bw-processing

Chris Mutel

Sep 18, 2023

CONTENTS

Library for storing numeric data for use in matrix-based calculations. Designed for use with the Brightway life cycle assessment framework.

CHAPTER

ONE

TABLE OF CONTENTS

CHAPTER

TWO

BACKGROUND

The Brightway LCA framework has stored data used in constructing matrices in binary form as numpy arrays for years. This package is an evolution of that approach, and adds the following features:

- **Consistent names for row and column fields**. Previously, these changed for each matrix, to reflect the role each row or column played in the model. Now they are always the same for all arrays (`"row"` and `"col"`), making the code simpler and easier to use.
- **Provision of metadata**. Numpy binary files are only data - `bw_processing` also produces a metadata file following the data package standard. Things like data license, version, and unique id are now explicit and always included.
- **Support for vector and array data**. Vector (i.e. only one possible value per input) and array (i.e. many possible values, also called presamples) data are now both natively supported in data packages.
- **Portability**. Processed arrays can include metadata that allows for reindexing on other machines, so that processed arrays can be distributed and reused. Before, this was not possible, as integer IDs were randomly assigned on each computer, and would be different from machine to machine or even across Brightway projects.
- **Dynamic data sources**. Instead of requiring that data for matrix construction be present and savedd on disk, it can now be generated dynamically, either through code running locally or on another computer system. This is a big step towards embeddding life cycle assessment in a web of environmental models.
- **Use PyFilesystem2 for file IO**. The use of this library allows for data packages to be stored on your local computer, or on many logical or virtual file systems.
- **Simpler handling of numeric values whose sign should be flipped**. Sometimes it is more convenient to specify positive numbers in dataset definitions, even though such numbers should be negative when inserted into the resulting matrices. For example, in the technosphere matrix in life cycle assessment, products produced are positive and products consumed are negative, though both values are given as positive in datasets. Brightway used to use a type mapping dictionary to indicate which values in a matrix should have their sign flipped after insertion. Such mapping dictionaries are brittle and inelegant. `bw_processing` uses an optional boolean vector, called `flip`, to indicate if any values should be flipped.
- **Separation of uncertainty distribution parameters from other data**. Fitting data to a probability density function (PDF), or an estimate of such a PDF, is only one approach to quantitative uncertainty analysis. We would like to support other approaches, including direct sampling from real data. Therefore, uncertainty distribution parameters are stored separately, only loaded if needed, and are only one way to express quantitative uncertainty.

CHAPTER

THREE

CONCEPTS

3.1 Data packages

Data objects can be vectors or arrays. Vectors will always produce the same matrix, while arrays have multiple possible values for each element of the matrix. Arrays are a generalization of the presamples library.

3.2 Data needed for matrix construction

3.3 Vectors versus arrays

3.4 Persistent versus dynamic

Persistent data is fixed, and can be completely loaded into memory and used directly or written to disk. Dynamic data is only resolved as the data is used, during matrix construction and iteration. Dynamic data is provided by *interfaces* - Python code that either generates the data, or wraps data coming from other software. There are many possible use cases for data interfaces, including:

- Data that is provided by an external source, such as a web service
- Data that comes from an infinite python generator
- Data from another programming language
- Data that needs processing steps before it can be directly inserted into a matrix

Only the actual numerical values entered into the matrix is dynamic - the matrix index values (and optional flip vector) are still static, and need to be provided as Numpy arrays when adding dynamic resources.

Interfaces must implement a simple API. Dynamic vectors must support the python generator API, i.e. implement `__next__()`.

Dynamic arrays must pretend to be Numpy arrays, in that they need to implement `.shape` and `.__getitem__(args)`.

- `.shape` must return a tuple of two integers. The first should be the number of elements returned, though this is not used. The second should be the number of columns available - an integer. This second value can also be `None`, if the interface is infinite.
- `.__getitem__(args)` must return a one-dimensional Numpy array corresponding to the column `args[1]`. This method is called when one uses code like `some_array[: 20]`. In our case, we will always take all rows (the `:`), so the first value can be ignored.

Here are some example interfaces (also given in `bw_processing/examples/interfaces.py`):

```
import numpy as np


class ExampleVectorInterface:
    def __init__(self):
        self.rng = np.random.default_rng()
        self.size = self.rng.integers(2, 10)

    def __next__(self):
        return self.rng.random(self.size)


class ExampleArrayInterface:
    def __init__(self):
        rng = np.random.default_rng()
        self.data = rng.random((rng.integers(2, 10), rng.integers(2, 10)))

    @property
    def shape(self):
        return self.data.shape

    def __getitem__(self, args):
        if args[1] >= self.shape[1]:
            raise IndexError
        return self.data[:, args[1]]
```

3.4.1 Interface dehydrating and rehydrating

Serialized datapackages cannot contain executable code, both because of our chosen data formats, and for security reasons. Therefore, when loading a datapackage with an interface, that interface object needs to be reconstituted as Python code - we call this cycle dehydration and rehydration. Dehydration happens automatically when a datapackage is finalized with `finalize_serialization()`, but rehydration needs to be done manually using `rehydrate_interface()`. For example:

```
from fs.zipfs import ZipFS
from bw_processing import load_datapackage

my_dp = load_datapackage(ZipFS("some-path.zip"))
my_dp.rehydrate_interface("some-resource-name", ExampleVectorInterface())
```

You can list the dehydrated interfaces present with `.dehydrated_interfaces()`.

You can store useful information for the interface object initialization under the resource key `config`. This can be used in instantiating an interface if you pass `initialize_with_config`:

```
from fs.zipfs import ZipFS
from bw_processing import load_datapackage
import requests
import numpy as np


class MyInterface:
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, url):
        self.url = url

    def __next__(self):
        return np.array(requests.get(self.url).json())


my_dp = load_datapackage(ZipFS("some-path.zip"))
data_obj, resource_metadata = my_dp.get_resource("some-interface")
print(resource_metadata['config'])
>>> {"url": "example.com"}

my_dp.rehydrate_interface("some-interface", MyInterface, initialize_with_config=True)
# interface is substituted, need to retrieve it again
data_obj, resource_metadata = my_dp.get_resource("some-interface")
print(data_obj.url)
>>> "example.com"
```

3.5 Policies

Data package policies define how the data should be used. Policies apply to the entire data package; you may wish to adjust what is stored in which data packages to get the effect you desire.

There are two policies that apply to all data resources:

sum_intra_duplicates (default `True`): What to do if more than one data point for a given matrix element is given in each *vector or array resource*. If true, sum these values; otherwise, the last value provided is used.

sum_inter_duplicates (default: `False`): What to do if data from a given resource overlaps data already present in the matrix. If true, add the given value to the existing value; otherwise, the existing values will be overwritten.

There are three policies that apply only to array data resources, where a different column from the array is used in matrix construction each time the array is iterated over:

combinatorial (default `False`): If more than one array resource is available, this policy controls whether all possible combinations of columns are guaranteed to occur. If `combinatorial` is `True`, we use `itertools.combinations` to generate column indices for the respective arrays; if `False`, column indices are either completely random (with replacement) or sequential.

