dtoolcore Documentation

Release 3.18.2

Tjelvar Olsson

Mar 20, 2022

Contents

CHAPTER 1

Content

1.1 Manage scientific data sets

- Documentation: http://dtoolcore.readthedocs.io
- GitHub: https://github.com/jic-dtool/dtoolcore
- PyPI: https://pypi.python.org/pypi/dtoolcore
- Free software: MIT License

1.1.1 Features

- Core API for adding different types of metadata to files on disk
- Automatic generation of structural metadata
- Programmatic discovery and access of items in a dataset
- Structural metadata includes hash, size and modification time for subsequent integrity checks
- Ability to annotate individual files with arbitrary metadata
- Metadata stored on disk as plain text files, i.e. disk datasets generated using this API can be accessed without special tools
- Ability to create plugins for custom storage solutions
- Plugins for iRODS and Microsoft Azure storage backends available

- Cross-platform: Linux, Mac and Windows are all supported
- Works with Python 2.7, 3.5 and 3.6
- No external dependencies

1.1.2 Overview

The dtoolcore project provides a Python API for managing (scientific) data. It allows researchers to:

- Package data and metadata into a dataset
- Organise and backup datasets easily
- Find datasets of interest
- Verify the contents of datasets
- Discover and work with data programatically

1.2 Descriptive documentation

1.2.1 Quickstart

The easiest way to create a dataset is to use the *dtoolcore.DataSetCreator* context manager.

The code below creates a frozen dataset without any metadata or data.

```
>>> from dtoolcore import DataSetCreator
>>> with DataSetCreator("my-awesome-dataset", "/tmp") as ds_creator:
...     uri = ds_creator.uri
```

Clearly, this dataset is not very interesting. However, we can use it to show how one can load a *dtoolcore.DataSet* instance from a dataset's URI, using the *dtoolcore.DataSet.from_uri()* method.

```
>>> from dtoolcore import DataSet
>>> dataset = DataSet.from_uri(uri)
>>> print(dataset.name)
my-awesome-dataset
```

1.2.2 Creating a dataset

A dtool dataset packages data and metadata. Descriptive metadata is stored in a "README" file at the top level of the dataset. It is best practise to use the YAML file format for the README.

The code below creates a variable that holds a string with descriptive metadata in YAML format.

```
>>> readme_content = "---\ndescription: three text files with animals"
```

The `readme_content` can be added to the dataset when creating a *dtoolcore.DataSetCreator* context manager.

```
>>> with DataSetCreator("animal-dataset", "/tmp", readme_content) as ds_creator:
...     animal_ds_uri = ds_creator.uri
...     for animal in ["cat", "dog", "parrot"]:
...         handle = animal + ".txt"  # Unix-like relpath
...         fpath = ds_creator.prepare_staging_abspath_promise(handle)
...         with open(fpath, "w") as fh:
...             fh.write(animal)
...
```

The code above does several things. It stores the URI of the dataset in the variable `animal_ds_uri`. It loops over the strings `cat`, `dog`, `parrot` and creates a so called "handle" for each one of them. A handle is a human readable label of an item in a dataset. It has to be unique and look like a Unix-style relpath. The handle is then passed into the *dtoolcore.DataSetCreator.prepare_staging_abspath_promise()* method which returns the absolute path to a file that needs to be created within the lifetime of the context manager. Otherwise a `dtoolcore.DtoolCoreBrokenStagingPromise` exception is raised. The code then uses these absolute paths to create files in these locations. When the context manager exits these files are added to the dataset, the temporary location where the files were created is deleted and the dataset is frozen.

1.2.3 Working with items in a dataset

Below a dataset is loaded from the `animal_ds_uri`.

```
>>> animal_dataset = DataSet.from_uri(animal_ds_uri)
```

The readme content can be accessed using the *dtoolcore.DataSet.get_readme_content()* method.

```
>>> print(animal_dataset.get_readme_content())
---
description: three text files with animals
```

Items in a dataset are accessed using their identifiers. The item identifiers can be accessed using the *dtoolcore.DataSet.identifiers* property.

```
>>> for i in animal_dataset.identifiers:
...     print(i)
...
e55aada093b34671ec2f9467fe83f0d3d8c31f30
d25102a700e072b528db79a0f22b3a5ffe5e8f5d
26f0d76fb3c3e34f0c7c8b7c3461b7495761835c
```

To view information about each item one can use the *dtoolcore.DataSet.item_properties()* method that returns a dictionary with the items `hash`, `size_in_bytes`, `relpath` (also known as "handle").

```
>>> for i in animal_dataset.identifiers:
...     item_props = animal_dataset.item_properties(i)
...     info_str = "{hash} {size_in_bytes} {relpath}".format(**item_props)
...     print(info_str)
...
d077f244def8a70e5ea758bd8352fcd8 3 cat.txt
68238cd792d215bdfdddc7bbb6d10db4 6 parrot.txt
06d80eb0c50b49a509b49f2424e8c805 3 dog.txt
```

