Python-dwca-reader Documentation

Release 0.14.0

Nicolas Noé

Sep 09, 2020

CONTENTS

CHAPTER

ONE

WHAT IS PYTHON-DWCA-READER?

A simple Python package to read and parse Darwin Core Archive (DwC-A) files, as produced by the GBIF website, the IPT and many other biodiversity informatics tools.

It intends to be Pythonic and simple to use.

Archives can be enclosed in either a directory or a zip/tgz archive.

It supports most common features from the Darwin Core Archive standard, including extensions and Simple Darwin Core expressed as text (aka Archives consisting of a single CSV data file, possibly with Metadata but without Metafile).

It officially supports Python 3.5+ and has been reported to work on Jython by at least one user. It works on Linux, Mac OS and since v0.10.2 also on Windows. Use version 0.13.2 if you need Python 2.7 support.

CHAPTER

TWO

STATUS

It is currently considered beta quality. It helped many users across the world, but the API is still slightly moving (for the better!)

Concerning performances, it has been reported to work fine with 50Gb archives.

CHAPTER

THREE

MAJOR LIMITATIONS

- It doesn't currently fully implement the Darwin Core Archive standard, but focus on the most common/useful features. Don't hesitate to report any incompatible DwC-A on the GitHub repository, and we'll do our best to support it.
- No write support.

3.1 Installation

Quite simply:

```
$ pip install python-dwca-reader
```

3.2 Tutorial

3.2.1 Example uses

Basic use, access to metadata and data from the Core file

```
from dwca.read import DwCAReader
from dwca.darwincore.utils import qualname as qn

# Let's open our archive...
# Using the with statement ensure that resources will be properly freed/cleaned after␣
↪use.
with DwCAReader('my-archive.zip') as dwca:
    # We can now interact with the 'dwca' object

    # We can read scientific metadata (EML) through a xml.etree.ElementTree.Element␣
↪object in the 'metadata'
    # attribute.
    dwca.metadata

    # The 'descriptor' attribute gives access to the Archive Descriptor (meta.xml)␣
↪and allow
    # inspecting the archive:
    # For example, discover what the type the Core file is: (Occurrence, Taxon, ...)
    print("Core type is: %s" % dwca.descriptor.core.type)
    # => Core type is: http://rs.tdwg.org/dwc/terms/Occurrence
```

(continues on next page)

(continued from previous page)

```
    # Check if a Darwin Core term in present in the core file
    if 'http://rs.tdwg.org/dwc/terms/locality' in dwca.descriptor.core.terms:
        print("This archive contains the 'locality' term in its core file.")
    else:
        print("Locality term is not present.")

    # Using full qualnames for DarwincCore terms (such as 'http://rs.tdwg.org/dwc/
↪terms/country') is verbose...
    # The qualname() helper function make life easy for common terms.
    # (here, it has been imported as 'qn'):
    qn('locality')
    # => u'http://rs.tdwg.org/dwc/terms/locality'

    # Combined with previous examples, this can be used to things more clear:
    # For example:
    if qn('locality') in dwca.descriptor.core.terms:
        pass

    # Or:
    if dwca.descriptor.core.type == qn('Occurrence'):
        pass

    # Finally, let's iterate over the archive core rows and get the data:
    for row in dwca:
        # row is an instance of CoreRow
        # iteration respects their order of appearance in the core file

        # Print() can be used for debugging purposes...
        print(row)

        # => --
        # => Rowtype: http://rs.tdwg.org/dwc/terms/Occurrence
        # => Source: Core file
        # => Row ID:
        # => Data: {u'http://rs.tdwg.org/dwc/terms/basisOfRecord': u'Observation', u
↪'http://rs.tdwg.org/dwc/terms/family': # => u'Tetraodontidae', u'http://rs.tdwg.org/
↪dwc/terms/locality': u'Borneo', u'http://rs.tdwg.#
        # => org/dwc/terms/scientificName': u'tetraodon fluviatilis'}
        # => --

        # You can get the value of a specific Darwin Core term through
        # the "data" dict:
        print("Value of 'locality' for this row: %s" % row.data[qn('locality')])
        # => Value of 'locality' for this row: Mumbai

    # Alternatively, we can get a list of core rows instead of iterating:
    # BEWARE: all rows will be loaded in memory!
    rows = dwca.rows

    # Or retrieve a specific row by its id:
    occurrence_number_three = dwca.get_row_by_id(3)

    # Caution: ids are generally a fragile way to identify a core row in an archive,␣
↪since the standard doesn't
    # guarantee unicity (nor even that there will be an id). The index (position) of␣
↪the row (starting at 0) is
```

(continues on next page)

(continued from previous page)

```
    # generally preferable.

    occurrence_on_second_line = dwca.get_row_by_index(1)

    # We can retreive the (absolute) of embedded files
    # NOTE: this path point to a temporary directory that will be removed at the end␣
↪of the DwCAReader object life
    # cycle.
    path = dwca.absolute_temporary_path('occurrence.txt')
```

Access to Darwin Core Archives with extensions (star schema)

```
from dwca.read import DwCAReader

with DwCAReader('archive_with_vernacularnames_extension.zip') as dwca:
    # Let's ask the archive what kind of extensions are in use:
    for e in dwca.descriptor.extensions:
        print(e.type)
    # => http://rs.gbif.org/terms/1.0/VernacularName

    first_core_row = dwca.rows[0]

    # Extension rows are accessible from a core row as a list of ExtensionRow␣
↪instances:
    for extension_line in first_core_row.extensions:
        # Display all rows from extension files reffering to the first Core row
        print(extension_line)
```

Another example with multiple extensions (no new API here)

```
from dwca.read import DwCAReader

with DwCAReader('multiext_archive.zip') as dwca:
    rows = dwca.rows
    ostrich = rows[0]

    print("You'll find below all extensions rows reffering to Ostrich")
    print("There should be 3 vernacular names and 2 taxon description")
    for ext in ostrich.extensions:
        print(ext)

    print("We can then simply filter by type...")
    for ext in ostrich.extensions:
        if ext.rowtype == 'http://rs.gbif.org/terms/1.0/VernacularName':
            print(ext)
```

GBIF Downloads

The GBIF website allow visitors to export occurrences as a Darwin Core Archive. The resulting file contains a few more things that are not part of the Darwin Core Archive standard. These additions also works with python-dwca-reader. See *The GBIF Occurrence download format* for explanations on the file format and how to use it.