Note that you will get `StopIteration` if you exhaust all combinations when `combinatorial` is `True`.

Note that `combinatorial` cannot be `True` if infinite array interfaces are present.

sequential (default `False`): Array resources have multiple columns, each of which represents a valid system state. Default behaviour is to choose from these columns at random (including replacement), using a RNG and the data package `seed` value. If `sequential` is `True`, columns in each array will be chosen in order starting from column zero, and will rewind to zero if the end of the array is reached.

Note that if `combinatorial` is `True`, `sequential` is ignored; instead, the column indices are generated by `itertools.combinations`.

Please make sure you understand how `combinatorial` and `sequential` interact! There are three possibilities:

- `combinatorial` and `sequential` are both `False`. Columns are returned at random.
- `combinatorial` is `False`, `sequential` is `True`. Columns are returned in increasing numerical order without any interaction between the arrays.

- `combinatorial` is `True`, `sequential` is ignored: Columns are returned in increasing order, such that all combinations of the different array resources are provided. `StopIteration` is raised if you try to consume additional column indices.

CHAPTER

FOUR

INSTALL

Install using pip or conda (channel `cmutel`). Depends on `numpy` and `pandas` (for reading and writing CSVs). Has no explicit or implicit dependence on any other part of Brightway.

CHAPTER

FIVE

USAGE

The main interface for using this library is the `Datapackage` class. However, instead of creating an instance of this class directly, you should use the utility functions `create_datapackage` and `load_datapackage`.

A datapackage is a set of file objects (either in-memory or on disk) that includes a metadata file object, and one or more data resource files objects. The metadata file object includes both generic metadata (i.e. when it was created, the data license) and metadata specific to each data resource (how it can be used in calculations, its relationship to other data resources). Datapackages follow the data package standard.

5.1 Creating datapackages

Datapackages are created using `create_datapackage`, which takes the following arguments:

- dirpath: `str` or `pathlib.Path` object. Where the datapackage should be saved. `None` for in-memory datapackages.
- name: `str`: The name of the overall datapackage. Make it meaningful to you.
- id_: `str`, optional. A unique id for this package. Automatically generated if not given.
- metadata: `dict`, optional. Any additional metadata, such as license and author.
- overwrite: `bool`, default `False`. Overwrite an existing resource with the same `dirpath` and `name`.
- compress: `bool`, default `False`. Save to a zipfile, if saving to disk.

Calling this function return an instance of `Datapackage`. You still need to add data.

CHAPTER

SIX

CONTRIBUTING

Your contribution is welcome! Please follow the pull request workflow, even for minor changes.

When contributing to this repository with a major change, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository.

Please note we have a code of conduct, please follow it in all your interactions with the project.

6.1 Documentation and coding standards

- Black formatting
- Semantic versioning

CHAPTER

SEVEN

MAINTAINERS

- Chris Mutel

CHAPTER

EIGHT

LICENSE

BSD-3-Clause. Copyright 2020 Chris Mutel.

8.1 bw_processing

8.1.1 bw_processing package

Subpackages

bw_processing.examples package

Submodules

bw_processing.examples.interfaces module

class bw_processing.examples.interfaces.**ExampleArrayInterface**

Bases: object

property shape

class bw_processing.examples.interfaces.**ExampleVectorInterface**

Bases: object

bw_processing.examples.parquet_files module

A basic example on how to use parquet files.

Module contents

Submodules

bw_processing.array_creation module

bw_processing.array_creation.**chunked**(*iterable*, *chunk_size*)

bw_processing.array_creation.**create_array**(*iterable*, *nrows=None*, *dtype=<class 'numpy.float32'>*)

Create a numpy array data `iterable`. Returns a filepath of a created file (if `filepath` is provided, or the array.

`iterable` can be data already in memory, or a generator.

`nrows` can be supplied, if known. If `iterable` has a length, it will be determined automatically. If `nrows` is not known, this function generates chunked arrays until `iterable` is exhausted, and concatenates them.

Either `nrows` or `ncols` must be specified.

bw_processing.array_creation.**create_chunked_array**(*iterable*, *ncols*, *dtype=<class 'numpy.float32'>*, *bucket_size=500*)

Create a numpy array from an iterable of indeterminate length.

Needed when we can't determine the length of the iterable ahead of time (e.g. for a generator or a database cursor), so can't create the complete array in memory in on step

Creates a list of arrays with `bucket_size` rows until `iterable` is exhausted, then concatenates them.

Parameters

- **iterable** – Iterable of data used to populate the array.
- **ncols** – Number of columns in the created array.
- **dtype** – Numpy dtype of the created array
- **bucket_size** – Number of rows in each intermediate array.

Returns:.

Returns the created array. Will return a zero-length array if `iterable` has no data.

bw_processing.array_creation.**create_chunked_structured_array**(*iterable*, *dtype*, *bucket_size=20000*)

Create a numpy structured array from an iterable of indeterminate length.

Needed when we can't determine the length of the iterable ahead of time (e.g. for a generator or a database cursor), so can't create the complete array in memory in on step

Creates a list of arrays with `bucket_size` rows until `iterable` is exhausted, then concatenates them.

Parameters

- **iterable** – Iterable of data used to populate the array.
- **dtype** – Numpy dtype of the created array
- **format_function** – If provided, this function will be called on each row of `iterable` before insertion in the array.
- **bucket_size** – Number of rows in each intermediate array.

Returns:.

Returns the created array. Will return a zero-length array if `iterable` has no data.

bw_processing.array_creation.**create_structured_array**(*iterable*, *dtype*, *nrows=None*, *sort=False*, *sort_fields=None*)

Create a numpy structured array for data `iterable`. Returns a filepath of a created file (if `filepath` is provided, or the array.

`iterable` can be data already in memory, or a generator.

`nrows` can be supplied, if known. If `iterable` has a length, it will be determined automatically. If `nrows` is not known, this function generates chunked arrays until `iterable` is exhausted, and concatenates them.

bw_processing.array_creation.**get_ncols**(*iterator*)

bw_processing.array_creation.**peek**(*iterator*)

bw_processing.constants module

class bw_processing.constants.**MatrixSerializeFormat**(*value*)

Bases: str, Enum

Enum with the serializing formats for the vectors and matrices.

NUMPY = 'numpy'

PARQUET = 'parquet'

bw_processing.datapackage module

class bw_processing.datapackage.**Datapackage**

Bases: *DatapackageBase*

Interface for creating, loading, and using numerical datapackages for Brightway.

Note that there are two entry points to using this class, both separate functions: `create_datapackage()` and `load_datapackage()`. Do not create an instance of the class with `Datapackage()`, unless you like playing with danger :)

Data packages can be stored in memory, in a directory, or in a zip file. When creating data packages for use later, don't forget to call `.finalize_serialization()`, or the metadata won't be written and the data package won't be usable.

Potential gotchas:

- There is currently no way to modify a zipped data package once it is finalized.
- Resources that are interfaces to external data sources (either in Python or other) can't be saved, but must be recreated each time a data package is used.

add_csv_metadata(*, *dataframe*, *valid_for*, *name=None*, ***kwargs*)

Add an iterable metadata object to be stored as a CSV file.