To get the content of an item one can use the *dtoolcore.DataSet.item_content_abspath()* method. The method guarantees that the content of the item will be available in the abspath provided. This is important when working with datasets stored in the cloud, for example in an AWS S3 bucket.

```
>>> for i in animal_dataset.identifiers:
...     fpath = animal_dataset.item_content_abspath(i)
...     with open(fpath, "r") as fh:
...         print(fh.read())
...
cat
parrot
dog
```

1.2.4 Annotating a dataset with key/value pairs

The descriptive metadata in the readme is not ideally suited for programatic access to metadata. If one needs to interact with metadata programatically it is much easier to do so using so called "annotations". These are key/value pairs that can be added to a dataset.

In the code below the *dtoolcore.DataSet.put_annotation()* method is used to add add the key/value pair "category"/"pets" to the dataset.

```
>>> animal_dataset.put_annotation("category", "pets")
```

The *dtoolcore.DataSet.get_annotation()* can then be used to access the value of the "category" annotation.

```
>>> print(animal_dataset.get_annotation("category"))
pets
```

It is also possible to add an annotation to a dataset inside a *dtoolcore.DataSetCreator* conext manager using the *dtoolcore.DataSetCreator.put_annotation()* method.

1.2.5 Working with item metadata

It is also possible to add per item metadata. This is, for example, useful if one wants to access only a subset of items from a dataset. Below is a dictionary that can be used to look up the family of a set of animals.

```
>>> family = {
...     "tiger": "felidae",
...     "lion": "felidae",
...     "horse": "equidae"
... }
```

The code below creates a new dataset and adds the "family" of the animal as a piece of metadata to each item using the *dtoolcore.DataSetCreator.add_item_metadata()* method.

```
>>> with DataSetCreator("animal-2-dataset", "/tmp") as ds_creator:
...     animal2_ds_uri = ds_creator.uri
...     for animal in ["tiger", "lion", "horse"]:
...         handle = animal + ".txt"  # Unix-like relpath
...         fpath = ds_creator.prepare_staging_abspath_promise(handle)
...         with open(fpath, "w") as fh:
...             fh.write(animal)
...         ds_creator.add_item_metadata(handle, "family", family[animal])
...
```

Per item metadata are stored in what is referred to as "overlays". It is possible to get back the content of an overlay using the *dtoolcore.DataSet.get_overlay()* method.

```
>>> animal2_dataset = DataSet.from_uri(animal2_ds_uri)
>>> family_overlay = animal2_dataset.get_overlay("family")
```

The `family_overlay` is a Python dictonary, where they keys correspond to the item identifiers.

```
>>> for key, value in family_overlay.items():
...     print("{} {}".format(key, value))
...
85b263904920cc18caa5630e4124f4311847e6b8 felidae
433635d53dae167009941349491abf7aae9becbd felidae
f480009aa5a5c43d09f40f39df7a5a3ec5f42237 equidae
```

The code below uses this per item metadata to only process the cats ("felidae").

```
>>> for i in animal2_dataset.identifiers:
...     if family_overlay[i] != "felidae":
...         continue
...     fpath = animal2_dataset.item_content_abspath(i)
...     with open(fpath, "r") as fh:
...         print(fh.read())
...
lion
tiger
```

To add an overlay to an existing dataset one can use the *dtoolcore.DataSet.put_overlay* method. This takes as input a dictonary where each item has a keyed entry.

1.2.6 Creating a derived dataset

In data processing it can be useful to track the provenance of the input. This is most easily done by making use of the `dtoolcore.DerivedDataSetCreator` context manager.

```
>>> from dtoolcore import DerivedDataSetCreator
```

Suppose we wanted to transfor the animals from the `animal_dataset` into the sounds that they make. Let's create a dictionary to help us do this.

```
>>> animal_sounds = {
...     "dog": "bark",
...     "cat": "meow",
...     "parrot": "squak"
... }
...
```

The code below creates a dataset derived from the `animal_dataset`.

```
>>> with DerivedDataSetCreator("animal-sounds-dataset", "/tmp", animal_dataset) as ds_
↪creator:
...     animal_sounds_ds_uri = ds_creator.uri
...     for i in animal_dataset.identifiers:
...         input_abspath = animal_dataset.item_content_abspath(i)
...         with open(input_abspath, "r") as fh:
...             animal = fh.read()
...         handle = animal_dataset.item_properties(i)["relpath"]
...         output_abspath = ds_creator.prepare_staging_abspath_promise(handle)
...         with open(output_abspath, "w") as fh:
```

(continues on next page)

