Datconv Documentation

Release 0.8.2

Grzegorz Wierzchowski

Dec 29, 2019

Contents

CHAPTER 1

Important links

If you are reading this from github, package official documentation is avaialble on http://datconv.readthedocs.io.

If you are reading documentation, source code is hosted on https://github.com/gwierzchowski/datconv. In order to communicate with author, report bug etc. - please use Issues tab on github.

CHAPTER 2

Datconv - Universal Data Converter

Datconv is a program designed to perform configurable comversion of file with data in one format to file with data in another format or database.

Program should be run using Python 2.7 or Python 3.x interpretter. It also requires installation of external packages: `lxml`, `PyYAML`. For more information see *INSTALL.rst* file distributed in source pack.

Both input and output files can be text or binary files. However it is assumed that those files have following structure:

—
Header
—
Record 1
Record 2
. . . .
Record N
—
Footer
—

There may be different types of records (i.e. every record has attribute called record type). Each record may contain different number and kind of data (has different internal structure) even among records of the same type.

Program has modular architecture with following swichable compoments:

Reader Major obligatory component responsible for:

- reading input data (i.e. every reader class assumes certain input file format)
- driving entire data conversion process (i.e. main processing loop in implemented in this class)
- determine internal representation of header, records and footer (this heavily depands on reader and kind of input format).

API of this component: *Reader interface*

Filter Optional compoment that is able to:

- filter data (i.e. do not pass certain records further - i.e. to Writer)
- change data (i.e. change on the fly contents of certain records)
- produce data (i.e. cause that certain records, maybe slightly modified, are being sent multiply times to writer)
- break conversion process (i.e. cause that conversion stop on certain record).

API of this component: *Filter interface*

Writer Obligatory component responsible for:

- re-packing data from element-tree internal format to strings or objects.

API of this component: *Writer interface*

Output Connector Obligatory component responsible for:

- writing data to destination storage.

API of this component: *Output Connector interface*

Logger All messages intended to be presented to user are being send (except few very initial error messages) to Logger classes from Python standard package `logging`. Datconv can use all logging comfiguration power available in this package.

In this version of package following compoments are included:

- Readers: XML (sax), CSV (sax), JSON (sax).
- Filters: Few basic/sample filters.
- Writers: XML, CSV, XPath (helper module), JSON.
- Output Connectors: File (Text, MS Excel), Databases (SQLite, PostgreSQL, Crate).

So Datconv program can be used to convert files between XML, CVS and JSON formats and saving data in those formats to database. Sax means that used parsers are of event type - i.e. entire data are not being stored in memory (typically only just one record), what means that program is able to process large files without allocating a lot of memory. It may be also usefull in case you have some files in custom program/company specific data format that you want to look up or convert. Then it is enough to write the reader component for your cutom data format compatible with Datconv API and let Datconv do the rest. Actually this is how I'm currently using this program in my work.

If you'd prefer to work in JavaScript environment please look at Pandat Project which has similar design and purpose.

This program was inspired by design of Pandoc Project.

CHAPTER 3

Performance

Main design principle of this tool was generality and flexibility rather than performance and use scenarios with very big data. This version of program runs in one thread (on one CPU core) and does not consume a lot of modern computer resources. So in case of processing of very big data consider dividing data into smaller chunks and run few instances of this program in parallell or use rfrom-rto parameters avaialble in readers. Or if you have to process big files in short time and do not need that much flexibility (espacially filtering possibilities) probaly special dedicated program (which will not translate data to internal XML-like format) would process your data faster.

Measured performance (version 0.6.0):

- Hardware: Powerfull Laptop (2017 year), CPU Frequency 2.9 GHz, SSD Drive
- Input: XML File: 942MB (400.000 records)
- Output: JSON File: 639MB
- Conversioon time without filter: 4 min 41 s
- With filter (`datconv.filters.delfield`, 2 tags to remove): 4 min 41 s (the same; probably smaller record to write to JSON file compensated effort for Filter invocation).
- Opposite direction (JSON output converted back to XML): 5 min 51 s.

CHAPTER 4

Installation

4.1 Installation and Usage

For package description see README.rst file

4.1.1 Installation (System wide - for all users):

Note: In case of using Python 3 on linux system replace `pip` with `pip3` and `python` with `python3` in below commands.

Prerequisites

This program requires following 3-rd party packages: `PyYAML`, `lxml`. They must be installed in order to run Datconv. In addition JSON readers require `ijson` package to be installed.

PyYAML and **ijson** can be installed from Python Package Index:
Linux: `sudo pip install PyYAML ijson`
Windows: `pip install PyYAML ijson`

lxml is little bit more involved as it is extension package and installation from pip may fail. This package should be installed from system specific installer.
Linux (Debian based): *sudo apt-get install python-lxml* or *sudo apt-get install python3-lxml*
Windows: Download from PyPI appropriate lxml binary windows installer and install from it.

Installation Method 1

Recommended method for installation of this package is from Python Package Index:
Linux: `sudo pip install datconv`
Windows: `pip install datconv`

Installation Method 2

If you want to install particular version, download source-ball and issue:
Linux: `sudo pip install datconv-<ver>.tar.gz`
Windows: `pip install datconv-<ver>.tar.gz`

Installation Method 3

Alternatively unpack source-ball and from unpacked folder run command:
Linux: `sudo python setup.py install`
Windows: `python setup.py install`

Note: More installation options are possible - see documentation of Python `distutils` package.

Installation from Source

If you have downloaded archive snapshot, first unpack it and from root folder run command: `python setup.py sdist` which will create `dist` subfolder and create source-ball in it. Then apply method 2 or 3 above.

Files deployed by installation script

- Main command line utility: Linux: `<BINDATA>/datconv`, where `<BINDATA>` is by default `/usr/local/bin` Windows: `<BINDATA>\Scripts\datconv-run.py`, where `<BINDATA>` is by default `C:\Program Files\PythonXX` where XX is Python version
- `datconv` package and its sub-packages
- Documentation: Linux: `<PREFIX>/share/doc/datconv/*`, where `<PREFIX>` is by default `/usr/local` Windows: `<PREFIX>\doc\datconv\*`, where `<PREFIX>` is by default `C:\Program Files\PythonXX`

4.1.2 Re-Installation / Upgrade

To upgrade packages from PyPi use -U option: `[sudo] pip install -U PyYAML datconv`.
Other installation methods specified above remain valid when upgrading package.

Note: if you upgrade from previous pandata/datconv version check `Upgrade.rst` file deployed in documentation folder.

4.1.3 Usage:

Please refer to on-line or deployed documentation and Pydoc accesible information contained in this package. Consider also installing `datconv_tests` package which contain test scripts for this package. It can also be used as source of samples of how this package may be used.

Official on-line documentation

Official program documentation is avaialble here.