Data analysis and manipulation with Pandas

Python-dwca-reader provides specific tools to make working with Pandas easier, see *Interaction with Pandas Package* for concrete examples.

3.3 Interaction with Pandas Package

Warning: You'll need to install Pandas first.

Pandas is a powerful data analysis package that provides the user a large set of functionalities, such as easy slicing, filtering, calculating and summarizing statistics or plotting.

Python-dwca-reader exposes a `pd_read()` method to easily load the content of a data file (core or extension) from the archive into a Pandas DataFrame.

```
from dwca.read import DwCAReader

with DwCAReader('gbif-results.zip') as dwca:
   print("Core data file is: {}".format(dwca.descriptor.core.file_location)) # =>
↪'occurrence.txt'

   core_df = dwca.pd_read('occurrence.txt', parse_dates=True)

   # All Pandas functionalities are now available on the core_df DataFrame
```

Note: `DwCAReader.pd_read()` is a simple wrapper around pandas.read_csv() and accept the same optional arguments. Only a few of them (*delimiter*, *skiprows*, *encoding*, ...) will be ignored because DwCAReader sets them appropriately for the data file.

Note: Alternatively, you can do *core_df = dwca.pd_read(dwca.core_file_location, ...)* which is handy if you don't know the name of the core data file.

As a small example, some applications on the `core_df`:

Warning: You'll need to install Seaborn for this example.

```
import pandas as pd
import seaborn as sns

# Number of records for each institutioncode
core_df["institutionCode"].value_counts()

# Select the coordinate information of the first twenty records
core_df.loc[:20, ["decimalLatitude", "decimalLongitude"]]

# Count the number of records with date information after 1950
sum(core_df["year"] > 1950)
```

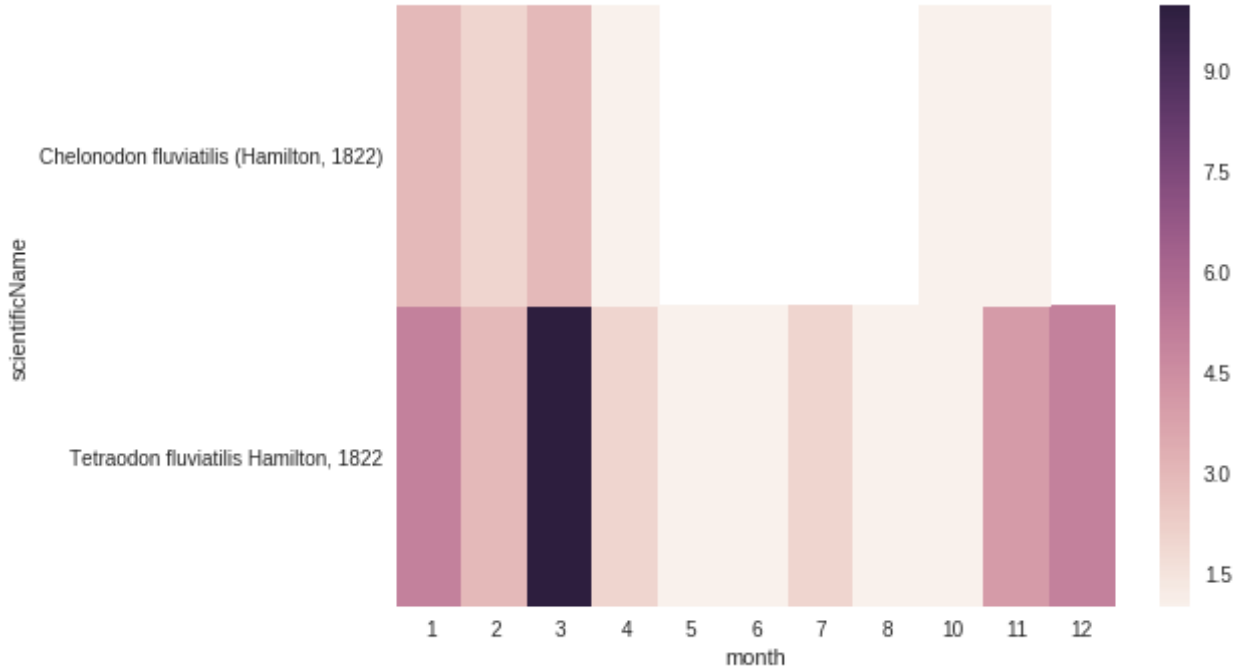
(continues on next page)

(continued from previous page)

```
# Convert eventDate to DateTime python object
core_df['eventDate'] = pd.to_datetime(core_df['eventDate'])

# Select only those records with coordinates, not (0, 0) coordinates and eventDate␣
↪provided
core_df[(core_df["decimalLatitude"] != 0.0) &
        (core_df["decimalLatitude"].notnull()) &
        (core_df["decimalLongitude"] != 0.0) &
        (core_df["decimalLongitude"].notnull()) &
        (core_df["eventDate"].notnull())]

# Count the number of records for each species for each month
count_occ = core_df.pivot_table(index="scientificName",
                                columns="month",
                                values="id",
                                aggfunc='count')
# Visualisation of the counts on a heatmap (Seaborn)
sns.heatmap(count_occ)
```



For more information about Pandas and Seaborn, see their respective documentation.

