, it is generally preferred to the basic form since later versions of a class may change how they generate identifiers while keeping property URIs and RDF types the same (or declaring new ones as sub-properties or sub-classes). Use of the query form is also recommended when a class generates identifiers based on some number of properties, but a subclass doesn't use the superclass identifier scheme (`Cell` and `Neuron` are an example). The query form allows to query for instances of the superclass for subclass instances.

2.8 Transactions

Transactions in owmeta-core are managed through the transaction library. The default RDF store is transactional. You can execute code within a transaction using a transaction manager. owmeta-core connections come with a transaction manager which you can access via the `transaction_manager` attribute. It's recommended to use a context manager to start and commit transactions like this:

```
>>> from rdflib.term import URIRef
>>> from owmeta_core import connect
>>> with connect() as conn, conn.transaction_manager:
...     conn.rdf.add((
...         URIRef('http://example.org/bob'),
...         URIRef('http://example.org/likes'),
...         URIRef('http://example.org/alice')))
```

Because this is a common pattern, there's a *transaction()* method that does something equivalent which is provided for convenience:

```
>>> with connect().transaction() as conn:
...     conn.rdf.add((
...         URIRef('http://example.org/bob'),
...         URIRef('http://example.org/likes'),
...         URIRef('http://example.org/alice')))
```

Similar usage is possible with project connections through the high-level *OWM* interface:

```
>>> from owmeta_core.command import OWM
>>> owm = OWM(non_interactive=True)
>>> owm.init(default_context_id=URIRef("http://example.org/context"))
Initialized owmeta-core project at .../.owm

>>> with owm.connect().transaction() as conn:
...     conn.rdf.add((
...         URIRef('http://example.org/bob'),
...         URIRef('http://example.org/likes'),
...         URIRef('http://example.org/alice')))
```

However, the methods of *OWM* and its "sub-commands" will typically manage the transactions themselves, so it wouldn't be necessary to start a transaction explicitly before calling these methods–in fact, doing so would typically cause an exception. For example, in this code:

```
>>> owm.say('http://example.org/bob',
...         'http://example.org/likes',
...         'http://example.org/eve')
```

we don't have to declare a transaction since the *say* method handles that for us.

For read-only operations, it is not strictly necessary to read from the RDF store within the context of a transaction, but it is recommended if you're in a multithreaded context to avoid getting an inconsistent picture of the data if there's an update part way through your operation.

CHAPTER

THREE

FOR DEVELOPERS

3.1 Testing in owmeta-core

3.1.1 Preparing for tests

owmeta_core should be installed like:

```
pip install -e .
```

3.1.2 Running tests

Tests should be run via setup.py like:

```
python setup.py test
```

you can pass options to `pytest` like so:

```
python setup.py test --addopts '-k CommandTest'
```

3.1.3 Writing tests

Tests are written using Python's unittest. In general, a collection of closely related tests should be in one file. For selecting different classes of tests, tests can also be tagged using pytest marks like:

```
@pytest.mark.tag
class TestClass(unittest.TestCase):
    ...
```

Currently, marks are used to distinguish between unit-level tests and others which have the `inttest` mark

3.1.4 Deselecting tests

Tests can be deselected by adding a pytest "marker" to the test function, class, or module and then adding `-m 'not <your_marker>'` to the pytest command line. Marking tests to be explicitly deselected is preferred to skipping tests since skipped tests tend to break silently, especially with conditional skips such as with with `pytest.mark.skipif`. A set of markers is, however, deselected by default in the `addopts` line in our `pytest.ini` file. Deselected marks are added on a case-by-case basis and will always run on CI.

3.2 Writing documentation

Documentation for owmeta-core is housed in two locations:

1. In the top-level project directory as `INSTALL.md` and `README.md`.
2. As a Sphinx project under the `docs` directory

By way of example, to add a page about useful facts concerning *C. elegans* to the documentation, include an entry in the list under `toctree` in `docs/index.rst` like:

```
worm-facts
```

and create the file `worm-facts.rst` under the `docs` directory and add a line:

```
.. _worm-facts:
```

to the top of your file, remembering to leave an empty line before adding all of your wonderful worm facts.