The purpose of storing metadata is to enable data exchange; therefore, this method assumes that data is written to disk.

The normal use case of this method is to link integer indices from either structured or presample arrays to a set of fields that uniquely identifies each object. This allows for matching based on object attributes from computer to computer, where database ids or other computer-generated codes might not be consistent.

Uses pandas to store and load data; therefore, metadata must already be a pandas dataframe.

In contrast with presamples arrays, `iterable_data_source` cannot be an infinite generator. We need a finite set of data to build a matrix.

In contrast to `self.create_structured_array`, this always stores the dataframe in `self.data`; no proxies are used.

Parameters

- **dataframe** (*) – Dataframe to be persisted to disk.

- **valid_for** (*) – List of resource names that this metadata is valid for; must be either structured or presample indices arrays. Each item in `valid_for` has the form `("resource_name", "rows" or "cols")`. `resource_name` should be either a structured or a presamples indices array.
- **name** (*) – The name of this resource. Names must be unique in a given data package
- **extra** (*) – Dict of extra metadata

Returns
Nothing, but appends objects to `self.metadata['resources']` and `self.data`.

Raises
- ***** **AssertionError** – If inputs are not in correct form
- ***** **AssertionError** – If `valid_for` refers to unavailable resources

Return type
None

add_dynamic_array(*, *matrix*, *interface*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

interface must support the presamples API.

Parameters
- **matrix** (*str*) –
- **interface** (*Any*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_dynamic_vector(*, *matrix*, *interface*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters
- **matrix** (*str*) –
- **interface** (*Any*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_json_metadata(*, *data*, *valid_for*, *name=None*, ***kwargs*)

Add an iterable metadata object to be stored as a JSON file.

The purpose of storing metadata is to enable data exchange; therefore, this method assumes that data is written to disk.

The normal use case of this method is to provide names and other metadata for parameters whose values are stored as presamples arrays. The length of `data` should match the number of rows in the corresponding presamples array, and `data` is just a list of string labels for the parameters. However, this method can also be used to store other metadata, e.g. for external data resources.

In contrast to `self.create_structured_array`, this always stores the dataframe in `self.data`; no proxies are used.

Parameters

- **data** (*) – Data to be persisted to disk.
- **valid_for** (*) – Name of structured data or presample array that this metadata is valid for.
- **name** (*) – The name of this resource. Names must be unique in a given data package
- **extra** (*) – Dict of extra metadata

Returns

Nothing, but appends objects to `self.metadata['resources']` and `self.data`.

Raises

- ***** **AssertionError** – If inputs are not in correct form
- ***** **AssertionError** – If `valid_for` refers to unavailable resources

Return type

None

add_persistent_array(*, *matrix*, *data_array*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters

- **matrix** (*str*) –
- **data_array** (*ndarray*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type

None

add_persistent_vector(*, *matrix*, *indices_array*, *name=None*, *data_array=None*, *flip_array=None*, *distributions_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters

- **matrix** (*str*) –

- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **data_array** (*ndarray | None*) –
- **flip_array** (*ndarray | None*) –
- **distributions_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_persistent_vector_from_iterator(*, *matrix=None*, *name=None*, *dict_iterator=None*, *nrows=None*, *matrix_serialize_format_type=None*, ***kwargs*)

Create a persistant vector from an iterator. Uses the utility function `resolve_dict_iterator`.

This is the **only array creation method which produces sorted arrays**.

Parameters

- **matrix** (*str | None*) –
- **name** (*str | None*) –
- **dict_iterator** (*Any | None*) –
- **nrows** (*int | None*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

finalize_serialization()

Return type
None

write_modified()

Write the data in modified files to the filesystem (if allowed).

class bw_processing.datapackage.**DatapackageBase**

Bases: ABC

Base class for datapackages. Not for normal use - you should use either *Datapackage* or *FilteredDatapackage*.

dehydrated_interfaces()

Return a list of the resource groups which have dehydrated interfaces

Return type
List[str]

del_resource(*name_or_index*)

Remove a resource, and delete its data file, if any.

Parameters
name_or_index (*str | int*) –

Return type
None

del_resource_group(*name*)

Remove a resource group, and delete its data files, if any.

Use `exclude_resource_group` if you want to keep the underlying resource in the filesystem.

Parameters
name (*str*) –

Return type
None

exclude(*filters*)

Filter a datapackage to exclude resources matching a filter.

Usage cases:

Filter out a given resource:

exclude_generic({"matrix': "some_label"})

Filter out a resource group with a given kind:

exclude_generic({"group': "some_group", "kind": "some_kind"})

Parameters
filters (*Dict[str, str]*) –

Return type
FilteredDatapackage

filter_by_attribute(*key*, *value*)

Create a new `FilteredDatapackage` which satisfies the filter `resource[key] == value`.

All included objects are the same as in the original data package, i.e. no copies are made. No checks are made to ensure consistency with modifications to the original datapackage after the creation of this filtered datapackage.

This method was introduced to allow for the efficient construction of matrices; each datapackage can have data for multiple matrices, and we can then create filtered datapackages which exclusively have data for the matrix of interest. As such, they should be considered read-only, though this is not enforced.

Parameters

- **key** (*str*) –
- **value** (*Any*) –

Return type
FilteredDatapackage

get_resource(*name_or_index*)

Return data and metadata for `name_or_index`.

Parameters
name_or_index (*) – Name (str) or index (int) of a resource in the existing metadata.

Raises

- ***IndexError** – Integer index out of range of given metadata
- ***ValueError** – String name not present in metadata
- ***NonUnique** – String name present in two resource metadata sections

Returns
(data object, metadata dict)

Return type
(*Any*, <class 'dict'>)

property groups: dict

Return a dictionary of `{group label: filtered datapackage}` in the same order as the group labels are first encountered in the datapackage metadata.

Ignores resources which don't have group labels.

rehydrate_interface(*name_or_index*, *resource*, *initialize_with_config=False*)

Substitute the undefined interface in this datapackage with the actual interface resource `resource`. Loading a datapackage with an interface loads an instance of `UndefinedInterface`, which should be substituted (rehydrated) with an actual interface instance.

If `initialize_with_config` is true, the `resource` is initialized (i.e. `resource(**config_data)`) with the resource data under the key `config`. If `config` is missing, a `KeyError` is raised.

`name_or_index` should be the data source name. If this value is a string and doesn't end with `.data`, `.data` is automatically added.

Parameters

- **name_or_index** (*str | int*) –
- **resource** (*Any*) –
- **initialize_with_config** (*bool*) –

Return type
None

property resources: list

class bw_processing.datapackage.**FilteredDatapackage**

Bases: *DatapackageBase*

A subset of a datapackage. Used in matrix construction or other data manipulation operations.

Should be treated as read-only.

bw_processing.datapackage.**create_datapackage**(*fs=None*, *name=None*, *id_=None*, *metadata=None*, *combinatorial=False*, *sequential=False*, *seed=None*, *sum_intra_duplicates=True*, *sum_inter_duplicates=False*, *matrix_serialize_format_type=MatrixSerializeFormat.NUMPY*)

Create a new data package.