When the DwCA contains multiple files, joining the extensions with the core file could be of interest for further analysis.

```
import pandas as pd
from dwca.read import DwCAReader

with DwCAReader('dwca-2extensions.zip') as dwca:
    # Check the core file of the Archive  (Occurrence, Taxon, ...)
    print("Core type is: {}".format(dwca.descriptor.core.type))
```

(continues on next page)

(continued from previous page)

```
    # Check the available extensions
    print("Available extensions: {}".format([ext.split("/")[-1] for ext in dwca.
↪descriptor.extensions_type]))

    taxon_df = dwca.pd_read('taxon.txt')
    descr_df = dwca.pd_read('description.txt')
    vern_df = dwca.pd_read('vernacularname.txt')

# Join the information of the description and vernacularname extension to the core␣
↪taxon information
# (cfr. database JOIN)
taxon_df = pd.merge(taxon_df, descr_df, left_on='id', right_on='coreid', how="left")
taxon_df = pd.merge(taxon_df, vern_df, left_on='id', right_on='coreid', how="left")
```

The result is the core file joined with the extension files. More information about the Pandas merge is provided in the documentation.

Remark that reading in the data to Pandas will load the entire file into memory. For large archives, this won't be feasible. Pandas support the usage of chunks, reading in a processing the data in chunks. As an example, consider the selection of those occurrences for which the `eventDate` was a Sunday:

```
import pandas as pd
from dwca.read import DwCAReader

chunksize = 10 # Chosen chunksize to process the data (pick a larger value for real␣
↪world cases)
with DwCAReader('gbif-results.zip') as dwca:
    sunday_occ = []
    for chunk in dwca.pd_read('occurrence.txt', chunksize=chunksize):
        chunk['eventDate'] = pd.to_datetime(chunk['eventDate'])

        # Subselect only the records recorded on a sunday
        sunday_occ.append(chunk[chunk['eventDate'].dt.weekday == 6]) # Monday = 0,␣
↪Sunday = 6

sunday_occ = pd.concat(sunday_occ)
```

More advanced processing is supported by Pandas. However, when only interested in counting the number of occurrences for a specific condition, Pandas is not always required. As an example, counting the number of occurrences for each species in the data set is easily supported by the `Counter` datatype of Python:

```
from collections import Counter

from dwca.read import DwCAReader
from dwca.darwincore.utils import qualname as qn

with DwCAReader('/Users/nicolasnoe/Desktop/gbif-results.zip') as dwca:
    count_species = Counter()

    for row in dwca:
        count_species.update([row.data[qn('scientificName')]])

    print(count_species)
```

Hence, the added value of Pandas depends on the type of analysis. Some more extensive applications of Pandas to work with Darwin Core data is provided in this data cleaning tutorial and data analysis tutorial.

3.4 Complete API

3.4.1 Reader objects

High-level classes to open and read DarwinCore Archive.

class dwca.read.**DwCAReader**(*path: str*, *extensions_to_ignore: List[str] = None*, *tmp_dir: str = None*)

Bases: object

This class is used to represent a Darwin Core Archive as a whole.

It gives read access to the contained data, to the scientific metadata, ... It supports archives with or without Metafile, such as described on page 2 of the Reference Guide to the XML Descriptor.

Parameters

- **path** (*str*) – path to the Darwin Core Archive (either a zip/tgz file or a directory) to open.
- **extensions_to_ignore** (*list*) – path (relative to the archive root) of extension data files to ignore. This will improve speed and memory usage for large archives. Missing files are silently ignored.
- **tmp_dir** (*str*) – temporary directory to use to uncompress the archive (if needed). If not provided, Python default will be used.

Raises *dwca.exceptions.InvalidArchive*

Raises *dwca.exceptions.InvalidSimpleArchive*

Usage:

```
from dwca.read import DwCAReader

dwca = DwCAReader('my_archive.zip')

# Iterating on core rows is easy:
for core_row in dwca:
    # core_row is an instance of dwca.rows.CoreRow
    print(core_row)

# Scientific metadata (EML) is available as an ElementTree.Element object
print(dwca.metadata)

# Close the archive to free resources
dwca.close()
```

The archive can also be opened using the *with* statement. This is recommended, since it ensures resources will be properly cleaned after usage:

```
from dwca.read import DwCAReader

with DwCAReader('my-archive.zip') as dwca:
    pass  # Do what you want

# When leaving the block, resources are automatically freed.
```

absolute_temporary_path(*relative_path: str*) → str

Return the absolute path of a file located within the archive.

This method allows raw access to the files contained in the archive. It can be useful to open additional, non-standard files embedded in the archive, or to open a standard file with another library.

Parameters **relative_path** (*str*) – the path (relative to the archive root) of the file.

Returns the absolute path to the file.

Usage:

```
dwca.absolute_temporary_path('occurrence.txt')  # => /tmp/afdfsec7/occurrence.
↪txt
```

Warning: If the archive is contained in a zip or tgz file, the returned path will point to a temporary file that will be removed when closing the *dwca.read.DwCAReader* instance.

Note: File existence is not tested.

archive_path = None
The path to the Darwin Core Archive file, as passed to the constructor.

close() → None
Close the Darwin Core Archive and remove temporary/working files.