Additional configuration for Pydoc

There are some pydoc descriptions in several script files that are installed into folders not in python module search path, therefore standard pydoc browser would not get those descriptions. It is however possible to hack that: Linux: edit */usr/bin/pydoc* file:

- at begin add: `import sys`
- before `pydoc.gui()` call, add `sys.path.append('/usr/local/bin')`
- create symlink: `sudo ln -s /usr/local/bin/datconv /usr/local/bin/datconv-run.py`.

Windows: edit `C:\Program Files\PythonXX\Tools\Scripts\pydocgui.pyw` file:

- at begin add: `import sys`
- before `pydoc.gui()` call, add `sys.path.append('C:\\Program Files\\PythonXX\\Scripts')`

Obtaining Pydoc information

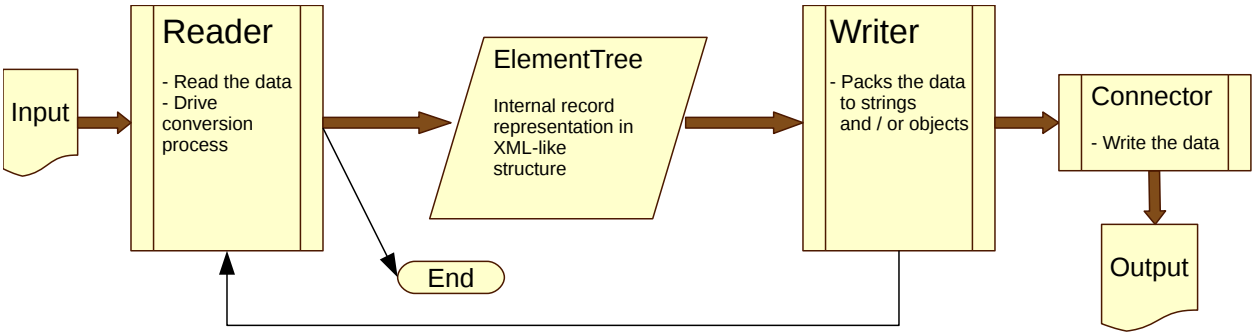
Perform above 'hacks', and run conmand:
Linux (Python2): `pydoc -g`
Linux (Python3): `pydoc3 -b`
Windows: Start Menu/Python/Module Docs
and then press 'Open Browser' or manually navigate to given URL.

Note: This URL does not work with Windows IE browser.

Then go to below sections (near bottom of page):
Linux: `/usr/local/lib/pythonX.X/dist-packages/`, `/usr/local/bin`
Windows: `C:\Program Files\PythonXX\Lib\site-packages`, `C:\Program Files\PythonXX\Scripts` and go into `datconv(package)` or `datconv-run` link.

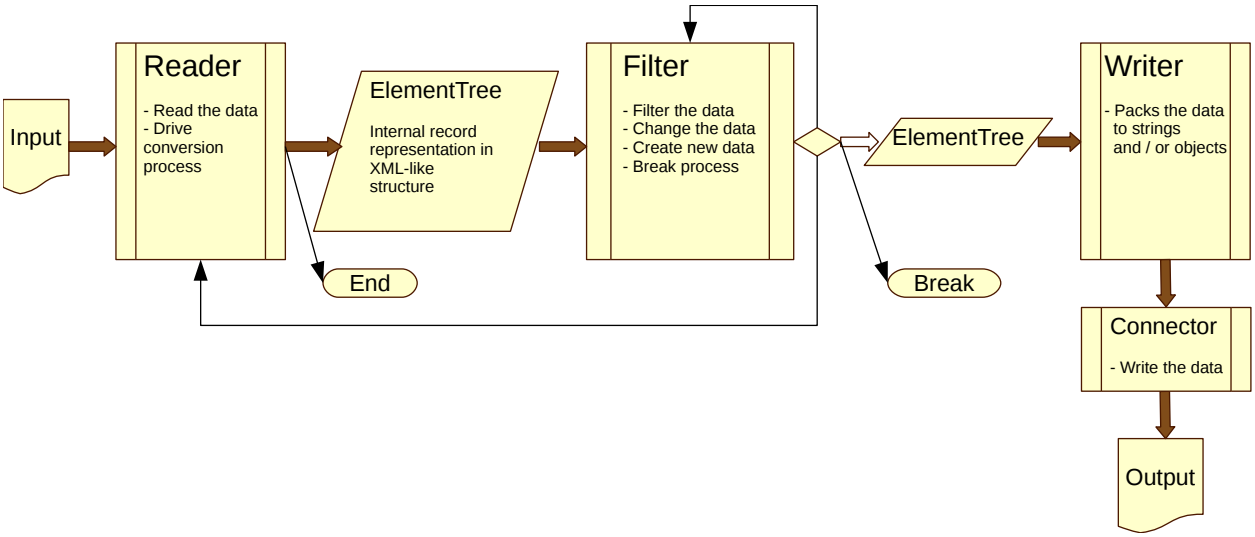
CHAPTER 5

Datconv data flow schema (without filter)



CHAPTER 6

Datconv data flow schema (with filter)



CHAPTER 7

Contents

7.1 Tutorial, Samples

Note: Below examples are suited for Linux system. In Windows System:

- use Windows file paths
- replace main program name `datconv` with `datconv-run.py`
- if you use Notepad++ editor, use `.yml` configuration file extension rather than `.yaml`, this ensures that syntax hightlighting will be working out of box.

7.1.1 Minimal XML to JSON conversion example

Let say we have following XML file that we want to convert to JSON format `file.xml`:

```
<Data>
    <Hello>World</Hello>
</Data>
```

At first we have to write simple configuration file `conv.yaml`:

```
Reader:
    Module:  datconv.readers.dcxml
    CArg:
        bratags: [Data]
    PArg:
        inpath:  ./file.xml
        outpath: ./file.json

Writer:
    Module: datconv.writers.dcjson
```

then, run Datconv tool against this file:

```
datconv ./conv.yaml
```

or skip `PArg:` key in the configuration file and pass file names in command line:

```
datconv ./conv.yaml --Reader:PArg:inpath:./file.xml --Reader:PArg:outpath:./file.json
```

Following `file.json` file will be created:

```
[
{"Data": {}},
{"Hello": "World"}
]
```

If you don't want `Data` object to be in output file, additional parameter must be added to writer's arguments in configuration file:

```
Writer:
    Module: datconv.writers.dcjson
    CArg:
        add_header: false
```

In case your input file would look like this:

```
<Data>
    <Hello World="!" />
</Data>
```

you typically want to add another argument in configuration file:

```
Writer:
    Module: datconv.writers.dcjson
    CArg:
        add_header: false
        with_prop: true
```

to obtain output:

```
[
{"Hello": {"World": "!"}}
]
```

or with yet another option:

```
Writer:
    Module: datconv.writers.dcjson
    CArg:
        add_header: false
        with_prop: true
        json_opt:
            indent: 2
```

```
[
{
  "Hello": {
      "World": "!"
```