(continued from previous page)

```
...             fh.write(animal_sounds[animal])
...
```

The derived dataset has now been created and it can be loaded using the *dtoolcore.DataSet.from_uri* method.

```
>>> animal_sounds_dataset = DataSet.from_uri(animal_sounds_ds_uri)
```

This has been automatically annotated with `source_dataset_name`, `source_dataset_uuid`, and `source_dataset_uri`.

```
>>> print(animal_sounds_dataset.get_annotation("source_dataset_name"))
animal-dataset
```

The code example below looks at the data in the `animal-sounds-dataset` dataset.

```
>>> for i in animal_sounds_dataset.identifiers:
...     handle = animal_sounds_dataset.item_properties(i)["relpath"]
...     fpath = animal_sounds_dataset.item_content_abspath(i)
...     with open(fpath, "r") as fh:
...         content = fh.read()
...     print("{} - {}".format(handle, content))
...
cat.txt - meow
parrot.txt - squak
dog.txt - bark
```

1.2.7 Tagging a dataset

It is possible to add "tags" to datasets and protodatasets. Tags are labels that can be used to organise datasets into groups. A tag is basically a short piece of text describing a dataset. It is possible to label a dataset with several tags.

In the code below we label the `animal_sounds_dataset` with the tags "animal" and "sound".

```
>>> animal_sounds_dataset.put_tag("animal")
>>> animal_sounds_dataset.put_tag("sound")
```

The code below iterates over all the tags in the dataset and prints them.

```
>>> for tag in animal_sounds_dataset.list_tags():
...     print(tag)
...
animal
sound
```

It is possible to delete a tag.

```
>>> animal_sounds_dataset.delete_tag("sound")
```

If the tag does not exist the command above would simply do nothing, but would not raise any exceptions.

1.3 API documentation

1.3.1 dtoolcore

API for creating and interacting with dtool datasets.

class dtoolcore.**DataSet**(*uri*, *admin_metadata*, *config_path=None*)
Class for reading the contents of a dataset.

base_uri
Return the base URI of the dataset.

delete_tag(*tag*)
Delete a tag from a dataset.

Parameters **tag** – tag

Raises DtoolCoreKeyError if the tag does not exist

classmethod from_uri(*uri*, *config_path=None*)
Return an existing *dtoolcore.DataSet* from a URI.

Parameters **uri** – unique resource identifier where the existing *dtoolcore.DataSet* is stored

Returns *dtoolcore.DataSet*

generate_manifest(*progressbar=None*)
Return manifest generated from knowledge about contents.

get_annotation(*annotation_name*)
Return annotation.

Parameters **annotation_name** – name of the annotation

Raises DtoolCoreKeyError if the annotation does not exist

Returns annotation

get_overlay(*overlay_name*)
Return overlay as a dictionary.

Parameters **overlay_name** – name of the overlay

Returns overlay as a dictionary

get_readme_content()
Return the content of the README describing the dataset.

Returns content of README as a string

identifiers
Return iterable of dataset item identifiers.

item_content_abspath(*identifier*)
Return absolute path at which item content can be accessed.

Parameters **identifier** – item identifier

Returns absolute path from which the item content can be accessed

item_properties(*identifier*)
Return properties of the item with the given identifier.

Parameters **identifier** – item identifier

Returns dictionary of item properties from the manifest

list_annotation_names()
Return list of annotation names.

list_overlay_names()
Return list of overlay names.

list_tags()
Return the dataset's tags as a list.

name
Return the name of the dataset.

put_annotation(*annotation_name*, *annotation*)
Store annotation so that it is accessible by the given name.

Parameters

- **annotation_name** – name of the annotation
- **annotation** – JSON serialisable value or data structure

Raises DtoolCoreInvalidNameError if the annotation name is invalid

put_overlay(*overlay_name*, *overlay*)
Store overlay so that it is accessible by the given name.

Parameters

- **overlay_name** – name of the overlay
- **overlay** – overlay must be a dictionary where the keys are identifiers in the dataset

Raises DtoolCoreTypeError if the overlay is not a dictionary, DtoolCoreValueError if identifiers in overlay and dataset do not match DtoolCoreInvalidNameError if the overlay name is invalid

put_readme(*content*)
Update the README of the dataset and backup the previous README.

The client is responsible for ensuring that the content is valid YAML.

Parameters **content** – string to put into the README

put_tag(*tag*)
Annotate the dataset with a tag.

Parameters **tag** – tag

Raises DtoolCoreInvalidNameError if the tag is invalid

Raises DtoolCoreValueError if the tag is not a string

update_name(*new_name*)
Update the name of the proto dataset.

Parameters **new_name** – the new name of the proto dataset

uri
Return the URI of the dataset.

uuid
Return the UUID of the dataset.

class dtoolcore.**DataSetCreator**(*name*, *base_uri*, *readme_content=''*, *creator_username=None*)
Context manager for creating a dataset.