Note:

- Alternatively, *DwCAReader* can be instanciated using the *with* statement. (see example above).

core_contains_term(*term_url: str*) → bool
Return *True* if the Core file of the archive contains the *term_url* term.

core_file = None
An instance of *dwca.files.CSVDataFile* for the core data file.

property core_file_location
The (relative) path to the core data file.

Example: *'occurrence.txt'*

descriptor = None
An `descriptors.ArchiveDescriptor` instance giving access to the archive descriptor/metafile (`meta.xml`)

extension_files = None
A list of *dwca.files.CSVDataFile*, one entry for each extension data file , sorted by order of appearance in the Metafile (or an empty list if the archive doesn't use extensions).

get_corerow_by_id(*row_id: str*) → dwca.rows.CoreRow
Return the (core) row whose ID is *row_id*.

Parameters **row_id** (*str*) – ID of the core row you want

Returns *dwca.rows.CoreRow* – the matching row.

Raises *dwca.exceptions.RowNotFound*

> **Warning:** It is rarely a good idea to rely on the row ID, because: 1) Not all Darwin Core Archives specifies row IDs. 2) Nothing guarantees that the ID will actually be unique within the archive (depends of the data publisher). In that case, this method don't guarantee which one will be returned. *get_corerow_by_position()* may be more appropriate in this case.

get_corerow_by_position(*position: int*) → dwca.rows.CoreRow
Return a core row according to its position/index in core file.

Parameters **position** (*int*) – the position (starting at 0) of the row you want in the core file.

Returns *dwca.rows.CoreRow* – the matching row.

Raises *dwca.exceptions.RowNotFound*

Note:

- If index is bigger than the length of the archive, None is returned
- The position is often an appropriate way to unambiguously identify a core row in a DwCA.

---

get_descriptor_for(*relative_path: str*) → dwca.descriptors.DataFileDescriptor
Return a descriptor for the data file located at relative_path.

Parameters **relative_path** (*str*) – the path (relative to the archive root) to the data file you want info about.

Returns *dwca.descriptors.DataFileDescriptor*

Raises *dwca.exceptions.NotADataFile* if *relative_path* doesn't reference a valid data file.

Examples:

```
dwca.get_descriptor_for('occurrence.txt')
dwca.get_descriptor_for('verbatim.txt')
```

metadata = None
A `xml.etree.ElementTree.Element` instance containing the (scientific) metadata of the archive, or *None* if the archive has no metadata.

open_included_file(*relative_path: str*, **args: Any*, ***kwargs: Any*) → IO
Simple wrapper around Python's build-in *open* function.

To be used only for reading.

> **Warning:** Don't forget to close the files after usage. This is especially important on Windows because temporary (extracted) files won't be cleanable if not closed.

orphaned_extension_rows() → Dict[str, Dict[str, List[int]]]
Return a dict of the orphaned extension rows.

Orphaned extension rows are extension rows who reference non-existing core rows. This methods returns a dict such as:

```
{'description.txt': {u'5': [3, 4], u'6': [5]},
 'vernacularname.txt': {u'7': [4]}}
```

Meaning:

- in *description.txt*, rows at position 3 and 4 reference a core row whose ID is '5', but such a core row doesn't exists. Row at position 5 references an imaginary core row with ID '6'
- in *vernacularname.txt*, the row at position 4 references an imaginary core row with ID '7'

pd_read(*relative_path*, ***kwargs*)

Return a Pandas DataFrame for the data file located at *relative_path*.

This method wraps pandas.read_csv() and accept the same keyword arguments. The following arguments will be ignored (because they are set appropriately for the data file): *delimiter*, *skiprows*, *header* and *names*.

Parameters **relative_path** (*str*) – path to the data file (relative to the archive root).

Raises *ImportError* if Pandas is not installed.

Raises *dwca.exceptions.NotADataFile* if *relative_path* doesn't designate a valid data file in the archive.

> **Warning:** You'll need to install Pandas before using this method.

> **Note:** Default values of Darwin Core Archive are supported: A column will be added to the DataFrame if a term has a default value in the Metafile (but no corresponding column in the CSV Data File).

property rows

A list of `rows.CoreRow` objects representing the content of the archive.

> **Warning:** All rows will be loaded in memory. In case of a large Darwin Core Archive, you may prefer using a for loop.

source_metadata = None

If the archive contains source-level metadata (typically, GBIF downloads), this is a dict such as:

```
{'dataset1_UUID': <dataset1 EML> (xml.etree.ElementTree.Element object),
 'dataset2_UUID': <dataset2 EML> (xml.etree.ElementTree.Element object), ...}
```

See *The GBIF Occurrence download format* for more details.

property use_extensions

True if the archive makes use of extensions.

3.4.2 Row objects

Objects that represents data rows coming from DarwinCore Archives.

class dwca.rows.**CoreRow**(*csv_line: str*, *position: int*, *datafile_descriptor: dwca.descriptors.DataFileDescriptor*)

Bases: *dwca.rows.Row*

This class is used to represent a row/line from a Darwin Core Archive core data file.

You probably won't instantiate it manually but rather obtain it via *dwca.read.DwCAReader.get_corerow_by_position()*, *dwca.read.DwCAReader.get_corerow_by_id()* or simply by looping over a *dwca.read.DwCAReader* object.

property extensions
A list of *ExtensionRow* instances that relates to this Core row.

id = None
The row id

class dwca.rows.**ExtensionRow**(*csv_line: str, position: int, datafile_descriptor: dwca.descriptors.DataFileDescriptor*)
Bases: *dwca.rows.Row*

This class is used to represent a row/line from a Darwin Core Archive extension data file.