(continues on next page)

(continued from previous page)

```
  }
}
]
```

7.1.2 JSON to XML conversion example

Let say we have JSON query result returned by `cbq` tool from Couchbase saved in file `cb.json` that we want to convert to XML:

```
{
    "requestID": "f5a71946-275a-45ef-a13e-f2a335b9b84b",
    "signature": {
        "name": "json",
        "phone": "json"
    },
    "results": [
        {
            "name": "Hilton Chambers",
            "phone": "+44 161 236-4414"
        },
        {
            "name": "Sachas Hotel",
            "phone": null
        },
        {
            "name": "The Mitre Hotel",
            "phone": "+44 161 834-4128"
        }
    ],
    "status": "success",
    "metrics": {
        "elapsedTime": "9.516157ms",
        "executionTime": "9.488693ms",
        "resultCount": 3,
        "resultSize": 253,
        "sortCount": 3
    }
}
```

configuration file would look like this:

```
Reader:
    Module:  datconv.readers.dcijson_keys
    CArg:
        headkeys: [requestID, signature]
        reckeys: [results]
        footkeys: [status, metrics]
    PArg:
        inpath:  ./cb.json
        outpath: ./cb.json.xml

Writer:
    Module: datconv.writers.dcxml
    CArg:
        pretty: true
```

and output file `cb.json.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Datconv>
<requestID val="f5a71946-275a-45ef-a13e-f2a335b9b84b"/>
<signature phone="json" name="json"/>
<results>
    <name>Hilton Chambers</name>
    <phone>+44 161 236-4414</phone>
</results>

<results>
    <name>Sachas Hotel</name>
    <phone>None</phone>
</results>

<results>
    <name>The Mitre Hotel</name>
    <phone>+44 161 834-4128</phone>
</results>

<status val="success"/>
<metrics sortCount="3" executionTime="9.488693ms" elapsedTime="9.516157ms"␣
↪resultCount="3" resultSize="253"/>
</Datconv>
```

7.1.3 XML to CSV conversion example

Let say we want to convert output XML file from above example to CSV.

Configuration file:

```
Reader:
    Module:  datconv.readers.dcxml
    CArg:
        rectags: [results]
    PArg:
        inpath:  ./cb.json.xml
        outpath: ./cb.xml.csv

Writer:
    Module: datconv.writers.dccsv
    CArg:
        columns:
            - ['name','*','name',null]
            - ['phone','*','phone',null]
```

and output file:

```
name,phone
Hilton Chambers,+44 161 236-4414
Sachas Hotel,None
The Mitre Hotel,+44 161 834-4128
```

7.1.4 Using filter

If we want to somehow change the data on the fly during conversion we can use the filter. There are few filters shipped with `datconv` package, see: *datconv.filters package*. But usually you need to write your own custom filter. For instance imagine that in above described conversion we want to skip records that do not have phone number. We should write folliwing filter:

```
# Standard Python Libs
import logging

# Libs installed using pip
from lxml import etree

# Datconv generic modules
from datconv.filters import SKIP, WRITE, REPEAT, BREAK

Log = None

class DCFilter:
    def filterRecord(self, record):
        tag = record.find('.//phone')
        if tag is not None and tag.text != 'None':
            return WRITE
        else:
            return SKIP
```

and save it as file `with_phone.py` in folder `custom` created where we run `datconv` program. In addtion we have to create empty file `__init__.py` in this folder (to make it valid Python package) and add following key to conversion configuration file:

```
Filter:
    Module: custom.with_phone
```

Then when you run conversion, you will get expected result:

```
name,phone
Hilton Chambers,+44 161 236-4414
The Mitre Hotel,+44 161 834-4128
```

Note that current folder is automatically added to Python search path by `datconv` script.

7.1.5 Concatenating several filters

If we have a library of generic filters and woud like to use few of them in one data conversion run it is possible with provided `pipe` filter. E.g. following configuration will use above filter and standard filter that will remove `name` field from the output records:

```
Filter:
    Module: datconv.filters.pipe
    CArg:
        flist:
            - Module: custom.with_phone
            - Module: datconv.filters.delfield
              CArg:
                  field: [name]
```

7.1.6 Populating database

Let say we have output file from: *JSON to XML conversion example* (file `cb.json.xml`) that we would like to import to SQLite database. At first we have to create table with respective fields to store data. Connector *datconv.outconn.sqlite.ddl module* can be helpfull here. Let's run datconv with following configuration:

```
Reader:
    Module: datconv.readers.dcxml
    CArg:
        rectags: [results]
    PArg:
        inpath: ./sampl.xml

Writer:
    Module: datconv.writers.dcxpaths
    CArg:
        add_header: false
        add_type: true
        ignore_rectyp: true

OutConnector:
    Module: datconv.outconn.sqlite.ddl
    CArg:
        path: ./sampl.sql
        table: sample
```

This will produce file `sampl.sql` with proposed table definition:

```
CREATE TABLE sample (
  name TEXT,
  phone TEXT
);
```

Edit this definition (possibly define primary key etc.) and run with your database to create table. This step may look somewhat like art-for-art in this simple sample, but with large number of fields or tables may save you some typing or copy/pasting.

Then, change Writer and Connector in your configuration file in following way (place path to your SQLite database file as `connstring:` parameter):

```
Reader:
    Module: datconv.readers.dcxml
    CArg:
        rectags: [results]
    PArg:
        inpath: ./sampl.xml

Writer:
    Module: datconv.writers.dcjson
    CArg:
        ignore_rectyp: true

OutConnector:
    Module: datconv.outconn.sqlite.jinsert
    CArg:
        connstring: ./sampl.sqlite
        table: sample
```

and run datconv again to insert records into the table. Note use of `ignore_rectyp` option to get rid of outer `<results>` tag which otherwise would be interpretted as the only one column in the table. In case of problems with inserts one can add `dump_sql` option to Connector in order to dump generated INSERT statements to file or change OutConnector to plain file to check generated JSON.

7.1.7 Default Configuration

It may happen that we have some typical files or typical conversion scenario that we frequently use. In such case it is reasonable to use *Default Configuration* file. Let say we frequently do conversion described in section *JSON to XML conversion example*. To simplify program usage we may save configuration file for this case as the file `.datconv_def.yml` located in root of our home folder making one change in the file: replacing file names with positional arguments, like below:

```
Reader:
    Module:  datconv.readers.dcijson_keys
    CArg:
        headkeys: [requestID, signature]
        reckeys: [results]
        footkeys: [status, metrics]
    PArg:
        inpath:  $1
        outpath: $2

Writer:
    Module: datconv.writers.dcxml
    CArg:
        pretty: true
```

After that one may call datconv in following way:

```
datconv def <imput file> <output file>
```

to perform conversion according to saved schema.

7.1.8 More examples

More examples are contained in package `datconv_test` avaialble from PyPi.

7.2 Default Configuration

Default Configuration file is a file with specific name: `.datconv_def.yml` located in root of user home folder. So in typical installations it this the file:

System	Standard Datconv default configuration file
Linux	`/home/<login>/.datconv_def.yml`
Windows	`C:\Users\<login>\.datconv_def.yml`
OS X	`/Users/<login>/.datconv_def.yml`

Existance of this file is optional. Datconv during startup checks if the file is present. If is present it is read and interpretted as YAML file. The keys in this file are being merged with keys in main configuration file (file passed as first positional argument during datconv invocation from shell). This merge is being done under following rules:

1. Keys from main configuration file takes presedence. I.e. keys from default configuration file are used only if there is no equivalent key in main configuration file.
2. If main configuration file declare different `Module:` that default configuration file, then respective (from the same component) `CArg:` key from default configuration file is completly discarded.
3. If there is '=' character before main configuration file name specified in shell, than default configuration file is not baing read and discarded ('=' character is stripped from main configuration file name).
4. If main configuration file is declared as 'def' then it means that there is no main configuration file, and all settings are taken from default configuration file.
5. If option is not present neither in main nor default configuration file, then value listed as 'defaut' in documentation (class constructor default value or value given as default in conf_template.yaml file) is used.

The syntax of default configuration file is the same as main configuration: *datconv program*.

Defaults configuration file can be used in one of follwing ways:

- not used at all. Clear configuration, no hidden logic.
- used as only one configuration file. User allways edit default configuration file, and then run `datconv def <in> <out>` from folder where data fiels are without worying of path to configuration file.
- mixed mode. E.g. logger configuration or `DefLogLevel:` kay can be specified in default configuration file and the rest in main configuration file.

Note:

- When Datconv is called directly from Python (as `Datconv.Run(conf)` invocation) then default configuration file is not used.
- When DEBUG logging level is emabled, Datconv logs its run configuration (after merging).

7.3 The Datconv API reference

7.3.1 datconv program

The **`datconv`** script has following call syntax:

```
datconv [=]conf_file [--key1:val [--key2:val ...]] [arg1 [arg2 ...]]
where:
conf_file - path to file in YAML format in which Reader, Filter and Writer compoments␣
↪are configured.
            See below listing for more detailed desctiption of this file.
            If there is '=' before conf_file then default configuration file is not␣
↪used.
            If conf_file is equal to 'def' than only default configuration file is␣
↪used.
--key1:val - any number of arguments that add new settings or overwrite settings from␣
↪conf_file.
            It works this way: let say that in conf_file we have:
            Writer:
                Module: datconv.writers.dcxml
                CArg:
                    pretty:   true
            by invoking option --Writer:CArg:pretty:false we overwrite 'pretty'␣
↪option of Writer.
```

(continues on next page)

(continued from previous page)

```
            Note that in YAML file we must have space after : at end of the key,␣
↪while in command line there are no spaces.
arg1 - any number of arguments (that do not begin with --).
    Those arguments will replace $1, $2, ... markers in conf_file according to their␣
↪position in command line:
    i.e. $1 will be replaced by first argument that do not begin with --, etc.
or
datconv --default[-raw]
which prints path to and contents of default configuration file;
if post-fix -raw is used configuration file is printed as it is (without parsing).

or
datconv --version
which prints version number to standard output and exit.

or
datconv --help
which prints short usage information

The datconv script returns to shell:
    0 on sucess
    1 on general error (exception)
    2 on invalid command parameters
    3 on user break (Ctrl-C)
```

Sample main YAML configuration file layout:

```
# Major configuration file for datconv script.
# It must follow YAML syntax and has at least 2 obligatory top level keys: Reader,␣
↪Writer.
#
# Note that speficied modules must be loadable from datconv script, i.e.
#   packages must be placed directly in folder from which datconv script is run (its␣
↪current folder)
#   or datconv must be wrapped in script that define PYTHONPATH environmental␣
↪variable which point to packages root folder
#   or packages root folder must be added to Python configuration as folder with␣
↪packages
#   every package to be loadable must have (even empty) file named __init__.py
#
# The keys listed below are samle keys, for full list of available options see conf_
↪template.yaml files
# contained in readers, writers and filters folders.

# Obligatory key that specify Reader module
Reader:
    # Obligatory key that specify Python module which implements Reader
    # Module must define class DCReader, which must follow interface specification␣
↪described in readers/_skeleton.py
    Module: datconv.readers.dcxml
    # Optional or not (depends on configured reader) key that specify DCReader class␣
↪constructor parameters
    CArg:
        # Here follows DCReader class constructor parameters
        # Concrete keys depands on choosen Reader.
        # See particular readers' documentation.
        # Note that if default values are good than they mey be omitted.
```

(continues on next page)

(continued from previous page)

```
        # However it is not allowed to have any extra keys not specified in reader␣
↪documentation
        # If you want to preserve some keys for future use - outcomemnt them
        encoding:  utf-8
        # ...

    # Usually obligatory key that specify parameters for DCReader.Process method
    PArg:
        # Here follows DCReader.Process parameters
        # Concrete keys depands on choosen Reader.
        # See particular readers documentation.
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.xml
        # optional - if not defined OutConnector is used
        #outpath: out/AddnDrawNbrs_c5019_s38.xml
        # ...

# Obligatory key that specify Writer module
Writer:
    # Obligatory key that specify Python module which implements Writer
    # Module must define class DCWriter, which must follow interface specification␣
↪described in writers/_skeleton.py
    Module: datconv.writers.dcxml
    # Optional or not (depends on configured writer) key that specify DCWriter class␣
↪constructor parameters
    CArg:
        # Here follows DCWriter class constructor parameters
        # Concrete keys depands on choosen Writer.
        # See particular writers' documentation.
        encoding: utf-8

# Optional key that specify Filter module
# If it is missing or null no filter is used
# default: null
Filter:
    # If Filter is defined, this key is obligatory and specify Python module which␣
↪implements Filter
    # Module must define class DCFilter, which must follow interface specification␣
↪described in filters/_skeleton.py
    Module: datconv.filters.stat
    # Optional or not (depends on configured filter) key that specify DCFilter class␣
↪constructor parameters
    CArg:
        # Here follows DCFilter class constructor parameters
        # Concrete keys depands on choosen Filter.
        # See particular filters' documentation.
        retval: 0
        # ...