All arguments are optional; if a PyFilesystem2 filesystem is not provided, a MemoryFS will be used.

All metadata elements should follow the datapackage specification.

Licenses are specified as a list in `metadata`. The default license is the Open Data Commons Public Domain Dedication and License v1.0.

Parameters

- **fs** (*) – A *Filesystem*, optional. A new *MemoryFS* is used if not provided.
- **name** (*) – *str*, optional. A new uuid is used if not provided.
- **str** (** id.*) –

- **provided.** (*optional. A new uuid is used if not*) –
- **dict** (** metadata.*) –
- **above.** (*optional. Metadata dictionary following datapackage specification; see*) –
- **bool** (** sum_inter_duplicates.*) – Policy on how to sample columns across multiple data arrays; see readme.
- **`False.** (*default*) – Policy on how to sample columns across multiple data arrays; see readme.
- **bool** – Policy on how to sample columns in data arrays; see readme.
- **`False.** – Policy on how to sample columns in data arrays; see readme.
- **int** (** seed.*) –
- **generator.** (*optional. Seed to use in random number*) –
- **bool** –
- **together** (*default False. Should duplicate elements in across data resources be summed*) –
- **values.** (*or should the last value replace previous*) –
- **bool** –
- **together** –
- **package.** (*or should the last value replace previous values. Order of data resources is given by the order they are added to the data*) –
- **MatrixSerializeFormat** (** matrix_serialize_format_type.*) –
- **type.** (*default MatrixSerializeFormat.NUMPY. Matrix serialization format*) –
- **id_** (*str | None*) –
- **metadata** (*dict | None*) –
- **combinatorial** (*bool*) –
- **sequential** (*bool*) –
- **seed** (*int | None*) –
- **sum_intra_duplicates** (*bool*) –
- **sum_inter_duplicates** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat) –

Returns
A *Datapackage* instance.

Return type
Datapackage

bw_processing.datapackage.**load_datapackage**(*fs_or_obj*, *mmap_mode=None*, *proxy=False*)

Load an existing datapackage.

Can load proxies to data instead of the data itself, which can be useful when interacting with large arrays or large packages where only a subset of the data will be accessed.

Proxies use something similar to *functools.partial* to create a callable class instead of returning the raw data (see https://github.com/brightway-lca/bw_processing/issues/9 for why we can't just use *partial*). datapackage access methods (i.e. *.get_resource*) will automatically resolve proxies when needed.

Parameters

- **DatapackageBase.** (** fs_or_obj. A Filesystem or an instance of*) –
- **str** (** mmap_mode.*) –
- **arrays.** (*optional. Define memory mapping mode to use when loading Numpy*) –
- **bool** (** proxy.*) –
- **above.** (*default False. Load proxies instead of complete Numpy arrays; see*) –
- **fs_or_obj** (*DatapackageBase | FS*) –
- **mmap_mode** (*str | None*) –
- **proxy** (*bool*) –

Returns
A *Datapackage* instance.

Return type
Datapackage

bw_processing.datapackage.**simple_graph**(*data*, *fs=None*, ***metadata*)

Easy creation of simple datapackages with only persistent vectors.

`data` is a dictionary with the form:

..code-block:: python

matrix_name (str): [
(row id (int), col id (int), value (float), flip (bool, default False))

]

`fs` is a filesystem.

`metadata` are passed as kwargs to `create_datapackage()`.

Returns the datapackage.

Parameters

- **data** (*dict*) –
- **fs** (*FS | None*) –

bw_processing.errors module

exception bw_processing.errors.**BrightwayProcessingError**

Bases: `Exception`

exception bw_processing.errors.**Closed**

Bases: *BrightwayProcessingError*

Datapackage closed, can't be written to anymore.

exception bw_processing.errors.**FileIntegrityError**

Bases: *BrightwayProcessingError*

MD5 hash does not agree with file contents

exception bw_processing.errors.**InconsistentFields**

Bases: *BrightwayProcessingError*

Given fields not the same for each element

exception bw_processing.errors.**InvalidMimetype**

Bases: *BrightwayProcessingError*

Provided mimetype missing or not understood

exception bw_processing.errors.**InvalidName**

Bases: *BrightwayProcessingError*

Name fails datapackage requirements:

A short url-usable (and preferably human-readable) name of the package. This MUST be lower-case and contain only alphanumeric characters along with ".", "_" or "-" characters.

exception bw_processing.errors.**LengthMismatch**

Bases: *BrightwayProcessingError*

Number of resources doesn't match the number of data objects

exception bw_processing.errors.**NonUnique**

Bases: *BrightwayProcessingError*

Nonunique elements when uniqueness is required

exception bw_processing.errors.**PotentialInconsistency**

Bases: *BrightwayProcessingError*

Given operation could cause inconsistent data

exception bw_processing.errors.**ShapeMismatch**

Bases: *BrightwayProcessingError*

Array shapes in a resource group are not consistent

exception bw_processing.errors.**WrongDatatype**

Bases: *BrightwayProcessingError*

Wrong type of data written to a resource

bw_processing.filesystem module

bw_processing.filesystem.**clean_datapackage_name**(*name*)

Clean string `name` of characters not allowed in data package names.

Replaces with underscores, and drops multiple underscores.

Parameters
name (*str*) –

Return type
str

bw_processing.filesystem.**md5**(*filepath*, *blocksize=65536*)

Generate MD5 hash for file at *filepath*

Parameters

- **filepath** (*str | Path*) –
- **blocksize** (*int*) –

Return type
str

bw_processing.filesystem.**safe_filename**(*string*, *add_hash=True*, *full=False*)

Convert arbitrary strings to make them safe for filenames. Substitutes strange characters, and uses unicode normalization.

if *add_hash*, appends hash of *string* to avoid name collisions.

From http://stackoverflow.com/questions/295135/turn-a-string-into-a-valid-filename-in-python

Parameters

- **string** (*str | bytes*) –
- **add_hash** (*bool*) –
- **full** (*bool*) –

Return type
str

bw_processing.indexing module

bw_processing.indexing.**reindex**(*datapackage*, *metadata_name*, *data_iterable*, *fields=None*, *id_field_datapackage='id'*, *id_field_destination='id'*)

Use the metadata to set the integer indices in `datapackage` to those used in `data_iterable`.

Used in data exchange. Often, the integer ids provided in the data package are arbitrary, and need to be mapped to the values present in your database.

Updates the datapackage in place.

Parameters

- **datapackage** (*) – datapackage of *Filesystem*. Input to *load_datapackage* function.
- **metadata_name** (*) – Name identifying a CSV metadata resource in `datapackage`
- **data_iterable** (*) – Iterable which returns objects that support `.get()`.
- **fields** (*) – Optional list of fields to use while matching
- **id_field_datapackage** (*) – String identifying the column providing an integer id in the datapackage
- **id_field_destination** (*) – String identifying the column providing an integer id in `data_iterable`

Raises

- ** **KeyError** – `data_iterable` is missing `id_field_destination` field
- ** **KeyError** – `metadata_name` is missing `id_field_datapackage` field

- * **NonUnique** – Multiple objects found in `data_iterable` which matches fields in `datapackage`
- * **KeyError** – `metadata_name` is not in `datapackage`
- * **KeyError** – No object found in `data_iterable` which matches fields in `datapackage`
- * **ValueError** – `metadata_name` is not CSV metadata.
- * **ValueError** – The resources given for `metadata_name` are not present in this `datapackage`
- * **AttributeError** – `data_iterable` doesn't support field retrieval using `.get()`.