You can get a preview of what your documentation will look like when it is published by running `sphinx-build` on the docs directory. To get the `sphinx-build` command, install the documentation requirements with:

```
pip install -r doc-requirements.txt
```

Then, you can run `sphinx-build` like this:

```
sphinx-build -w sphinx-errors docs <build_destination>
```

You can also invoke the command with default arguments (i.e., with output to `build/sphinx` using `setup.py`:

```
python setup.py build_sphinx
```

The docs will be compiled to html which you can view by pointing your web browser at `<build_destination>/index.html`. The documentation will be rendered using the same theme as is used on the readthedocs.org site.

3.2.1 API Documentation

API documentation is generated by the Sphinx autodoc and apidoc extensions. The numpydoc format should be easy to pick up on, but a reference is available here. Just add a docstring to your function/class/method and your class should appear among the other documented classes. Note, however, that "special" methods like `__call__` will not show up by default – if they need to be documented for a given class, add a declaration like this to the class documentation:

```
class SpecialMethodDocExample:
    '''
    Example class doc

```

(continues on next page)

(continued from previous page)

```
    .. automethod:: __call__
    '''

    def __call__(self):
        '''
        Hey, I'm in the API documentation!
        '''
```

3.2.2 Substitutions

Project-wide substitutions can be (conservatively!) added to allow for easily changing a value over all of the documentation. Currently defined substitutions can be found in `conf.py` in the `rst_epilog` setting. More about substitutions

3.2.3 Conventions

If you'd like to add a convention, list it here and start using it. It can be reviewed as part of a pull request.

1. Narrative text should be wrapped at 80 characters.
2. Long links should be extracted from narrative text. Use your judgement on what 'long' is, but if it causes the line width to stray beyond 80 characters that's a good indication.

3.3 owmeta-core coding standards

Pull requests are *required* to follow the PEP-8 Guidelines for contributions of Python code to owmeta-core, with some exceptions noted below. Compliance can be checked with the `pep8` tool and these command line arguments:

```
--max-line-length=120 --ignore=E261,E266,E265,E402,E121,E123,E126,E226,E24,E704,E128
```

Refer to the pep8 documentation for the meanings of these error codes.

Lines of code should only be wrapped before 120 chars for readability. Comments and string literals, including docstrings, can be wrapped to a shorter length.

Some violations can be corrected with `autopep8`.

3.4 Design documents

These comprise the core design artifacts for owmeta.

3.4.1 Project Bundles

A project bundle is composed of:

- a universally unique identifier,
- a version number,
- a collection of contexts,
- a distinguished "imports" context describing relationships between contexts, both those in the bundle, and between contexts in the bundle and in dependencies,

plus several optional components:

- a human-friendly name,
- a description of the bundle's contents,
- a collection of files,
- a listing of dependencies on other bundles,
- a set of mappings between project-scoped identifiers and universal context identifiers.

They solve the problem of contexts containing different statements having the same identifier for different purposes.

There are several ways we can get different contexts with the same identifier:

- through revisions of a context over time,
- by distinct groups using the same context identifier,
- or by contexts being distributed with different variants (e.g., a full and an abridged version).

In solving this problem of context ID aliasing, bundles also helps solve the problem of having contexts with inconsistent statements in the same project by providing a division within a project, between groups of contexts that aren't necessarily related.

Dependencies

A bundle can declare other bundles upon which it depends, by listing those other bundles identifiers and version numbers. In addition, a bundle can declare contexts and files within the dependency that should be included or excluded. More interestingly, a dependency specification may declare that contexts declared within the dependency be renamed according to a number of rewrite rules. This is to allow for using bundles with conflicting Context Identifiers.

Certain problems come up when dealing with contexts across different bundles. This rewriting allows to keep separate the contexts in one bundle from another and to prevent contexts with the same ID from conflicting with one another just because they're brought in by a transitive dependency.