Inside the context manager one works on a proto dataset. When exiting the context manager the proto dataset is automatically frozen into a dataset, unless an exception has been raised in the context manager.

add_item_metadata(*handle*, *key*, *value*)
Add metadata to a specific item in the *dtoolcore.ProtoDataSet*.

Parameters

- **handle** – handle representing the relative path of the item in the *dtoolcore.ProtoDataSet*
- **key** – metadata key
- **value** – metadata value

name
Return the dataset name.

prepare_staging_abspath_promise(*handle*)
Return abspath and handle to stage a file.

For getting access to an abspath that can be used to write output to. It is the responsibility of this method to create any missing subdirectories. It is the responsibility of the user to create the file associated with the abspath.

Use the abspath to create a file in the staging directory. The file will be added to the dataset when exiting the context handler.

The handle can be used to generate an identifier for the item in the dataset using the *dtoolcore.utils.generate_identifier()* function.

Parameters **handle** – Unix like relpath

Returns absolute path to the file in staging area that the user promises to create

put_annotation(*annotation_name*, *annotation*)
Store annotation so that it is accessible by the given name.

Parameters

- **annotation_name** – name of the annotation
- **annotation** – JSON serialisable value or data structure

Raises DtoolCoreInvalidNameError if the annotation name is invalid

put_item(*fpath*, *relpath*)
Put an item into the dataset.

Parameters

- **fpath** – path to the item on disk
- **relpath** – relative path name given to the item in the dataset as a handle

Returns the handle given to the item

put_readme(*content*)
Update the README of the dataset and backup the previous README.

The client is responsible for ensuring that the content is valid YAML.

Parameters **content** – string to put into the README

put_tag(*tag*)
Annotate the dataset with a tag.

Parameters **tag** – tag

Raises DtoolCoreInvalidNameError if the tag is invalid

Raises ValueError if the tag is not a string

staging_directory
Return the staging directory.

An ephemeral directory that only exists within the DataSetCreator context manger. It can be used as a location to write output files that need to be added to the dataset.

The easiest way to add a file here is to use the `dtoolcore.DataSetCreator.get_staging_fpath()` method to get a path to write content to.

If you write files directly to the staging directory you will need to register them using the `dtoolcore.DataSetCreator.register_output_file()` method.

uri
Return the dataset URI.

class dtoolcore.**DerivedDataSetCreator**(*name*, *base_uri*, *source_dataset*, *readme_content=''*, *creator_username=None*)
Context manager for creating a derived dataset.

A derived dataset automatically has information about the source dataset (name, URI and UUID) automatically added to the readme and to annotations. It adds the "source_name", "source_uri", and "source_uuid" as annotations and to the descriptive metadata in the readme.

Inside the context manager one works on a proto dataset. When exiting the context manager the proto dataset is automatically frozen into a dataset, unless an exception has been raised in the context manager.

add_item_metadata(*handle*, *key*, *value*)
Add metadata to a specific item in the *dtoolcore.ProtoDataSet*.

Parameters

- **handle** – handle representing the relative path of the item in the *dtoolcore.ProtoDataSet*
- **key** – metadata key
- **value** – metadata value

name
Return the dataset name.

prepare_staging_abspath_promise(*handle*)
Return abspath and handle to stage a file.

For getting access to an abspath that can be used to write output to. It is the responsibility of this method to create any missing subdirectories. It is the responsibility of the user to create the file associated with the abspath.

Use the abspath to create a file in the staging directory. The file will be added to the dataset when exiting the context handler.

The handle can be used to generate an identifier for the item in the dataset using the *dtoolcore.utils.generate_identifier()* function.

Parameters **handle** – Unix like relpath

Returns absolute path to the file in staging area that the user promises to create

put_annotation(*annotation_name*, *annotation*)
Store annotation so that it is accessible by the given name.

Parameters

- **annotation_name** – name of the annotation
- **annotation** – JSON serialisable value or data structure

Raises DtoolCoreInvalidNameError if the annotation name is invalid

put_item(*fpath*, *relpath*)
Put an item into the dataset.

Parameters

- **fpath** – path to the item on disk
- **relpath** – relative path name given to the item in the dataset as a handle

Returns the handle given to the item

put_readme(*content*)
Update the README of the dataset and backup the previous README.

The client is responsible for ensuring that the content is valid YAML.

Parameters **content** – string to put into the README

put_tag(*tag*)
Annotate the dataset with a tag.

Parameters **tag** – tag

Raises DtoolCoreInvalidNameError if the tag is invalid

Raises ValueError if the tag is not a string

staging_directory
Return the staging directory.