Returns
Datapackage instance with modified data

Return type
None

`bw_processing.indexing.`**reset_index**(*datapackage*, *metadata_name*)

Reset the numerical indices in `datapackage` to sequential integers starting from zero.

Updates the datapackage in place.

Parameters

- **datapackage** (*) – datapackage or *Filesystem*. Input to *load_datapackage* function.
- **metadata_name** (*) – Name identifying a CSV metadata resource in `datapackage`

Returns
Datapackage instance with modified data

Return type
Datapackage

bw_processing.io_helpers module

`bw_processing.io_helpers.`**file_reader**(*, *fs*, *resource*, *mimetype*, *proxy=False*, *mmap_mode=None*, ***kwargs*)

Parameters

- **fs** (*FS*) –
- **resource** (*str*) –
- **mimetype** (*str*) –
- **proxy** (*bool*) –
- **mmap_mode** (*str | None*) –

Return type
Any

`bw_processing.io_helpers.`**file_writer**(*, *data*, *fs*, *resource*, *mimetype*, *matrix_serialize_format_type=MatrixSerializeFormat.NUMPY*, *meta_object=None*, *meta_type=None*, ***kwargs*)

Parameters

- **data** (*Any*) –

- **fs** (*FS*) –
- **resource** (*str*) –
- **mimetype** (*str*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat) –
- **meta_object** (*str | None*) –
- **meta_type** (*str | None*) –

Return type
None

bw_processing.io_helpers.**generic_directory_filesystem**(*, *dirpath*)

Parameters
dirpath (*Path*) –

Return type
OSFS

bw_processing.io_helpers.**generic_zipfile_filesystem**(*, *dirpath*, *filename*, *write=True*)

Parameters

- **dirpath** (*Path*) –
- **filename** (*str*) –
- **write** (*bool*) –

Return type
ZipFS

bw_processing.io_parquet_helpers module

bw_processing.io_pyarrow_helpers module

bw_processing.merging module

bw_processing.merging.**add_resource_suffix**(*metadata*, *suffix*)

Update the `name`, `path`, and `group` values to include `suffix`. The suffix comes after the basename but after the data type suffix (e.g. indices, data).

Given the suffix _foo" and the metadata:

{
"name": "sa-data-vector-from-dict.indices", "path": "sa-data-vector-from-dict.indices.npy", "group": "sa-data-vector-from-dict",
}

Returns

{
"name": "sa-data-vector-from-dict_foo.indices", "path": "sa-data-vector-from-dict_foo.indices.npy", "group": "sa-data-vector-from-dict_foo",
}

Parameters

- **metadata** (*dict*) –
- **suffix** (*str*) –

Return type
dict

bw_processing.merging.**mask_resource**(*obj*, *mask*)

Parameters

- **obj** (*Any*) –
- **mask** (*ndarray*) –

Return type
Any

bw_processing.merging.**merge_datapackages_with_mask**(*first_dp*, *first_resource_group_label*, *second_dp*, *second_resource_group_label*, *mask_array*, *output_fs=None*, *metadata=None*)

Merge two resources using a Numpy boolean mask. Returns elements from `first_dp` where the mask is `True`, otherwise `second_dp`.

Both resource arrays, and the filter mask, must have the same length.

Both datapackages must be static, i.e. not interfaces. This is because we don't yet have the functionality to select only some of the values in a resource group in `matrix_utils`.

This function currently **will not** mask or filter JSON or CSV metadata.

Parameters

- **first_dp** (*) – The datapackage from whom values will be taken when `mask_array` is `True`.
- **first_resource_group_label** (*) – Label of the resource group in `first_dp` to select values from.
- **second_dp** (*) – The datapackage from whom values will be taken when `mask_array` is `False`.
- **second_resource_group_label** (*) – Label of the resource group in `second_dp` to select values from.
- **mask_array** (*) – Boolean numpy array
- **output_fs** (*) – Filesystem to write new datapackage to, if any.
- **metadata** (*) – Metadata for new datapackage, if any.

Returns
A *Datapackage* instance. Will write the resulting datapackage to `output_fs` if provided.

Return type
DatapackageBase

bw_processing.merging.**update_nrows**(*resource*, *data*)

Parameters

- **resource** (*dict*) –
- **data** (*Any*) –

Return type
dict

bw_processing.merging.**write_data_to_fs**(*resource*, *data*, *fs*)

Parameters

- **resource** (*dict*) –
- **data** (*Any*) –
- **fs** (*FS*) –

Return type
None

bw_processing.proxies module

class bw_processing.proxies.**Proxy**(*func*, *label*, *kwargs*)

Bases: object

class bw_processing.proxies.**UndefinedInterface**

Bases: object

An interface to external data that isn't saved to disk.

bw_processing.unique_fields module

bw_processing.unique_fields.**as_unique_attributes**(*data*, *exclude=None*, *include=None*, *raise_error=False*)

Format `data` as unique set of attributes and values for use in `create_processed_datapackage`.

Each element in `data` must have the attribute `id`, and it must be unique. However, the field "id" is not used in selecting the unique set of attributes.

If no set of attributes is found that uniquely identifies all features is found, all fields are used. To have this case raise an error, pass `raise_error=True`.

data = [
{},

]

Parameters

- **data** (*iterable*) – List of dictionaries with the same fields.
- **exclude** (*iterable*) – Fields to exclude during search for uniqueness. `id` is Always excluded.
- **include** (*iterable*) – Fields to include when returning, even if not unique

Returns
(list of field names as strings, dictionary of data ids to values for given field names)

Raises
InconsistentFields – Not all features provides all fields.

bw_processing.unique_fields.**as_unique_attributes_dataframe**(*df*, *exclude=None*, *include=None*, *raise_error=False*)

bw_processing.unique_fields.**greedy_set_cover**(*data*, *exclude=None*, *raise_error=True*)

Find unique set of attributes that uniquely identifies each element in `data`.

Feature selection is a well known problem, and is analogous to the set cover problem, for which there is a well known heuristic.

Example

data = [
{'a': 1, 'b': 2, 'c': 3}, {'a': 2, 'b': 2, 'c': 3}, {'a': 1, 'b': 2, 'c': 4},

] greedy_set_cover(data) >>> {'a', 'c'}

Parameters

- **data** (*iterable*) – List of dictionaries with the same fields.
- **exclude** (*iterable*) – Fields to exclude during search for uniqueness. `id` is Always excluded.

Returns
Set of attributes (strings)

Raises
NonUnique – The given fields are not enough to ensure uniqueness.