Most of the time, you won't instantiate it manually but rather obtain it trough the extensions attribute of *CoreRow*.

core_id = None
The id of the core row this extension row is referring to.

class dwca.rows.**Row**(*csv_line: str, position: int, datafile_descriptor: dwca.descriptors.DataFileDescriptor*)
Bases: object

This class is used to represent a row/line in a Darwin Core Archive.

This class is intended to be subclassed rather than used directly.

data = None
A dict containing the Row data, such as:

```
{'dwc_term_1': 'value',
 'dwc_term_2': 'value',
 ...}
```

Usage:

```
myrow.data['http://rs.tdwg.org/dwc/terms/locality']  # => "Brussels"
```

Note: The *dwca.darwincore.utils.qualname()* helper is available to make such calls less verbose.

descriptor = None
An instance of *dwca.descriptors.DataFileDescriptor* describing the originating data file.

position = None
The row position/index (starting at 0) in the source data file. This can be used, for example with *dwca.read.DwCAReader.get_corerow_by_position()* or *dwca.files.CSVDataFile.get_row_by_position()*.

raw_fields = None

rowtype = None
The csv line type as stated in the archive descriptor. (or None if the archive has no descriptor). Examples: http://rs.tdwg.org/dwc/terms/Occurrence, http://rs.gbif.org/terms/1.0/VernacularName, ...

dwca.rows.**csv_line_to_fields**(*csv_line*, *line_ending*, *field_ending*, *fields_enclosed_by*)
Split a line from a CSV file.

Return a list of fields. Content is not trimmed.

3.4.3 Descriptor objects

Classes to represents descriptors of a DwC-A.

- *ArchiveDescriptor* represents the full archive descriptor, initialized from the metafile content.
- *DataFileDescriptor* describes characteristics of a given data file in the archive. It's either created from a subsection of the ArchiveDescriptor describing the data file, either by introspecting the CSV data file (useful for Archives without metafile).

class dwca.descriptors.**ArchiveDescriptor**(*metaxml_content: str*, *files_to_ignore: List[str] = None*)

Bases: object

Class used to encapsulate the whole Metafile (*meta.xml*).

extensions = None
A list of *dwca.descriptors.DataFileDescriptor* instances describing each of the archive's extension data files.

extensions_type = None
A list of extension (types) in use in the archive.

Example:

```
["http://rs.gbif.org/terms/1.0/VernacularName",
 "http://rs.gbif.org/terms/1.0/Description"]
```

metadata_filename = None
The path (relative to archive root) of the (scientific) metadata of the archive.

raw_element = None
A xml.etree.ElementTree.Element instance containing the complete Archive Descriptor.

class dwca.descriptors.**DataFileDescriptor**(*created_from_file: bool, raw_element: xml.etree.ElementTree.Element, represents_corefile: bool, datafile_type: Optional[str], file_location: str, file_encoding: str, id_index: int, coreid_index: int, fields: List[Dict], lines_terminated_by: str, fields_enclosed_by: str, fields_terminated_by: str*)

Bases: object

Those objects describe a data file fom the archive.

They're generally not instanciated manually, but rather by calling:

- *make_from_metafile_section()* (if the archive contains a metafile)
- *make_from_file()* (created by analyzing the data file)

coreid_index = None
If the section represents an extension data file, the index/position of the core_id column in that file. The *core_id* in an extension is the foreign key to the "extended" core row.

created_from_file = None
True if this descriptor was created by analyzing the data file.

fields = None
A list of dicts where each entry represent a data field in use.

Each dict contains:

- The term identifier
- (Possibly) a default value
- The column index/position in the CSV file (except if we use a default value instead)

Example:

```
[{'term': 'http://rs.tdwg.org/dwc/terms/scientificName',
  'index': '1',
  'default': None},

 {'term': 'http://rs.tdwg.org/dwc/terms/locality',
  'index': '2',
  'default': ''},

 # The data for `country` is a the default value 'Belgium' for all rows, so␣
↪there's
 # no column in CSV file.

 {'term': 'http://rs.tdwg.org/dwc/terms/country',
  'index': None,
  'default': 'Belgium'}]
```

fields_enclosed_by = None
The string or character used to enclose fields in the data file.

fields_terminated_by = None
The string or character used as a field separator in the data file. Example: "\t".

file_encoding = None
The encoding of the data file. Example: "utf-8".

file_location = None
The data file location, relative to the archive root.

property headers
A list of (ordered) column names that can be used to create a header line for the data file.

Example:

```
['id', 'http://rs.tdwg.org/dwc/terms/scientificName', 'http://rs.tdwg.org/dwc/
↪terms/basisOfRecord',
'http://rs.tdwg.org/dwc/terms/family', 'http://rs.tdwg.org/dwc/terms/locality
↪']
```

See also *short_headers* if you prefer less verbose headers.

id_index = None
If the section represents a core data file, the index/position of the id column in that file.

lines_terminated_by = None
The string or character used as a line separator in the data file. Example: "\n".

property lines_to_ignore
Return the number of header lines/lines to ignore in the data file.

classmethod make_from_file(*datafile_path*)
Create and return a DataFileDescriptor by analyzing the file at datafile_path.

Parameters **datafile_path** (*str*) – Relative path to a data file to analyze in order to instantiate the descriptor.

classmethod make_from_metafile_section(*section_tag*)
Create and return a DataFileDescriptor from a metafile <section> tag.

Parameters **section_tag** (`xml.etree.ElementTree.Element`) – The XML Element section containing details about the data file.

raw_element = None
The <section> element describing the data file, from the metafile. None if the archive contains no metafile.

represents_corefile = None
True if this descriptor is used to represent the core file an archive.

represents_extension = None
True if this descriptor is used to represent an extension file in an archive.

property short_headers
A list of (ordered) column names (short version) that can be used to create a header line for the data file.