An ephemeral directory that only exists within the DataSetCreator context manger. It can be used as a location to write output files that need to be added to the dataset.

The easiest way to add a file here is to use the `dtoolcore.DataSetCreator.get_staging_fpath()` method to get a path to write content to.

If you write files directly to the staging directory you will need to register them using the `dtoolcore.DataSetCreator.register_output_file()` method.

uri
Return the dataset URI.

exception dtoolcore.**DtoolCoreBrokenStagingPromise**

errno
exception errno

filename
exception filename

strerror
exception strerror

exception dtoolcore.**DtoolCoreInvalidNameError**

exception dtoolcore.**DtoolCoreKeyError**

exception dtoolcore.**DtoolCoreTypeError**

exception dtoolcore.**DtoolCoreValueError**

class dtoolcore.**ProtoDataSet**(*uri*, *admin_metadata*, *config_path=None*)
Class for building up a dataset.

add_item_metadata(*handle*, *key*, *value*)
Add metadata to a specific item in the *dtoolcore.ProtoDataSet*.

Parameters

- **handle** – handle representing the relative path of the item in the *dtoolcore.ProtoDataSet*
- **key** – metadata key
- **value** – metadata value

base_uri
Return the base URI of the dataset.

create()
Create the required directory structure and admin metadata.

delete_tag(*tag*)
Delete a tag from a dataset.

Parameters **tag** – tag

Raises DtoolCoreKeyError if the tag does not exist

freeze(*progressbar=None*)
Convert *dtoolcore.ProtoDataSet* to *dtoolcore.DataSet*.

classmethod from_uri(*uri*, *config_path=None*)
Return an existing *dtoolcore.ProtoDataSet* from a URI.

Parameters **uri** – unique resource identifier where the existing *dtoolcore.ProtoDataSet* is stored

Returns *dtoolcore.ProtoDataSet*

generate_manifest(*progressbar=None*)
Return manifest generated from knowledge about contents.

get_annotation(*annotation_name*)
Return annotation.

Parameters **annotation_name** – name of the annotation

Raises DtoolCoreKeyError if the annotation does not exist

Returns annotation

get_readme_content()
Return the content of the README describing the dataset.

Returns content of README as a string

list_annotation_names()
Return list of annotation names.

list_tags()
Return the dataset's tags as a list.

name
Return the name of the dataset.

put_annotation(*annotation_name*, *annotation*)
Store annotation so that it is accessible by the given name.

Parameters

- **annotation_name** – name of the annotation
- **annotation** – JSON serialisable value or data structure

Raises DtoolCoreInvalidNameError if the annotation name is invalid

put_item(*fpath*, *relpath*)
Put an item into the dataset.

Parameters

- **fpath** – path to the item on disk
- **relpath** – relative path name given to the item in the dataset as a handle, i.e. a Unix-like relpath

Returns the handle given to the item

put_readme(*content*)
Put content into the README of the dataset.

The client is responsible for ensuring that the content is valid YAML.

Parameters **content** – string to put into the README

put_tag(*tag*)
Annotate the dataset with a tag.

Parameters **tag** – tag

Raises DtoolCoreInvalidNameError if the tag is invalid

Raises DtoolCoreValueError if the tag is not a string

update_name(*new_name*)
Update the name of the proto dataset.

Parameters **new_name** – the new name of the proto dataset

uri
Return the URI of the dataset.

uuid
Return the UUID of the dataset.

dtoolcore.**copy**(*src_uri*, *dest_base_uri*, *config_path=None*, *progressbar=None*)
Copy a dataset to another location.

Parameters

- **src_uri** – URI of dataset to be copied
- **dest_base_uri** – base of URI for copy target
- **config_path** – path to dtool configuration file

Returns URI of new dataset

dtoolcore.**copy_resume**(*src_uri*, *dest_base_uri*, *config_path=None*, *progressbar=None*)
Resume coping a dataset to another location.

Items that have been copied to the destination and have the same size as in the source dataset are skipped. All other items are copied across and the dataset is frozen.

Parameters

- **src_uri** – URI of dataset to be copied
- **dest_base_uri** – base of URI for copy target
- **config_path** – path to dtool configuration file

Returns URI of new dataset

dtoolcore.**create_derived_proto_dataset**(*name*, *base_uri*, *source_dataset*, *readme_content=''*, *creator_username=None*)
Return *dtoolcore.ProtoDataSet* instance.

It adds the "source_name", "source_uri", and "source_uuid" as annotations.

Parameters

- **name** – dataset name
- **base_uri** – base URI for proto dataset
- **source_dataset** – source dataset
- **readme_content** – content of README as a string
- **creator_username** – creator username

dtoolcore.**create_proto_dataset**(*name*, *base_uri*, *readme_content=''*, *creator_username=None*)
Return *dtoolcore.ProtoDataSet* instance.