# Optional key that specify Filter module
# If it is missing or null datconv.outconn.dcfile(Reader:PArg:outpath) is used
# default: null
OutConnector:
    Module: datconv.outconn.dcfile
    CArg:
        # relative or absolute path to output file; obligatory
        path: "out/AcctAgentAdjustment_c5019_s38.xml"
```

(continues on next page)

(continued from previous page)

```
# Optional key that specify Logger class configuration
# If it is missing or null following configuration is used:
#   all log messages are being sent to standard error stream
#   messages of severity below WARNING are discarded
# If this key value is a string it means that Logger class is
#   inherited from calling code by invoking logging.getLogger('XXX').getChild('datconv
↪')
#   where XXX is the key value (name of parent logger).
# If this key value is dictionary (i.e. key contains bubkeys)
#   it directly specify loger configuration or redirects to other file (as described␣
↪below).
# default: null
Logger:
    # This key allows to redirect logger configuration to other file.
    Conf: Logger.yaml

    # As an alternative, all keys contained in Logger.yaml file may be explicitly␣
↪placed here (as subkeys of Logger key)

# Optional key that specify log level of default console logger used when Logger: key␣
↪is not present.
# Possible values: CRITICAL, ERROR, WARNING, INFO, DEBUG
# default: INFO
DefLogLevel: WARNING
```

See also:

- *Readers configuration keys*
- *Filters configuration keys*
- *Writers configuration keys*
- *Output connectors configuration keys*

7.3.2 datconv package

This module provides Datconv class which encapsulate all datconv program features and can be created and called from other Python script in one of following ways:

```
from datconv import Datconv
dc = Datconv()
conf = {...}
dc.Run(conf)
```

Or:

```
from datconv import Datconv
dc = Datconv()
conf = {...}
for rec in dc.Iterate(conf):
  print ('Field1: %s' % str(rec['Field1']))
```

class datconv.**Datconv**

Bases: object

Instead of calling datconv from command line, one can create instance of this class inside other Python script and call its Run() method.

Run(*conf*)
Method that runs conversion process.

Parameters **conf** – is a `dict()` object with keys as apecified by datconv main YAML configuration file. In this case datconv default configuration file is not used.

Returns 2 in case of invalid configuration; 0 if run sucessfully; may throw exception

Iterate(*conf*)
Method that runs conversion process in iteration mode - i.e. every output record is being returned to caling loop.

Parameters **conf** – is a `dict()` object with keys as apecified by datconv main YAML configuration file. In this case datconv default configuration file is not used.

GetHeader()
Returns Header assiciated with data. Method intendent to be used with iteration interface.

GetFooter()
Returns Footer assiciated with data. Method intendent to be used with iteration interface.

Version(*ext_module=None*, *ext_verobj='__version__'*)
If ext_module is None method returns datconv version. Otherwise it loads ext_module module and returns its ext_verobj object (`__version__` by default).

7.3.3 datconv sub-packages

datconv.readers package

This package contains datconv compatible Reader classes for common formats of files used in public.

Reader interface

This module contain Datconv Reader skeleton class suitable as starting point for new readers.

class datconv.readers._skeleton.**DCReader**
Bases: object

This class must be named exactly DCReader. It is responsible for:

- reading input data (i.e. every reader class assumes certain input file format)
- driving entire data conversion process (i.e. main processing loop in implemented in this class)
- determine internal representation of header, records and footer (this strongly depands on reader and kind of input format).

Additional constructor parameters may be added to constructor, but they all have to be named parameters. Parameters are usually passed from YAML file as subkeys of Reader:CArg key.

setWriter(*writer*)
Obligatory method that must be defined in Reader class. It is called by main datconv.py script after it read configuration file and create Writer class.

Parameters **writer** – is instance of Writer class.

setFilter(*flt*)
Obligatory method that must be defined in Reader class. It may be called by main datconv.py script after it read configuration file and create Filter class. If Filter is not configured this method is not called.

Parameters **flt** – is instance of Filter class.

Process(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)
Main method that drive all data conversion process. Parameters are usually passed from YAML file as subkeys of Reader:PArg key. Parameters given in this method are typical ones, however thay may be customized. Usually some kind of input and output path should be passed here. Also if structure of input data format allows for it, it is recommended to implement reading data from certain to certain record number.

Iterate(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)
Clone of Process method which will yield value returned from Writer. Parameters are usually passed from YAML file as subkeys of Reader:PArg key. Parameters given in this method are typical ones, however thay may be customized. Usually some kind of input and output path should be passed here. Also if structure of input data format allows for it, it is recommended to implement reading data from certain to certain record number.

datconv.readers.dcxml module

This module implements Datconv Reader which reads data from XML file.

exception datconv.readers.dcxml.**FilterBreak**
Bases: Exception

Exception class to support Reader.process break isued from Filter class.

exception datconv.readers.dcxml.**ToLimitBreak**
Bases: Exception

Exception class to support Reader.process break caused by reaching configured record limit.

class datconv.readers.dcxml.**ContentGenerator**(*bratags*, *headtags*, *rectags*, *foottags*, *wri*, *flt=None*, *lp_step=0*, *rfrom=1*, *rto=0*)
Bases: xml.sax.handler.ContentHandler

This class handles XML events generated by parser created by xml.sax.make_parser(). It implements most of the functionality of this XML Reader. See documentation of its base class for description of methods meaning.

See description of DCReader constructor and Process() method for meaning of most parameters.

startDocument()
Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for setDocumentLocator).

endDocument()
Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

startElement(*name*, *attrs*)
Signals the start of an element in non-namespace mode.

The name parameter contains the raw XML 1.0 name of the element type as a string and the attrs parameter holds an instance of the Attributes class containing the attributes of the element.

endElement(*name*)
Signals the end of an element in non-namespace mode.

The name parameter contains the name of the element type, just as with the startElement event.

characters(*content*)
Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

class datconv.readers.dcxml.**DCReader**(*bratags=[]*, *headtags=[]*, *rectags=[]*, *foottags=[]*, *log_prog_step=0*)

Bases: object

This Datconv XML Reader class uses xml.sax parser to read and interpret XML file. This parser uses ContentGenerator class from this module to handle XML events. See documentation of standard Python xml.sax library for more information how it works. This Reader assumens that srtucture of input XML file is following:

- there is/are some (one or more) BRACE tag(s); entire document content is included in this/those brace tag(s); well-formed XML document should have at least one such tag;
- then there is/are some optional HEAD tag(s); head tags begin and end completly before record tags begin;
- then there are RECORD tags; everything what is inside record tags is treated as record data and is being passed to Filter and Writer; record tags can not be nested - every record tag must end before another record tag begin; there may be several kinds (names) or record tags - in such case we say that we have multiply record types. If list of record tags is empty then every tag which is one level under brace tag and which is not head nor foot tag is treated as record tag.
- then there is/are some optional FOOTER tag(s); footer tags begin and end completly after record tags;

Constructor parameters explicitly list which tags are of what kind.

TODO: The text inside brace, header and footer tags is discarded (only attributes are passed to Writer).

TODO: The header tags between record tags are discarded (only ones before first record tag are passed to Writer.

TODO: This class does not support CDATA inside XML.

Parameters are usually passed from YAML file as subkeys of Reader:CArg key.

Parameters

- **bratags** – list of tag names that will be treated as brace tags (see above).
- **headtags** – list of tag names that will be treated as header tags (see above).
- **rectags** – list of tag names that will be treated as record tags (see above).
- **foottags** – list of tag names that will be treated as footer tags (see above).
- **log_prog_step** – log info message after this number of records or does not log progress messages if this key is 0 or logging level is set to value higher than INFO.

For more detailed descriptions see *Configuration keys*.

Process(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)
Parameters are usually passed from YAML file as subkeys of Reader:PArg key.

Parameters

- **inpath** – Path to input file.

- **outpath** – Path to output file passed to Writer (fall-back if output connector is not defined).
- **rfrom-rto** – specifies scope of records to be processed.

For more detailed descriptions see *Configuration keys*.

datconv.readers.dcijson_events module

This module implements Datconv Reader which reads data from JSON file.

exception datconv.readers.dcijson_events.**FilterBreak**
Bases: Exception

Exception class to support Reader.process break isued from Filter class.

exception datconv.readers.dcijson_events.**ToLimitBreak**
Bases: Exception

Exception class to support Reader.process break caused by reaching configured record limit.

class datconv.readers.dcijson_events.**DCReader**(*mode=3*, *rec_tag='rec'*, *log_prog_step=0*, *backend=None*)
Bases: object

This Datconv Reader class is utility class to help discover structure of JSON data file. It returns events generated by Python ijson JSON files parser.

Example records returned by this reader (mode == 3):

Input:

```
{
  "PadnDrawNbrs": {
    "cdc": 5019,
    "product": "addn"
  }
}
```

Output (*datconv.writers.dccsv module*):

```
prefix , event , value
item , start_map , None
item , map_key , PadnDrawNbrs
item.PadnDrawNbrs , start_map , None
item.PadnDrawNbrs , map_key , cdc
item.PadnDrawNbrs.cdc , number , 5019
item.PadnDrawNbrs , map_key , product
item.PadnDrawNbrs.product , string , addn
item.PadnDrawNbrs , end_map , None
item , end_map , None
```

Usage instructions of ijson package:

- https://pypi.python.org/pypi/ijson/
- http://softwaremaniacs.org/blog/2010/09/18/ijson/en/
- http://explique.me/Ijson/

Parameters are usually passed from YAML file as subkeys of Reader:CArg key.

Parameters

- **mode** – returns: 1-only unique prefixes; 2-unique (prefix,event) pairs; 3-all events (including data).
- **rec_tag** – name or tag to be placed as record marker.
- **log_prog_step** – log info message after this number of records or does not log progress messages if this key is 0 or logging level is set to value higher than INFO.
- **backend** – backend used by ijson package to parse json file, possible values:

`yajl2_cffi` - requires `yajl2` C library and `cffi` Python package to be installed in the system;

`yajl2` - requires `yajl2` C library to be installed in the system;

None - uses default, Python only backend.

For more detailed descriptions see *Configuration keys*.

Process(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)

Parameters are usually passed from YAML file as subkeys of `Reader:PArg` key.

Parameters

- **inpath** – Path to input file.
- **outpath** – Path to output file passed to Writer (fall-back if output connector is not defined).
- **rfrom-rto** – specifies scope of records to be processed.

For more detailed descriptions see *Configuration keys*.

datconv.readers.dcijson_keys module

This module implements Datconv Reader which reads data from JSON file.

exception datconv.readers.dcijson_keys.**FilterBreak**

Bases: `Exception`

Exception class to support Reader.process break isued from Filter class.

exception datconv.readers.dcijson_keys.**ToLimitBreak**

Bases: `Exception`

Exception class to support Reader.process break caused by reaching configured record limit.

class datconv.readers.dcijson_keys.**DCReader**(*headkeys=[]*, *reckeys=[]*, *footkeys=[]*, *log_prog_step=0*, *backend=None*)

Bases: `object`

This Datconv JSON Reader class uses ijson sax-type parser to read and interpret JSON file. It assumes that input file contain array of json objects and data records are values of some key(s) inside those objects.

Example (headkeys = [], reckeys = [], footkeys = []):

Input:

```
[
{
  "PadnDrawNbrs": {
    "cdc": 5019,
    "product": "addn"
```

(continues on next page)

(continued from previous page)

```
  }
},
{
  "SiteData": {
    "siteId": 38
  },
  "rec0Control": {
    "curDraw": 5
  }
}
]
```

Output (*datconv.writers.dcxml module*):

```
<Datconv>
<PadnDrawNbrs>
    <cdc>5019</cdc>
    <product>addn</product>
</PadnDrawNbrs>
<SiteData>
    <siteId>38</siteId>
</SiteData>
<rec0Control>
    <curDraw>5</curDraw>
</rec0Control>
</Datconv>
```

Parameters are usually passed from YAML file as subkeys of `Reader:CArg` key.

Parameters

- **`headkeys`** – list of key names that will be passed to Writer as header.
- **`reckeys`** – list of key names that will be treated as records. If empty all highest level keys that are not heders or footers are passed to Writer as records.
- **`footkeys`** – list of key names that will be passed to Writer as footer.
- **`log_prog_step`** – log info message after this number of records or does not log progress messages if this key is 0 or logging level is set to value higher than INFO.
- **`backend`** – backend used by ijson package to parse json file, possible values:

  `yajl2_cffi` - requires `yajl2` C library and `cffi` Python package to be installed in the system;

  `yajl2` - requires `yajl2` C library to be installed in the system;

  None - uses default, Python only backend.

For more detailed descriptions see *Configuration keys*.

`Process`(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)

Parameters are usually passed from YAML file as subkeys of `Reader:PArg` key.

Parameters

- **`inpath`** – Path to input file.
- **`outpath`** – Path to output file passed to Writer (fall-back if output connector is not defined).

- **rfrom-rto** – specifies scope of records to be processed.

For more detailed descriptions see *Configuration keys*.

datconv.readers.dcijson module

This module implements Datconv Reader which reads data from JSON file.

exception datconv.readers.dcijson.**FilterBreak**
Bases: Exception

Exception class to support Reader.process break isued from Filter class.

exception datconv.readers.dcijson.**ToLimitBreak**
Bases: Exception

Exception class to support Reader.process break caused by reaching configured record limit.

class datconv.readers.dcijson.**DCReader**(*rec_tag='rec'*, *log_prog_step=0*, *backend=None*)
Bases: object

This Datconv JSON Reader class uses ijson sax-type parser to read and interpret JSON file. It assumes that input file contain array of json objects and every such object is passed as record to Writer. This Reader passes always empty header and footer to Writer.

Example:

Input:

```
[
{
  "PadnDrawNbrs": {
    "cdc": 5019,
    "product": "addn"
  }
},
{
  "SiteData": {
    "siteId": 38
  },
  "rec0Control": {
    "curDraw": 5
  }
}
]
```

Output (*datconv.writers.dcxml module*):

```
<Datconv>
<rec>
    <PadnDrawNbrs>
        <cdc>5019</cdc>
        <product>addn</product>
    </PadnDrawNbrs>
</rec>
<rec>
    <SiteData>
        <siteId>38</siteId>
    </SiteData>
```

(continues on next page)

(continued from previous page)