Example:

```
['id', 'scientificName', 'basisOfRecord', 'family', 'locality']
```

See also *headers*.

property terms
Return a Python set containing all the Darwin Core terms appearing in file.

type = None

3.4.4 File objects

File-related classes and functions.

class dwca.files.**CSVDataFile**(*work_directory: str*, *file_descriptor: dwca.descriptors.DataFileDescriptor*)
Object used to access a DwCA-enclosed CSV data file.

Parameters

- **work_directory** – absolute path to the target directory (archive content, previously extracted if necessary).
- **file_descriptor** – an instance of *dwca.descriptors.DataFileDescriptor* describing the data file.

The file content can be accessed:

- By iterating on this object: a str is returned, including separators.
- With *get_row_by_position()* (A *dwca.rows.CoreRow* or *dwca.rows.ExtensionRow* object is returned)
- For an extension data file, with *get_all_rows_by_coreid()* (A *dwca.rows.CoreRow* or *dwca.rows.ExtensionRow* object is returned)

On initialization, an index of new lines is build. This may take time, but makes random access faster.

close() → None
Close the file.

The content of the file will not be accessible in any way afterwards.

property coreid_index
An index of the core rows referenced by this data file.

It is a Python dict such as:

```
{
core_id1: [1],     # Row at position 1 references a Core Row whose ID is core_
↪id1
core_id2: [8, 10] # Rows at position 8 and 10 references a Core Row whose ID␣
↪is core_id2
}
```

Raises AttributeError if accessed on a core data file.

Warning: for permformance reasons, dictionary values are arrays('L') instead of regular python lists

Warning: coreid_index is only available for extension data files.

Warning: Creating this index can be time and memory consuming for large archives, so it's created on the fly at first access.

file_descriptor = None
An instance of *dwca.descriptors.DataFileDescriptor*, as given to the constructor.

get_all_rows_by_coreid(*core_id: int*) → List[dwca.rows.ExtensionRow]
Return a list of *dwca.rows.ExtensionRow* whose Core Id field match *core_id*.

get_row_by_position(*position: int*) → Union[dwca.rows.CoreRow, dwca.rows.ExtensionRow]
Return the row at *position* in the file.

Header lines are ignored.

Raises IndexError if there's no line at *position*.

lines_to_ignore = None
Number of lines to ignore (header lines) in the CSV file.

3.4.5 Helpers

This module contains small helpers to make life easier.

dwca.darwincore.utils.**qualname**(*short_term*)
Takes a darwin core term (short form) and returns the corresponding qualname.

Note: It is generally used to make data access less verbose (see example below).

Raises `StopIteration` if short_term is not found.

Typical real-world example:

```
from dwca.darwincore.utils import qualname as qn

qn("Occurrence")  # => "http://rs.tdwg.org/dwc/terms/Occurrence"

# To access data row:
myrow.data[qn('scientificName')]  # => u"Tetraodon fluviatilis"

# Instead of the verbose:
myrow.data['http://rs.tdwg.org/dwc/terms/scientificName']  # => u"Tetraodon␣
↪fluviatilis"
```

3.4.6 Exceptions

Exceptions for the whole package.

exception `dwca.exceptions.`**InvalidArchive**
The archive appears to be invalid.

exception `dwca.exceptions.`**InvalidSimpleArchive**
The simple archive appears to be invalid.

exception `dwca.exceptions.`**NotADataFile**
The file doesn't exists or is not a data file.

exception `dwca.exceptions.`**RowNotFound**
The DwC-A Row cannot be found.

3.5 The GBIF Occurrence download format

Since 2013, the GBIF Data Portal exports occurrences (search results) in a format that is a superset of the Darwin Core Archive standard.

Python-dwca-reader used to provide a specialized `GBIFResultsReader` class that gave access to its specificities. `GBIFResultsReader` is now deprecated, but its features have been merged into `DwCAReader`. The rest of this document describe the specifics of GBIF downloads, and how to use them with python-dwca-reader.

3.5.1 Additions to the Darwin Core Archive standard & how to use

Warning: Those additions are not part of the official standard, and the GBIF download format can evolve at any point without prior announcement.

Source metadata

In addition to the general metadata file (`metadata.xml`), the archive also contains a `dataset` subdirectory. Each file in this subdirectory is an EML document describing a dataset whose rows are part of the archive data. The file name is `"<DATASET_UUID>.xml"`. Each row in `occurrence.txt` refers to this file using the `datasetID` column.

You can access this source metadata like this:

```
from dwca.read import DwCAReader

with DwCAReader('gbif-results.zip') as results:
    # 1. At the archive level, through the source_metadata dict:

    print(results.source_metadata)
    # {'dataset1_UUID': <dataset1 EML (xml.etree.ElementTree.Element instance)>,
    #  'dataset2_UUID': <dataset2 EML (xml.etree.ElementTree.Element instance)>, ...}

    # 2. From a CoreRow instance, we can get back to the metadata of its source␣
↪dataset:
    first_row = results.get_row_by_index(0)
    print(first_row.source_metadata)
    # => <Source dataset EML (xml.etree.ElementTree.Element instance)>
```

Interpreted/verbatim occurrences and multimedia data

While the Core data file (`occurrence.txt`) contains GBIF-interpreted occurrences, the verbatim (as published) data is also made available with an extension in `verbatim.txt`. Similarly, if there's multimedia information attached to the record it will be availabe in the `multimedia.txt` extension file.

Because there's a standard core-extension relationship (star schema) between those entities, you can access the related data from the core row using the usual extension mechanism:

```
from dwca.read import DwCAReader

with DwCAReader('gbif-results.zip') as results:
    first_row = results.get_row_by_index(0)

    first_row.extensions
```

Additional text files

The archive contains additional files such as `rights.txt` (aggregated IP rights) and `citations.txt` (citation information for the search results).