Parameters

- **name** – dataset name
- **base_uri** – base URI for proto dataset
- **readme_content** – content of README as a string
- **creator_username** – creator username

dtoolcore.**generate_admin_metadata**(*name*, *creator_username=None*)
Return admin metadata as a dictionary.

dtoolcore.**generate_proto_dataset**(*admin_metadata*, *base_uri*, *config_path=None*)
Return *dtoolcore.ProtoDataSet* instance.

Parameters

- **admin_metadata** – dataset administrative metadata
- **base_uri** – base URI for proto dataset
- **config_path** – path to dtool configuration file

dtoolcore.**iter_datasets_in_base_uri**(*base_uri*)
Yield *dtoolcore.DataSet* instances present in the base URI.

Params base_uri base URI

Returns iterator yielding *dtoolcore.DataSet* instances

dtoolcore.**iter_proto_datasets_in_base_uri**(*base_uri*)
Yield *dtoolcore.ProtoDataSet* instances present in the base URI.

Params base_uri base URI

Returns iterator yielding *dtoolcore.ProtoDataSet* instances

1.3.2 dtoolcore.compare

Module with helper functions for comparing datasets.

dtoolcore.compare.**diff_content**(*a*, *reference*, *progressbar=None*)
Return list of tuples where content differ.

Tuple structure: (identifier, hash in a, hash in reference)

Assumes list of identifiers in a and b are identical.

Storage broker of reference used to generate hash for files in a.

Parameters

- **a** – first *dtoolcore.DataSet*
- **b** – second *dtoolcore.DataSet*

Returns list of tuples for all items with different content

dtoolcore.compare.**diff_identifiers**(*a*, *b*)
Return list of tuples where identifiers in datasets differ.

Tuple structure: (identifier, present in a, present in b)

Parameters

- **a** – first *dtoolcore.DataSet*
- **b** – second *dtoolcore.DataSet*

Returns list of tuples where identifiers in datasets differ

dtoolcore.compare.**diff_sizes**(*a*, *b*, *progressbar=None*)
Return list of tuples where sizes differ.

Tuple structure: (identifier, size in a, size in b)

Assumes list of identifiers in a and b are identical.

Parameters

- **a** – first *dtoolcore.DataSet*
- **b** – second *dtoolcore.DataSet*

Returns list of tuples for all items with different sizes

1.3.3 dtoolcore.filehasher

Module for generating file hashes.

class dtoolcore.filehasher.**FileHasher**(*hash_func*)
Class for associating hash functions with names.

dtoolcore.filehasher.**hashsum_digest**(*hasher*, *filename*)
Helper function for creating hash functions.

See implementation of `dtoolcore.filehasher.shasum()` for more usage details.

dtoolcore.filehasher.**hashsum_hexdigest**(*hasher*, *filename*)
Helper function for creating hash functions.

See implementation of `dtoolcore.filehasher.shasum()` for more usage details.

dtoolcore.filehasher.**md5sum_digest**(*filename*)
Return digest of MD5sum of file.

Parameters **filename** – path to file

Returns shasum of file

dtoolcore.filehasher.**md5sum_hexdigest**(*filename*)
Return hex digest of MD5sum of file.

Parameters **filename** – path to file

Returns shasum of file

dtoolcore.filehasher.**sha1sum_hexdigest**(*filename*)
Return hex digest of SHA-1 hash of file.

Parameters **filename** – path to file

Returns shasum of file

dtoolcore.filehasher.**sha256sum_hexdigest**(*filename*)
Return hex digest of SHA-256 hash of file.

Parameters **filename** – path to file

Returns shasum of file

1.3.4 dtoolcore.storagebroker

Disk storage broker.

class dtoolcore.storagebroker.**BaseStorageBroker**
Base storage broker class defining the required interface.

add_item_metadata(*handle*, *key*, *value*)
Store the given key:value pair for the item associated with handle.

Parameters

- **handle** – handle for accessing an item before the dataset is frozen
- **key** – metadata key
- **value** – metadata value

create_structure()
Create necessary structure to hold a dataset.

delete_key(*key*)
Delete the file/object associated with the key.

delete_tag(*tag*)
Delete a tag from a dataset.

Parameters **tag** – tag

generate_base_uri(*uri*)
Return dataset base URI given a uri.

classmethod generate_uri(*name*, *uuid*, *base_uri*)
Return dataset URI.

get_admin_metadata()
Return the admin metadata as a dictionary.

get_admin_metadata_key()
Return the admin metadata key.

get_annotation(*annotation_name*)
Return value of the annotation associated with the key.

Returns annotation (string, int, float, bool)

Raises DtoolCoreAnnotationKeyError if the annotation does not exist

get_annotation_key(*annotation_name*)
Return the annotation key.

get_hash(*handle*)
Return the hash.

get_item_abspath(*identifier*)
Return absolute path at which item content can be accessed.