```
    <rec0Control>
        <curDraw>5</curDraw>
    </rec0Control>
</rec>
</Datconv>
```

Parameters are usually passed from YAML file as subkeys of `Reader:CArg` key.

Parameters

- **rec_tag** – name or tag to be placed as record marker.
- **log_prog_step** – log info message after this number of records or does not log progress messages if this key is 0 or logging level is set to value higher than INFO.
- **backend** – backend used by ijson package to parse json file, possible values:

  `yajl2_cffi` - requires `yajl2` C library and `cffi` Python package to be installed in the system;

  `yajl2` - requires `yajl2` C library to be installed in the system;

  None - uses default, Python only backend.

For more detailed descriptions see *Configuration keys*.

Process(*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)

Parameters are usually passed from YAML file as subkeys of `Reader:PArg` key.

Parameters

- **inpath** – Path to input file.
- **outpath** – Path to output file passed to Writer (fall-back if output connector is not defined).
- **rfrom-rto** – specifies scope of records to be processed.

For more detailed descriptions see *Configuration keys*.

datconv.readers.dccsv module

This module implements Datconv Reader which reads data from CSV file.

class `datconv.readers.dccsv.`**DCReader**(*columns='item'*, *strip=False*, *csv_opt=None*)

Bases: `object`

This module implements Datconv Reader which reads data from CSV file.

Parameters are usually passed from YAML file as subkeys of Reader:CArg key.

Parameters

- **columns** – this parameter may be one of 3 possible types:

  if it is positive number, it specifies line number in input file that stores column names.

  if it is a list, it directly specifies column names in input file.

  if it is string it stands for column name prefix, i.e. columns will have names <prefix>1, <prefix>2, ...
- **strip** – if True, strips white spaces from values

- **csv_opt** – dictionary with csv writer options. See documentation of csv standard Python library. If None, Reader tries to recognize format using `csv.Sniffer` class.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Process (*inpath*, *outpath=None*, *rfrom=1*, *rto=0*)

Parameters are usually passed from YAML file as subkeys of `Reader:PArg` key.

Parameters

- **inpath** – Path to input file.
- **outpath** – Path to output file passed to Writer (fall-back if output connector is not defined).
- **rfrom-rto** – specifies scope of records to be processed.

For more detailed descriptions see *Configuration keys*.

Configuration keys

Listing of all possible configuration keys to be used with readers contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
Reader:
    Module: datconv.readers.dcxml
    CArg:
        # List of tag names that will be treated as brace tags (see class description␣
↪in source or pydoc).
        # default: []
        bratags:  [PadnDrawNbrs]

        # List of tag names that will be treated as header tags (see class␣
↪description in source or pydoc).
        # Note: tags listed here will be placed in header passed to Writer; header␣
↪tags not listed here will be silently skipped.
        # default: []
        headtags: [SiteData, rec0Control]

        # List of tag names that will be treated as record tags (see class␣
↪description in source or pydoc).
        # If list of record tags is empty then every tag which is one level under␣
↪brace tag and which is not head nor foot tag is treated as record tag.
        # default: []
        rectags:  [Gampdf_winNbrs]

        # List of tag names that will be treated as footer tags (see class␣
↪description in source or pydoc).
        # Note: tags listed here will be placed in footer passed to Writer; footer␣
↪tags not listed here will be silently skipped.
        # default: []
        foottags: []

        # Log info message after this number of records
        # If this value is zero no progress logging is done.
        # default: 0
        log_prog_step: 10000
```

(continues on next page)

(continued from previous page)

```
    PArg:
        # Path to input file
        # Obligatory parameter
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.xml

        # Path to output file passed to Writer (fall-back if output connector is not␣
↪defined)
        # default: none (use defined output connector)
        outpath: out/AddnDrawNbrs_c5019_s38.xml

        # Start passing records to Filter and Writer from this record
        # default: 1
        rfrom:    1

        # Stop process on this record; if zero, process up to last record.
        # default: 0
        rto:      20

Reader:
    Module: datconv.readers.dcijson_events
    CArg:
        # Returns: 1-only unique prefixes; 2-unique (prefix,event) pairs; 3-all␣
↪events (including data).
        # default: 3
        mode: 3

        # Name or tag to be placed as record marker.
        # default: rec
        rec_tag: rec

        # Log info message after this number of records
        # If this value is zero no progress logging is done.
        # default: 0
        log_prog_step: 10000

        # Backend used by ijson package to parse json file, possible values:
        # - yajl2_cffi: requires yajl2 C library and cffi Python packege to be␣
↪installed in the system;
        # - yajl2 - requires yajl2 C library to be installed in the system;
        # - null - uses default, Python only backend.
        # default: null
        backend: yajl2_cffi

    PArg:
        # Path to input file
        # Obligatory parameter
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.json

        # Path to output file passed to Writer (fall-back if output connector is not␣
↪defined)
        # default: none (use defined output connector)
        outpath: out/AddnDrawNbrs_c5019_s38.json

        # Start passing records to Filter and Writer from this record
        # default: 1
        rfrom:    1
```

(continues on next page)

(continued from previous page)

```
        # Stop process on this record; if zero, process up to last record.
        # default: 0
        rto:      20

Reader:
    Module: datconv.readers.dcijson_keys
    CArg:
        # List of key names that will be passed to Writer as header.
        # default: []
        headkeys: [SiteData, rec0Control]

        # List of key names that will be treated as records. If empty all highest␣
↪level keys that are not heders or footers are passed to Writer as records.
        # default: []
        reckeys:  [Gampdf_winNbrs]

        # List of  key names that will be passed to Writer as footer.
        # default: []
        footkeys: []

        # Log info message after this number of records
        # If this value is zero no progress logging is done.
        # default: 0
        log_prog_step: 10000

        # Backend used by ijson package to parse json file (see above):
        # default: null
        backend: yajl2_cffi

    PArg:
        # Path to input file
        # Obligatory parameter
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.json

        # Path to output file passed to Writer (fall-back if output connector is not␣
↪defined)
        # default: none (use defined output connector)
        outpath: out/AddnDrawNbrs_c5019_s38.json

        # Start passing records to Filter and Writer from this record
        # default: 1
        rfrom:    1

        # Stop process on this record; if zero, process up to last record.
        # default: 0
        rto:      20

Reader:
    Module: datconv.readers.dcijson
    CArg:
        # Name or tag to be placed as record marker.
        # default: rec
        rec_tag: rec

        # Log info message after this number of records
        # If this value is zero no progress logging is done.
        # default: 0
```

(continues on next page)