You can access the content of those files:

```
from dwca.read import DwCAReader

with DwCAReader('gbif-results.zip') as results:
    citations = results.open_included_file('citations.txt').read()
```

3.6 Contributing to Python-dwca-reader

Contributions are more than welcome! Please also provide tests and documentation for your contributions.

3.6.1 Running the test suite

```
$ pip install nose
$ nosetests
```

Test coverage can be obtained after installing coverage.py

```
nosetests --with-coverage --cover-erase --cover-package=dwca
.....................................................................
Name                            Stmts   Miss  Cover
-------------------------------------------------
dwca/__init__.py                    0      0   100%
dwca/darwincore/__init__.py         0      0   100%
dwca/darwincore/terms.py            1      0   100%
dwca/darwincore/utils.py            4      0   100%
dwca/descriptors.py                92      1    99%
dwca/exceptions.py                  5      0   100%
dwca/files.py                      63      1    98%
dwca/read.py                      186      1    99%
dwca/rows.py                       96     11    89%
dwca/vendor.py                      5      2    60%
-------------------------------------------------
TOTAL                             452     16    96%
----------------------------------------------------------------------
Ran 104 tests in 1.514s

OK
```

3.6.2 Building the documentation

Locally:

```
$ pip install sphinx sphinx-rtd-theme
$ cd doc; make clean; make html
```

Online at http://python-dwca-reader.readthedocs.org/:

The online docs will be updated automagically after pushing to GitHub.

3.6.3 Releasing at PyPI

- (Ensuring it works -also on Windows-, the test coverage is good and the documentation is updated)
- Update the packaging (version number in dwca/version.py, CHANGES.txt, . . .) then run:

```
$ python setup.py sdist bdis_wheel
$ twine upload dist/*
```

- Create a new tag and push it to GitHub

```
$ git tag vX.Y.Z
$ git push origin --tags
```

3.7 Glossary

3.7.1 Metadata / Scientific metadata

3.7.2 Metafile

3.7.3 Simple archive / Simple Darwin Core

3.7.4 Source metadata

3.8 Changelog

3.8.1 Current

- GBIFResultsReader (deprecated since v0.10.0) is now removed
- DwCAReader.get_row_by_id() (long deprecated) is now removed, use DwCAReader.get_corerow_by_id() instead
- DwCAREader.get_row_by_index() (long deprecated) is now removed, use DwCAReader.get_corerow_by_position() instead

3.8.2 v0.14.0 (2020-04-27)

- Dropped support for Python 2.7
- API new: Temporary directory is now configurable

3.8.3 v0.13.2 (2019-09-20)

- Better standard support: fields with data column can also have a default value (issue #80)

3.8.4 v0.13.1 (2018-08-30)

- API new: DwCAReader.core_file_location
- API new: String representation (__str__) for CSVDataFile objects.
- API change: CSVDataFile.get_line_at_position() raises IndexError in case of line not found (previously: returned None)

3.8.5 v0.13.0 (2017-12-01)

- Bugfix: better encoding support for Metadata file - see issue #73.
- API change: DwCAReader.get_descriptor_for(filename) now raises NotADataFile exception if filename doesn't exists (previously: None was returned).
- API new: DwCAReader.core_file (previously private: _corefile).
- API new: DwCAReader.extension_files (previously private: _extensionfiles).

3.8.6 v0.12.0 (2017-11-10)

- API new: DwCAReader.pd_read() - See Pandas tutorial for usage.
- API new: new NotADataFile exception.

3.8.7 v0.11.2 (2017-10-18)

- API new: DwCAReader.get_descriptor_for()

3.8.8 v0.11.1 (2017-10-11)

- API new: DataFileDescriptor.short_headers

3.8.9 v0.11.0 (2017-10-10)

- Bugfix: An enclosed field can now contain the field separator (By Ben Cail)
- API change: DwCAReader.get_row_by_id() is renamed to DwCAReader.get_corerow_by_id()
- API change: DwCAReader.get_row_by_index() is renamed DwCAReader.get_corerow_by_position()
- API new: DwCAReader.orphaned_extension_rows() (thanks to Pieter Provoost).
- API new: CSVDataFile.coreid_index (was previously known as CSVDataFile._coreid_index).
- API new: Row/CoreRow/ExtensionRow.position

3.8.10 v0.10.2 (2017-04-11)

- experimental support for Windows.

3.8.11 v0.10.1 (2017-04-04)

- fixed temporary directory: previously, it was always created under current dir instead of something chosen by Python such as /tmp.

3.8.12 v0.10.0 (2017-03-16)

- GBIFResultsReader is now deprecated.
- API new: DwCAReader now provides source_metadata for GBIF-like archives (previously the main perk of GBIFResultsReader)
- API change: dwca.source_metadata is an empty dict (previously: None) when the archive doesn't has source metadata
- API new: dwca.utils._DataFile is now public and renamed to dwca.files.CSVDataFile.

3.8.13 v0.9.2 (2016-04-29)

- Updated Darwin Core terms for the qualname helper (including Event Core).
- Updated dev. script (https://github.com/BelgianBiodiversityPlatform/python-dwca-reader/issues/45).

3.8.14 v0.9.1 (2016-04-28)

- API new: DwCAReader.open_included_file(relative_path).
- InvalidArchive exception is now raised when a file descriptor references a non-existing field.