Parameters **identifier** – item identifier

Returns absolute path from which the item content can be accessed

get_item_metadata(*handle*)
Return dictionary containing all metadata associated with handle.

In other words all the metadata added using the `add_item_metadata` method.

Parameters **handle** – handle for accessing an item before the dataset is frozen

Returns dictionary containing item metadata

get_manifest()
Return the manifest as a dictionary.

get_manifest_key()
Return the manifest key.

get_overlay(*overlay_name*)
Return overlay as a dictionary.

get_overlay_key(*overlay_name*)
Return the overlay key.

get_readme_content()
Return the README descriptive metadata as a string.

get_readme_key()
Return the admin metadata key.

get_relpath(*handle*)
Return the relative path.

get_size_in_bytes(*handle*)
Return the size in bytes.

get_tag_key(*tag*)
Return the tag key.

get_text(*key*)
Return the text associated with the key.

get_utc_timestamp(*handle*)
Return the UTC timestamp.

has_admin_metadata()
Return True if the administrative metadata exists.

This is the definition of being a "dataset".

item_properties(*handle*)
Return properties of the item with the given handle.

iter_item_handles()
Return iterator over item handles.

list_annotation_names()
Return list of annotation names.

classmethod list_dataset_uris(*base_uri*, *config_path*)
Return list containing URIs in location given by base_uri.

list_overlay_names()
Return list of overlay names.

list_tags()
Return list of tags.

post_freeze_hook()
Post *dtoolcore.ProtoDataSet.freeze()* cleanup actions.

This method is called at the end of the *dtoolcore.ProtoDataSet.freeze()* method.

In the `dtoolcore.storage_broker.DiskStorageBroker` it removes the temporary directory for storing item metadata fragment files.

pre_freeze_hook()
Pre *dtoolcore.ProtoDataSet.freeze()* actions.

This method is called at the beginning of the *dtoolcore.ProtoDataSet.freeze()* method.

It may be useful for remote storage backends to generate caches to remove repetitive time consuming calls

put_admin_metadata(*admin_metadata*)
Store the admin metadata.

put_annotation(*annotation_name*, *annotation*)
Set/update value of the annotation associated with the key.

Raises DtoolCoreAnnotationTypeError if the type of the value is not str, int, float or bool.

put_item(*fpath*, *relpath*)
Put item with content from fpath at relpath in dataset.

Missing directories in relpath are created on the fly.

Parameters

- **fpath** – path to the item on disk
- **relpath** – relative path name given to the item in the dataset as a handle

Returns the handle given to the item

put_manifest(*manifest*)
Store the manifest.

put_overlay(*overlay_name*, *overlay*)
Store the overlay.

put_readme(*content*)
Store the readme descriptive metadata.

put_tag(*tag*)
Annotate the dataset with a tag.

put_text(*key*, *text*)
Put the text into the storage associated with the key.

update_readme(*content*)
Update the readme descriptive metadata.

class dtoolcore.storagebroker.**DiskStorageBroker**(*uri*, *config_path=None*)
Storage broker to interact with datasets on local disk storage.

The *dtoolcore.ProtoDataSet* class uses the dtoolcore.storage_broker.DiskStorageBroker to construct datasets by writing to disk and the *dtoolcore.DataSet* class uses it to read datasets from disk.

add_item_metadata(*handle*, *key*, *value*)
Store the given key:value pair for the item associated with handle.

Parameters

- **handle** – handle for accessing an item before the dataset is frozen
- **key** – metadata key
- **value** – metadata value

delete_key(*key*)
Delete the file/object associated with the key.

classmethod generate_uri(*name*, *uuid*, *base_uri*)
Return dataset URI.

get_admin_metadata_key()
Return the path to the admin metadata file.

get_annotation_key(*annotation_name*)
Return the path to the annotation file.

get_dtool_readme_key()
Return the path to the dtool readme file.

get_hash(*handle*)
Return the hash.

get_item_abspath(*identifier*)
Return absolute path at which item content can be accessed.

Parameters **identifier** – item identifier

Returns absolute path from which the item content can be accessed

get_item_metadata(*handle*)
Return dictionary containing all metadata associated with handle.

In other words all the metadata added using the `add_item_metadata` method.

Parameters **handle** – handle for accessing an item before the dataset is frozen

Returns dictionary containing item metadata

get_manifest_key()
Return the path to the readme file.

get_overlay_key(*overlay_name*)
Return the path to the overlay file.

get_readme_key()
Return the path to the readme file.

get_size_in_bytes(*handle*)
Return the size in bytes.

get_structure_key()
Return the path to the structure parameter file.

get_tag_key(*tag*)
Return the path to the tag file.

get_text(*key*)
Return the text associated with the key.

get_utc_timestamp(*handle*)
Return the UTC timestamp.

has_admin_metadata()
Return True if the administrative metadata exists.