An example

This example describes a likely naming conflict that can arise in context naming between bundles.

Bundles , , and . With dependencies like so:

```
->  ->
```

where both and contain a context with ID `c`. The dependency resolution system will find the `c` context in and if there is no remapping that removes the conflict, either in or in , then the system will deliver a message indicating that the context needs to be deconflicted and in which bundle each of the conflicting declarations is. At this point, the

maintainer of the package can make the change to omit `c` from , omit it from , rename `c` in , or rename it in . One special case, where 's `c` and 's `c` are identical, permits an automatic resolution; nonetheless, the system emits a warning in this case, with the option to fail similarly to the case where the contexts are distinct.

Core bundles

The "core" bundle contains (or depends on) metadata of all of the core classes in owmeta which are needed to make owmeta features work. The core bundle is generated automatically for whichever version of owmeta is in use and a reference to it is added automatically when a bundle is installed. A given bundle may, however, explicitly use a specific version of the core bundle.

Relationships

Where not specified, the subject of a relationship can participate in the relationship exactly once. For example, "A Dog has a Human", means "A Dog has one and only one Human".

- A Project can have zero or more Bundles
- A Bundle can belong to only one Project
- A Context Identifier is associated with one or more Content-Based Identifiers
- A Content-Based Identifier has a Hash
- A Content-Based Identifier has an RDF Serialization Format
- A Hash can appear in zero or more Content-Based Identifiers
- A Hash has an Algorithm ID and a Message Digest

Types

Below is a description in terms of lower-level types of some higher-level types referenced above.

- A Message Digest is a Base-64 encoding of a string of bytes
- An Algorithm ID is a string that identifies an algorithm. Valid strings will be determined by any applications reading or writing the hashes, but in general will come from the set of constructors of Python's `hashlib` module.
- An RDF Serialization Format is a string naming the format of a canonical RDF graph serialization. Supported format strings:

  "nt"
  : N-Triples

3.4.2 Project Distribution

Projects are distributed as *bundle* archives, also referred to as `dists` (short for distributions) in the documentation and commands. The layout of files in a dist is largely the same as the format of a `.owm` directory on initial clone. In other words the bundle contains a set of serialized graphs, an index of those graphs, an optional set of non-RDF data that accompanies data sources stored amongst the graphs, and a configuration file which serves as a working owmeta configuration file and a place for metadata about the bundle. The archive file format can be allowed to vary, between producers and consumers of dists, but at least the `tar.gz` format should be supported by general-purpose clients.

3.4.3 Data Packaging Lifecycle

The package lifecycle encompasses the creation of data, packaging of said data, and uploading to shared resources. The data packaging lifecycle draws from the Maven build lifecycle in the separation of local actions (e.g., `compile`, `stage`, `install` phases) from remote interactions (the `deploy` phase). To explain why we have these distinct phases, we should step back and look at what needs to happen when we share data.

In owmeta-core, we may be changing remote resources outside of the owmeta-core system. We also want to support local use and staging of data because it is expected that there is a lengthy period of data collection/generation, analysis, curation, and editing which precedes the publication of any data set. Having separate phases allows us to support a wider range of use-cases with owmeta-core in this local "staging" period.

To make the above more concrete, the prototypical example for us is around *`LocalFileDataSource`*, which wants to make the files described in the data source available for download. Typically, the local path to the file isn't useful outside of the machine. Also, except for files only a few tens of bytes in size, it isn't feasible to store the file contents in the same database as the metadata. We, still want to support metadata about these files and to avoid the necessity of making *n* different *`DataSource`* sub-classes for *n* different ways of getting a file. What we do is define a "deploy" phase that takes every *`LocalFileDataSource`* and "deploys" the files by uploading them to one or more remote stores or, in the case of a peer-to-peer solution, by publishing information about the file to a tracker or distributed hash table.