Note that `NonUnique` is not raised if `raise_error` is false.

bw_processing.utils module

bw_processing.utils.**as_uncertainty_type**(*row*)

Parameters
row (*dict*) –

Return type
int

bw_processing.utils.**check_name**(*name*)

Parameters
name (*str*) –

Return type
None

bw_processing.utils.**check_suffix**(*path*, *suffix=<class 'str'>*)

Add `suffix`, if not already in `path`.

Parameters
path (*str | Path*) –

Return type
str

bw_processing.utils.**dictionary_formatter**(*row*)

Format processed array row from dictionary input

Parameters
row (*dict*) –

Return type
tuple

bw_processing.utils.**load_bytes**(*obj*)

Parameters
obj (*Any*) –

Return type
Any

bw_processing.utils.**resolve_dict_iterator**(*iterator*, *nrows=None*)

Note that this function produces sorted arrays.

Parameters

- **iterator** (*Any*) –
- **nrows** (*int | None*) –

Return type
tuple

Module contents

class bw_processing.**Datapackage**

Bases: *DatapackageBase*

Interface for creating, loading, and using numerical datapackages for Brightway.

Note that there are two entry points to using this class, both separate functions: `create_datapackage()` and `load_datapackage()`. Do not create an instance of the class with `Datapackage()`, unless you like playing with danger :)

Data packages can be stored in memory, in a directory, or in a zip file. When creating data packages for use later, don't forget to call `.finalize_serialization()`, or the metadata won't be written and the data package won't be usable.

Potential gotchas:

- There is currently no way to modify a zipped data package once it is finalized.
- Resources that are interfaces to external data sources (either in Python or other) can't be saved, but must be recreated each time a data package is used.

add_csv_metadata(*, *dataframe*, *valid_for*, *name=None*, ***kwargs*)

Add an iterable metadata object to be stored as a CSV file.

The purpose of storing metadata is to enable data exchange; therefore, this method assumes that data is written to disk.

The normal use case of this method is to link integer indices from either structured or presample arrays to a set of fields that uniquely identifies each object. This allows for matching based on object attributes from computer to computer, where database ids or other computer-generated codes might not be consistent.

Uses pandas to store and load data; therefore, metadata must already be a pandas dataframe.

In contrast with presamples arrays, `iterable_data_source` cannot be an infinite generator. We need a finite set of data to build a matrix.

In contrast to `self.create_structured_array`, this always stores the dataframe in `self.data`; no proxies are used.

Parameters

- **dataframe** (*) – Dataframe to be persisted to disk.
- **valid_for** (*) – List of resource names that this metadata is valid for; must be either structured or presample indices arrays. Each item in `valid_for` has the form `("resource_name", "rows" or "cols")`. `resource_name` should be either a structured or a presamples indices array.
- **name** (*) – The name of this resource. Names must be unique in a given data package
- **extra** (*) – Dict of extra metadata

Returns
Nothing, but appends objects to `self.metadata['resources']` and `self.data`.

Raises

- ** AssertionError – If inputs are not in correct form
- ** AssertionError – If `valid_for` refers to unavailable resources

Return type
None

add_dynamic_array(*, *matrix*, *interface*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

interface must support the presamples API.

Parameters

- **matrix** (*str*) –
- **interface** (*Any*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | None) –

Return type
None

add_dynamic_vector(*, *matrix*, *interface*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters

- **matrix** (*str*) –
- **interface** (*Any*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –

- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_json_metadata(*, *data*, *valid_for*, *name=None*, ***kwargs*)

Add an iterable metadata object to be stored as a JSON file.

The purpose of storing metadata is to enable data exchange; therefore, this method assumes that data is written to disk.

The normal use case of this method is to provide names and other metadata for parameters whose values are stored as presamples arrays. The length of `data` should match the number of rows in the corresponding presamples array, and `data` is just a list of string labels for the parameters. However, this method can also be used to store other metadata, e.g. for external data resources.

In contrast to `self.create_structured_array`, this always stores the dataframe in `self.data`; no proxies are used.

Parameters

- **data** (*) – Data to be persisted to disk.
- **valid_for** (*) – Name of structured data or presample array that this metadata is valid for.
- **name** (*) – The name of this resource. Names must be unique in a given data package
- **extra** (*) – Dict of extra metadata

Returns
Nothing, but appends objects to `self.metadata['resources']` and `self.data`.

Raises

- * **AssertionError** – If inputs are not in correct form
- * **AssertionError** – If `valid_for` refers to unavailable resources

Return type
None

add_persistent_array(*, *matrix*, *data_array*, *indices_array*, *name=None*, *flip_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters

- **matrix** (*str*) –
- **data_array** (*ndarray*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **flip_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_persistent_vector(*, *matrix*, *indices_array*, *name=None*, *data_array=None*, *flip_array=None*, *distributions_array=None*, *keep_proxy=False*, *matrix_serialize_format_type=None*, ***kwargs*)

Parameters

- **matrix** (*str*) –
- **indices_array** (*ndarray*) –
- **name** (*str | None*) –
- **data_array** (*ndarray | None*) –
- **flip_array** (*ndarray | None*) –
- **distributions_array** (*ndarray | None*) –
- **keep_proxy** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

add_persistent_vector_from_iterator(*, *matrix=None*, *name=None*, *dict_iterator=None*, *nrows=None*, *matrix_serialize_format_type=None*, ***kwargs*)

Create a persistant vector from an iterator. Uses the utility function `resolve_dict_iterator`.

This is the **only array creation method which produces sorted arrays**.

Parameters

- **matrix** (*str | None*) –
- **name** (*str | None*) –
- **dict_iterator** (*Any | None*) –
- **nrows** (*int | None*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat | *None*) –

Return type
None

finalize_serialization()

Return type
None

write_modified()

Write the data in modified files to the filesystem (if allowed).

class bw_processing.**DatapackageBase**

Bases: ABC

Base class for datapackages. Not for normal use - you should use either *Datapackage* or *FilteredDatapackage*.

dehydrated_interfaces()

Return a list of the resource groups which have dehydrated interfaces

Return type
List[str]

del_resource(*name_or_index*)

Remove a resource, and delete its data file, if any.

Parameters
name_or_index (*str | int*) –

Return type
None

del_resource_group(*name*)

Remove a resource group, and delete its data files, if any.

Use `exclude_resource_group` if you want to keep the underlying resource in the filesystem.

Parameters
name (*str*) –

Return type
None

exclude(*filters*)

Filter a datapackage to exclude resources matching a filter.

Usage cases:

Filter out a given resource:

exclude_generic({"matrix': "some_label"})

Filter out a resource group with a given kind:

exclude_generic({"group': "some_group", "kind": "some_kind"})

Parameters
filters (*Dict[str, str]*) –

Return type
FilteredDatapackage

filter_by_attribute(*key*, *value*)

Create a new `FilteredDatapackage` which satisfies the filter `resource[key] == value`.

All included objects are the same as in the original data package, i.e. no copies are made. No checks are made to ensure consistency with modifications to the original datapackage after the creation of this filtered datapackage.