This is the definition of being a "dataset".

hasher = <dtoolcore.filehasher.FileHasher object>
Attribute used by *dtoolcore.ProtoDataSet* to write the hash function name to the manifest.

iter_item_handles()
Return iterator over item handles.

key = 'file'
Attribute used to define the type of storage broker.

list_annotation_names()
Return list of annotation names.

classmethod list_dataset_uris(*base_uri*, *config_path*)
Return list containing URIs in location given by base_uri.

list_overlay_names()
Return list of overlay names.

list_tags()
Return list of tags.

post_freeze_hook()
Post *dtoolcore.ProtoDataSet.freeze()* cleanup actions.

This method is called at the end of the *dtoolcore.ProtoDataSet.freeze()* method.

In the `dtoolcore.storage_broker.DiskStorageBroker` it removes the temporary directory for storing item metadata fragment files.

pre_freeze_hook()
Pre *dtoolcore.ProtoDataSet.freeze()* actions.

This method is called at the beginning of the *dtoolcore.ProtoDataSet.freeze()* method.

It may be useful for remote storage backends to generate caches to remove repetitive time consuming calls

put_item(*fpath*, *relpath*)
Put item with content from fpath at relpath in dataset.

Missing directories in relpath are created on the fly.

Parameters

- **fpath** – path to the item on disk
- **relpath** – relative path name given to the item in the dataset as a handle, i.e. a Unix-like relpath

Returns the handle given to the item

put_text(*key*, *text*)
Put the text into the storage associated with the key.

exception dtoolcore.storagebroker.**DiskStorageBrokerValidationWarning**

exception dtoolcore.storagebroker.**StorageBrokerOSError**

1.3.5 dtoolcore.utils

Utility functions for dtoolcore.

dtoolcore.utils.**base64_to_hex**(*input_string*)
Retun the hex encoded version of the base64 encoded input string.

dtoolcore.utils.**cross_platform_getuser**(*is_windows*, *no_username_in_env*)
Return the username or "unknown".

The function returns "unknown" if the platform is windows and the username environment variable is not set.

dtoolcore.utils.**generate_identifier**(*handle*)
Return identifier from a ProtoDataSet handle.

dtoolcore.utils.**generous_parse_uri**(*uri*)
Return a urlparse.ParseResult object with the results of parsing the given URI. This has the same properties as the result of parse_uri.

When passed a relative path, it determines the absolute path, sets the scheme to file, the netloc to localhost and returns a parse of the result.

dtoolcore.utils.**get_config_value**(*key*, *config_path=None*, *default=None*)
Get a configuration value.

Preference: 1. From environment 2. From JSON configuration file supplied in `config_path` argument 3. The default supplied to the function

Parameters

- **key** – name of lookup value
- **config_path** – path to JSON configuration file
- **default** – default fall back value

Returns value associated with the key

dtoolcore.utils.**get_config_value_from_file**(*key*, *config_path=None*, *default=None*)
Return value if key exists in file.

Return default if key not in config.

dtoolcore.utils.**getuser**()
Return the username.

dtoolcore.utils.**handle_to_osrelpath**(*handle*, *is_windows=False*)
Return OS specific relpath from handle.

dtoolcore.utils.**mkdir_parents**(*path*)
Create the given directory path. This includes all necessary parent directories. Does not raise an error if the directory already exists. :param path: path to create

dtoolcore.utils.**name_is_valid**(*name*)
Return True if the dataset name is valid.

The name can only be 80 characters long. Valid characters: Alpha numeric characters [0-9a-zA-Z] Valid special characters: - _ .

dtoolcore.utils.**relpath_to_handle**(*relpath*, *is_windows=False*)
Return handle from relpath.

Handles are Unix style relpaths. Converts Windows relpath to Unix style relpath. Strips "./" prefix.

dtoolcore.utils.**sanitise_uri**(*uri*)
Return fully qualified uri from the input, which might be a relpath.

dtoolcore.utils.**sha1_hexdigest**(*input_string*)
Return hex digest of the sha1sum of the input_string.

dtoolcore.utils.**timestamp**(*datetime_obj*)
Return Unix timestamp as float.

The number of seconds that have elapsed since January 1, 1970.

dtoolcore.utils.**unix_to_windows_path**(*unix_path*)
Return Windows path.

dtoolcore.utils.**windows_to_unix_path**(*win_path*)
Return Unix path.

dtoolcore.utils.**write_config_value_to_file**(*key*, *value*, *config_path=None*)
Write key/value pair to config file.

1.4 MIT License

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Python Module Index

d

Index

G

H

I

K

L

M

N

P

R

S

T

U

W