Packaging proceeds in phases to serve as an organizational structure for data producers, software developers, management, and information technology personnel. Compared with a more free-form build strategy like using an amalgam of shell scripts and disconnected commands, or even rule-based execution (e.g., GNU make), phases organize the otherwise implicit process by which the local database gets made available to other people. This explicitness is very useful since, when different people can take different roles in creating the configuration for each phase, having named phases where things happen aids in discussion, process development, and review. For instance, junior lab technicians may be responsible for creating or maintaining packaging with guidance from senior technicians or principal investigators. IT personnel may be interested in all phases since they all deal with the computing resources they manage, but they may focus on the phases that affect "remote" resources since those resources may, in fact, be managed within the same organization and require additional effort on the back-end to prepare those remote resources (e.g., generating access credentials).

The remainder of this document will describe the default lifecycle and what takes place within each phase.

Default Lifecycle

The default lifecycle takes a *bundle*, including the contents of a owmeta-core triple store, creates one or more packages from that, stages the packages for ongoing development, and, finally, deploys packages to shared resources so that colleagues and other interested parties can access them. Each phase is associated with a sub-command in *owm*.

Install

Preparation for distribution.

When we're generating data, our workspace is not necessarily in the right state for distribution. We may have created temporary files and notes to ourselves, or we may have generated data in trial runs, intentionally or mistakenly, which do not reflect our formal experimental conditions. In the install phase, we bring together just the data which we wish to distribute for a given bundle and place it in the local bundle cache. This includes gathering hashes for files that belong to the bundle and serializing named graphs. Once these data are installed, they should be immutable – in other words, they should not change any more. Consequently, the install phase is the appropriate time for creating summary statistics, signatures, and content-based identifiers.

Much of the data which is created in a research lab is append-only: observations are logged and timestamped either by a human or by a machine in the moment they happen, and, if recorded properly, such logs are rarely edited, or, if there is an amendment, it also is logged as such, with the original record preserved. As long as this append-only property

is preserved, we only need to designate the range of such time-stamped records which belong in a bundle to have the desired immutability for a locally installed bundle without requiring a file copy operation. Of course, if the source data is expected to be changed, then we would want either a copy-on-write mechanism (at the file system level) or to copy the files. Regardless, file hashes and/or signatures created during the install phase would be available for guarding against accidental changes.

owmeta-core will create a local repository to house installed packages. The repository stores the relationship between the human-friendly name for the package (serving a purpose similar to Maven's group-artifact-version coordinates) and the set of serialized RDF graphs in the package. Given that the repository is meant to serve a user across projects, the repository will be stored in the "user directory", if one can be found on the system.[1]

Deploy

Creation of configuration for upload/download. Sharing packages.

In the "deploy" phase, we publish our data to "remotes". A "remote" may be a repository or, in the case of a peer-to-peer file sharing system, a file index or DHT. Above, we referred to non-RDF data files on the local file system – during the deploy phase, these files are actually published and accession information (e.g., a database record identifier) for those files is generated and returned to the system where the deployment was initiated. This assumes a fully automated process for publication of files: If, instead, the publication platform requires some manual interaction, that must be done outside of owmeta-core and then the accession information would be provided with the deploy command.

3.4.4 Publishing DataSources

`DataSource` is a subclass of `DataObject` with a few features to make describing data files (CSV, HDF5, Excel) a bit more consistent and to make recovering those files, and information about them, more reliable. In order to have that reliability we have to take some extra measures when publishing a `DataSource`. In particular, we must publish local files referred to by the `DataSource` and relativize those references. This file publication happens in the *"deploy" phase* of the data packaging lifecycle. Before that, however, a description of what files need to be published is generated in the *"stage" phase*. In the "stage" phase, the `DataSources` with files needing publication are queried for in the configured triple store, and the "staging manager", the component responsible for coordinating the "stage" phase identifies file references that refer to the same files and directories.

[1] This will be the user directory as determined by `os.path.expanduser()`

CHAPTER

FOUR

OWMETA_CORE EXAMPLES

4.1 alt_objects.py

CHAPTER

FIVE

ISSUES

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z