This method was introduced to allow for the efficient construction of matrices; each datapackage can have data for multiple matrices, and we can then create filtered datapackages which exclusively have data for the matrix of interest. As such, they should be considered read-only, though this is not enforced.

Parameters

- **key** (*str*) –
- **value** (*Any*) –

Return type
FilteredDatapackage

get_resource(*name_or_index*)

Return data and metadata for `name_or_index`.

Parameters
name_or_index (*) – Name (str) or index (int) of a resource in the existing metadata.

Raises
- * **IndexError** – Integer index out of range of given metadata
- * **ValueError** – String name not present in metadata
- * **NonUnique** – String name present in two resource metadata sections

Returns
(data object, metadata dict)

Return type
(*Any*, <class 'dict'>)

property groups: dict

Return a dictionary of `{group label: filtered datapackage}` in the same order as the group labels are first encountered in the datapackage metadata.

Ignores resources which don't have group labels.

rehydrate_interface(*name_or_index*, *resource*, *initialize_with_config=False*)

Substitute the undefined interface in this datapackage with the actual interface resource `resource`. Loading a datapackage with an interface loads an instance of `UndefinedInterface`, which should be substituted (rehydrated) with an actual interface instance.

If `initialize_with_config` is true, the `resource` is initialized (i.e. `resource(**config_data)`) with the resource data under the key `config`. If `config` is missing, a `KeyError` is raised.

`name_or_index` should be the data source name. If this value is a string and doesn't end with `.data`, `.data` is automatically added.

Parameters
- **name_or_index** (*str | int*) –
- **resource** (*Any*) –
- **initialize_with_config** (*bool*) –

Return type
None

property resources: list

class bw_processing.**FilteredDatapackage**

Bases: *DatapackageBase*

A subset of a datapackage. Used in matrix construction or other data manipulation operations.

Should be treated as read-only.

class bw_processing.**MatrixSerializeFormat**(*value*)

Bases: `str`, `Enum`

Enum with the serializing formats for the vectors and matrices.

NUMPY = 'numpy'

PARQUET = 'parquet'

class bw_processing.**UndefinedInterface**

Bases: object

An interface to external data that isn't saved to disk.

bw_processing.**as_unique_attributes**(*data*, *exclude=None*, *include=None*, *raise_error=False*)

Format `data` as unique set of attributes and values for use in `create_processed_datapackage`.

Each element in `data` must have the attribute `id`, and it must be unique. However, the field "id" is not used in selecting the unique set of attributes.

If no set of attributes is found that uniquely identifies all features is found, all fields are used. To have this case raise an error, pass `raise_error=True`.

data = [
{},

]

Parameters

- **data** (*iterable*) – List of dictionaries with the same fields.
- **exclude** (*iterable*) – Fields to exclude during search for uniqueness. `id` is Always excluded.
- **include** (*iterable*) – Fields to include when returning, even if not unique

Returns
(list of field names as strings, dictionary of data ids to values for given field names)

Raises
InconsistentFields – Not all features provides all fields.

bw_processing.**as_unique_attributes_dataframe**(*df*, *exclude=None*, *include=None*, *raise_error=False*)

bw_processing.**clean_datapackage_name**(*name*)

Clean string `name` of characters not allowed in data package names.

Replaces with underscores, and drops multiple underscores.

Parameters
name (*str*) –

Return type
str

bw_processing.**create_array**(*iterable*, *nrows=None*, *dtype=<class 'numpy.float32'>*)

Create a numpy array data `iterable`. Returns a filepath of a created file (if `filepath` is provided, or the array.

`iterable` can be data already in memory, or a generator.

`nrows` can be supplied, if known. If `iterable` has a length, it will be determined automatically. If `nrows` is not known, this function generates chunked arrays until `iterable` is exhausted, and concatenates them.

Either `nrows` or `ncols` must be specified.

bw_processing.**create_datapackage**(*fs=None*, *name=None*, *id_=None*, *metadata=None*, *combinatorial=False*, *sequential=False*, *seed=None*, *sum_intra_duplicates=True*, *sum_inter_duplicates=False*, *matrix_serialize_format_type=MatrixSerializeFormat.NUMPY*)

Create a new data package.

All arguments are optional; if a PyFilesystem2 filesystem is not provided, a MemoryFS will be used.

All metadata elements should follow the datapackage specification.

Licenses are specified as a list in `metadata`. The default license is the Open Data Commons Public Domain Dedication and License v1.0.

Parameters

- **fs** (*) – A *Filesystem*, optional. A new *MemoryFS* is used if not provided.
- **name** (*) – *str*, optional. A new uuid is used if not provided.
- **str** (** id.*) –
- **provided.** (*optional. A new uuid is used if not*) –
- **dict** (** metadata.*) –
- **above.** (*optional. Metadata dictionary following datapackage specification; see*) –
- **bool** (** sum_inter_duplicates.*) – Policy on how to sample columns across multiple data arrays; see readme.
- **`False.** (*default*) – Policy on how to sample columns across multiple data arrays; see readme.
- **bool** – Policy on how to sample columns in data arrays; see readme.
- **`False.** – Policy on how to sample columns in data arrays; see readme.
- **int** (** seed.*) –
- **generator.** (*optional. Seed to use in random number*) –
- **bool** –
- **together** (*default False. Should duplicate elements in across data resources be summed*) –
- **values.** (*or should the last value replace previous*) –
- **bool** –
- **together** –
- **package.** (*or should the last value replace previous values. Order of data resources is given by the order they are added to the data*) –
- **MatrixSerializeFormat** (** matrix_serialize_format_type.*) –
- **type.** (*default MatrixSerializeFormat.NUMPY. Matrix serialization format*) –
- **id_** (*str | None*) –
- **metadata** (*dict | None*) –
- **combinatorial** (*bool*) –
- **sequential** (*bool*) –
- **seed** (*int | None*) –
- **sum_intra_duplicates** (*bool*) –

- **sum_inter_duplicates** (*bool*) –
- **matrix_serialize_format_type** (MatrixSerializeFormat) –

Returns
A *Datapackage* instance.

Return type
Datapackage

bw_processing.**create_structured_array**(*iterable*, *dtype*, *nrows=None*, *sort=False*, *sort_fields=None*)

Create a numpy structured array for data `iterable`. Returns a filepath of a created file (if `filepath` is provided, or the array.

`iterable` can be data already in memory, or a generator.

`nrows` can be supplied, if known. If `iterable` has a length, it will be determined automatically. If `nrows` is not known, this function generates chunked arrays until `iterable` is exhausted, and concatenates them.

bw_processing.**generic_directory_filesystem**(*, *dirpath*)

Parameters
dirpath (*Path*) –

Return type
OSFS

bw_processing.**generic_zipfile_filesystem**(*, *dirpath*, *filename*, *write=True*)

Parameters

- **dirpath** (*Path*) –
- **filename** (*str*) –
- **write** (*bool*) –

Return type
ZipFS

bw_processing.**load_datapackage**(*fs_or_obj*, *mmap_mode=None*, *proxy=False*)

Load an existing datapackage.

Can load proxies to data instead of the data itself, which can be useful when interacting with large arrays or large packages where only a subset of the data will be accessed.