3.8.15 v0.9.0 (2016-04-05)

- Support for new types of archives:
  - DwCA without metafile (see page 2 of http://www.gbif.org/resource/80639), including or not a Metadata document (EML.xml). Fixes #47 and #7.
  - DwCA where data fields are enclosed by some character (using fieldsEnclosedBy when the Archive provides a Metafile, autodetection otherwise). Fixes issue #53.
  - Archives without a metadata attribute in the Metafile. See #51.
  - Tgz archives.
- API change: SectionDescriptor => DataFileDescriptor
- API change: DataFileDescriptor.encoding => DataFileDescriptor.file_encoding
- API change: the reader previously only supported zip archives, and raised BadZipFile when the file format was not understood. It nows also supports .tgz, and throw InvalidArchive when the archive cannot be opened as .zip nor as .tgz.
- API new: DataFileDescriptor.fields_enclosed_by
- API new: DwCAReader.use_extensions

3.8.16 v0.8.1 (2016-03-10)

- Support for archives contained in a single (sub)directory. See https://github.com/BelgianBiodiversityPlatform/python-dwca-reader/issues/49

3.8.17 v0.8.0 (2016-02-11)

- Experimental support for Python 3.5 (while maintaining compatibility with 2.7)

3.8.18 v0.7.0 (2015-08-20)

- Python-dwca-reader doesn't rely anymore on BeautifulSoup/lxml, using ElementTree from the standard library instead fot its XML parsing needs. This has a series of consequences: * It should be easier to install and deploy (also on platforms such as Jython). * All methods and attributes that used to return BeautifulSoup objects will now return xml.etree.ElementTree.Element instances. This includes DwCAReader.metadata. SectionDescriptor.raw_beautifulsoup and ArchiveDescriptor.raw_beautifulsoup have been replaced by SectionDescriptor.raw_element and ArchiveDescriptor.raw_element (Element objects).

3.8.19 v0.6.5 (2015-08-18)

- New InvalidArchive exception. Currrently, it is only raised when a DwC-A references a metadata file that's not present in the archive.

3.8.20 v0.6.4 (2015-02-17)

- Performance: an optional 'extension_to_ignores' parameter (List) can be passed to DwCAReader's constructor. In cases where an archive contains large but unneeded extensions, this can greatly improve memory consumption. A typical use-case for that would be the huge 'verbatim.txt' contained in GBIF downloads.

3.8.21 v0.6.3 (2015-02-16)

- Performance: we now use core_id based indexes for extensions. There's a memory penalty, but extension file parsing is now only done once.

3.8.22 v0.6.2 (2015-01-26)

- Better performance with extensions.

3.8.23 v0.6.1 (2015-01-09)

- It can now open not zipped (directory) Darwin Core Archives
- More testing for Descriptor classes.
- **Better respect of the standard (http://rs.tdwg.org/dwc/terms/guides/text/):**
  - We now support default value (n) for linesTerminatedBy and fieldsTerminatedBy.
- Lower memory use with large archives.

3.8.24 v0.6.0 (2014-08-08)

- Better performance thanks to a better architecture
- API add: brand new _ArchiveDescriptor and _SectionDescriptor
- API change: DwCAReader.descriptor is an instance of _ArchiveDescriptor (previously BeautifulSoup)
- API remove: DwCAReader.core_rowtype (use DwCAReader.descriptor.core.type instead)
- API remove: DwCAReader.extensions_rowtype (use DwCAReader.descriptor.extensions_type instead)
- API remove: DwCAReader.core_terms (use DwCAReader.descriptor.core.terms instead)

3.8.25 v0.5.1 (2014-08-05)

- **Performance: dramatically improved performance of get_row_by_index/looping for large files by** building an index of line positions at file opening (so there's a slight overhead there)

3.8.26 v0.5.0 (2014-01-21)

- API new: DwCAReader.descriptor
- API change: "for core_line in dwca.each_line():" => "for core_row in dwca:"
- **API change: from_core and from_extension attributes of DwCALine (and sublasses) have been removed.** The isinstance built-in function can be used to test if a line is an instance of DwCACoreLine or of DwCAExtensionLine.
- API change: DwCAReader.lines => DwCAReader.rows
- API change: DwCACoreLine => CoreRow
- API change: DwCAExtensionLine => ExtensionRow
- API change: DWCAReader.get_line_by_id => DWCAReader.get_row_by_id
- API change: DWCAReader.get_line_by_index => DWCAReader.get_row_by_index
- API change: DwCAReader.get_row_by_* methods throw RowNotFound when failure instead of returning None
- Cleaner code and better documentation.

3.8.27 v0.4.0 (2013-09-24)

- API change: dwca.get_line() -> dwca.get_line_by_id()
- API new: dwca.get_line_by_index()
- (Core File) iteration order is now guaranteed for dwca.each_line()
- Refactoring: DwCALine subclassed (as DwCACoreLine and DwCAExtensionLine)

3.8.28 v0.3.3 (2013-09-05)

- DwCALines are now hashable.

3.8.29 v0.3.2 (2013-08-28)

- API: added the dwca.absolute_temporary_path() method.

3.8.30 v0.3.1 (2013-08-09)

- Bugfix: lxml added as a requirement.

3.8.31 v0.3.0 (2013-08-08)

- XML parsing (metadata, EML, . . .) now uses BeautifulSoup 4 instead of v3.

3.8.32 v0.2.1 (2013-08-02)

- Added a property (core_terms) to DwCAReader to get a list the DwC terms in use in the core file.

3.8.33 v0.2.0 (2013-07-31)

- Specific support for GBIF Data portal (occurrences) export.
- Small bug fixes.

3.8.34 v0.1.1 (2013-05-28)

- Fixes packaging issues.

3.8.35 v0.1.0 (2013-05-28)

- Initial release.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

d

INDEX