Proxies use something similar to *functools.partial* to create a callable class instead of returning the raw data (see https://github.com/brightway-lca/bw_processing/issues/9 for why we can't just use *partial*). datapackage access methods (i.e. *.get_resource*) will automatically resolve proxies when needed.

Parameters

- **DatapackageBase.** (** fs_or_obj. A Filesystem or an instance of*) –
- **str** (** mmap_mode.*) –
- **arrays.** (*optional. Define memory mapping mode to use when loading Numpy*) –
- **bool** (** proxy.*) –
- **above.** (*default False. Load proxies instead of complete Numpy arrays; see*) –
- **fs_or_obj** (DatapackageBase | *FS*) –

- **mmap_mode** (*str | None*) –
- **proxy** (*bool*) –

Returns
A *Datapackage* instance.

Return type
Datapackage

bw_processing.**md5**(*filepath*, *blocksize=65536*)

Generate MD5 hash for file at *filepath*

Parameters

- **filepath** (*str | Path*) –
- **blocksize** (*int*) –

Return type
str

bw_processing.**merge_datapackages_with_mask**(*first_dp*, *first_resource_group_label*, *second_dp*, *second_resource_group_label*, *mask_array*, *output_fs=None*, *metadata=None*)

Merge two resources using a Numpy boolean mask. Returns elements from `first_dp` where the mask is `True`, otherwise `second_dp`.

Both resource arrays, and the filter mask, must have the same length.

Both datapackages must be static, i.e. not interfaces. This is because we don't yet have the functionality to select only some of the values in a resource group in `matrix_utils`.

This function currently **will not** mask or filter JSON or CSV metadata.

Parameters

- **first_dp** (*) – The datapackage from whom values will be taken when `mask_array` is `True`.
- **first_resource_group_label** (*) – Label of the resource group in `first_dp` to select values from.
- **second_dp** (*) – The datapackage from whom values will be taken when `mask_array` is `False`.
- **second_resource_group_label** (*) – Label of the resource group in `second_dp` to select values from.
- **mask_array** (*) – Boolean numpy array
- **output_fs** (*) – Filesystem to write new datapackage to, if any.
- **metadata** (*) – Metadata for new datapackage, if any.

Returns
A *Datapackage* instance. Will write the resulting datapackage to `output_fs` if provided.

Return type
DatapackageBase

bw_processing.**reindex**(*datapackage*, *metadata_name*, *data_iterable*, *fields=None*, *id_field_datapackage='id'*, *id_field_destination='id'*)

Use the metadata to set the integer indices in `datapackage` to those used in `data_iterable`.

Used in data exchange. Often, the integer ids provided in the data package are arbitrary, and need to be mapped to the values present in your database.

Updates the datapackage in place.

Parameters

- **datapackage** (*) – datapackage of *Filesystem*. Input to *load_datapackage* function.
- **metadata_name** (*) – Name identifying a CSV metadata resource in `datapackage`
- **data_iterable** (*) – Iterable which returns objects that support `.get()`.
- **fields** (*) – Optional list of fields to use while matching
- **id_field_datapackage** (*) – String identifying the column providing an integer id in the datapackage
- **id_field_destination** (*) – String identifying the column providing an integer id in `data_iterable`

Raises

- * **KeyError** – `data_iterable` is missing `id_field_destination` field
- * **KeyError** – `metadata_name` is missing `id_field_datapackage` field
- * **NonUnique** – Multiple objects found in `data_iterable` which matches fields in `datapackage`
- * **KeyError** – `metadata_name` is not in `datapackage`
- * **KeyError** – No object found in `data_iterable` which matches fields in `datapackage`
- * **ValueError** – `metadata_name` is not CSV metadata.
- * **ValueError** – The resources given for `metadata_name` are not present in this `datapackage`
- * **AttributeError** – `data_iterable` doesn't support field retrieval using `.get()`.

Returns
Datapackage instance with modified data

Return type
None

bw_processing.**reset_index**(*datapackage*, *metadata_name*)

Reset the numerical indices in `datapackage` to sequential integers starting from zero.

Updates the datapackage in place.

Parameters

- **datapackage** (*) – datapackage or *Filesystem*. Input to *load_datapackage* function.
- **metadata_name** (*) – Name identifying a CSV metadata resource in `datapackage`

Returns
Datapackage instance with modified data

Return type
Datapackage

bw_processing.**safe_filename**(*string*, *add_hash=True*, *full=False*)

Convert arbitrary strings to make them safe for filenames. Substitutes strange characters, and uses unicode normalization.

if *add_hash*, appends hash of *string* to avoid name collisions.

From http://stackoverflow.com/questions/295135/turn-a-string-into-a-valid-filename-in-python

Parameters

- **string** (*str | bytes*) –
- **add_hash** (*bool*) –
- **full** (*bool*) –

Return type
str

bw_processing.**simple_graph**(*data*, *fs=None*, ***metadata*)

Easy creation of simple datapackages with only persistent vectors.

`data` is a dictionary with the form:

..code-block:: python

matrix_name (str): [
(row id (int), col id (int), value (float), flip (bool, default False))

]

`fs` is a filesystem.

`metadata` are passed as kwargs to `create_datapackage()`.

Returns the datapackage.

Parameters

- **data** (*dict*) –
- **fs** (*FS | None*) –

8.2 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the BSD 3 Clause and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

8.2.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.2.2 How to request a feature

Request features on the Issue Tracker.

8.2.3 How to set up your development environment

Install the package with development requirements:

```
$ pip install -e ".[dev]"
```

8.2.4 How to build the documentation locally

Make sure you have installed the `dev` and `docs` extras of the package.

```
$ pip install -e ".[dev,docs]"
```

Build the documentation providing the `docs` directory at the root of the project as the source and specifying the output directory.

```
# use docs as source and docs/_build as output
sphinx-build docs docs/_build
```

8.2.5 How to test the project

1. Install the package with development requirements:

```
$ pip install -e ".[testing]"
```

2. Run the full test suite:

```
$ pytest
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

8.2.6 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The test suite must pass without errors and warnings.
- Include unit tests.
- If your changes add functionality, update the documentation accordingly.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running one of following commands, depending on your dependencies manager:

```
# conda or mamba
$ conda install pre-commit
```

or

```
$ pip install pre-commit
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.3 Contributor Covenant Code of Conduct

8.3.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

8.3.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.3.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

8.3.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

8.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at cmutel@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

8.3.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

8.4 License

```
Copyright 2023 Chris Mutel

Redistribution and use in source and binary forms, with or without modification, are␣
↪permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of␣
↪conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list␣
↪of conditions and the following disclaimer in the documentation and/or other materials␣
↪provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be␣
↪used to endorse or promote products derived from this software without specific prior␣
↪written permission.
```

(continues on next page)

(continued from previous page)

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY␣
↪EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES␣
↪OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT␣
↪SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,␣
↪INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED␣
↪TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR␣
↪BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN␣
↪CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY␣
↪WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

PYTHON MODULE INDEX

b

INDEX

D

E

F

G

I

L

M

N

P

R

S

U

W