(continued from previous page)

```
        log_prog_step: 10000

        # Backend used by ijson package to parse json file (see above):
        # default: null
        backend: yajl2_cffi

    PArg:
        # Path to input file
        # Obligatory parameter
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.json

        # Path to output file passed to Writer (fall-back if output connector is not␣
↪defined)
        # default: none (use defined output connector)
        outpath: out/AddnDrawNbrs_c5019_s38.json

        # Start passing records to Filter and Writer from this record
        # default: 1
        rfrom:    1

        # Stop process on this record; if zero, process up to last record.
        # default: 0
        rto:      20

Reader:
    Module:  datconv.readers.dccsv
    CArg:
        # this parameter may be one of 3 possible types:
        #    if it is positive number, it specifies line number in input file that␣
↪stores column names.
        #    if it is a list, it directly specifies column names in input file.
        #       Specified names must be possible to use as XML tag names.
        #    if it is string it stands for column name prefix, i.e. columns will have␣
↪names <prefix>1, <prefix>2, ...
        # default: 'item'
        columns: 1

        # if True, strips white spaces from values
        # default: false
        strip: true

        # Python csv writer class constructor options. See documantation of csv␣
↪standard Python library.
        # Caution: Escape characters must be contained in double quotes ('\n' will␣
↪not work).
        # If null, Reader tries to recognize format using csv.Sniffer class.
        # default: null
        csv_opt:
            lineterminator: "\n"

    PArg:
        # Path to input file
        # Obligatory parameter
        inpath:  ../GET-Data/cdc_5019/AddnDrawNbrs_c5019_s38.csv

        # Path to output file passed to Writer (fall-back if output connector is not␣
↪defined)
```

(continues on next page)

(continued from previous page)

```
        # default: none (use defined output connector)
        outpath: out/AddnDrawNbrs_c5019_s38.json

        # Start passing records to Filter and Writer from this record
        # default: 1
        rfrom:    1

        # Stop process on this record; if zero, process up to last record.
        # default: 0
        rto:      20
```

datconv.filters package

Common data or structures to be used in Datconv Filters.

datconv.filters.**SKIP = 0**
Name for zero, when Filter returnes this value record is skipped.

datconv.filters.**WRITE = 1**
When Filter returns this bit, record is being posted to Writer.

datconv.filters.**REPEAT = 2**
When Filter returns this bit, other bits are checked and flow is again returned to Filter function with the same record. This bit is used to generate/produce data.

datconv.filters.**BREAK = 4**
When Filter returns this bit, impot process is breaken and DCWriter.writeFooter function is being called.

Filter interface

This module contain Datconv Filter skeleton class suitable as starting point for new filters.

class datconv.filters._skeleton.**DCFilter**
Bases: object

This class must be named exactly DCFilter. It is being called after Reader read record and before Writer write record. It is able to:

- filter data (i.e. do not pass certain records further - i.e. to writer for placing in output)
- change data (i.e. change on the fly contents of certain records)
- produce data (i.e. cause that certain records, maybe slightly modified, are being sent multiply times to writer)
- break conversion process (i.e. caused that conversion stop on certain record).

Additional constructor parameters may be added to this method, but they all have to be named parameters. Parameters are usually passed from YAML file as subkeys of Filter:CArg key.

setHeader(*header*)
Facultative method that may be defined in Filter class. Informs Filter about contents of header and give it a chance to change it. If this method is present in Filter it is called by Reader before data conversion begins and before Writer calls writeHeader.

Parameters **header** – is instance of header as passed by Reader (always a list, but type of elements is up to Reader). This parameter is passed later to Writer.

filterRecord(*record*)

Obligatory method that must be defined in Filter class. It is called to perform filter tasks described above.

Parameters **record** – is instance of root XML model of record returned by Reader (class of lxml.etree.ElementTree).

This method may check or manipulate contents of record.

There are several ways to access already known data from current record, e.g.:

record.tag - the name of root tag (i.e. record name).

record.find(xpath) - returns first found (or None) record's sub-tag using relative, simplified xpath.

e.g. record.find('.//TIME') - searches record tree and returns first found <TIME . . . /> tag.

record.findtext(xpath) - as above but returns .text attribute (see below) of found tag (or raise Exception if tag is not found)..

record.xpath(xpath) - evaluate full absolute xpath expression on record (i.e. return list of matched tags, or string, number etc. - depands on xpath).

e.g. record.xpath('/Gampdf_winNbrs/winSet') - returns list of all winSet subtags of root Gampdf_winNbrs tag.

On record and also on tags returned by above methods the data associated with tag may be obtained using:

tag.tag - tag name (i.e. field name)

tag.text - text that is contained between opening and closing tag (usually data value)

tag.keys() - iterable containing tag attbibute names

tag['attrib'] - the value of tag attribute named 'attrib'; raise Exception if tag does not contain 'attrib' atribute.

tag.get('attrib') - as above but returns None if no such attribute.

record.insert(0, newtag) - inserts new tag at begining of record

etree.SubElement(record, 'NEWTAG') - inserts new tag at and of record

See `lxml` package for more documentation.

This method should return combination of following bits:

WRITE - to cause program to pass record to Writer for writting to output

REPEAT - to cause program to call filterRecord with the same record (instead or reading next record from input). This is used to produce / create new records. This option should be used with caution to avoid infinite loop (i.e. Filter should mainain its own replication counter and stop returning REPEAT at some point).

BREAK - to cause program to break process on this record (i.e. Reader will not read next record). In case when REPEAT | BREAK is returned, the REPEAT bit takes precedence.

or return SKIP (0) - what will cause that record will be skiped (will not be passed to Writer).

setFooter(*footer*)

Facultative method that may be defined in Filter class. Informs Filter about contents of footer and give it a chance to change it. If this method is present in Filter it is called by Reader after data conversion and before Writer calls `setFooter`.

Parameters **footer** – is instance of footer as passed by Reader (always a list, but type of elements is up to Reader). This parameter is passed later to Writer.

datconv.filters.delfield module

General Filter that allows to remove certain fields from record.

class datconv.filters.delfield.**DCFilter**(*field=[]*)
Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters **field** – list of fields to remove. Fields must be in form of XPaths understandable by lxml.etree._Element.find method (relative paths)

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.filters.rectyp module

General Filter that allows to filter out certain record types.

class datconv.filters.rectyp.**DCFilter**(*inclusive=True*, *rectyp=[]*)
Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters

- **inclusive** – if False, record types given in rectyp are excluded, otherwise only rectyp records are included;
- **rectyp** – list of record types (root tags of records).

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.filters.pipe module

General Filter that allows users to run several other filters one after one. Values returned by configured filters' are combined in following way:

- to get record written (sent to Writer) all filters must set WRITE bit
- to get record repeated at leat one filter must set REPEAT bit
- to get process break at leat one filter must set BREAK bit
- REPEAT bit takes precedence over BREAK bit (i.e. if both are set record is re-evaluated)

class datconv.filters.pipe.**DCFilter**(*flist*, *pass_skiped=True*)
Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters

- **flist** – list of filters to be run in chain with their parameters;
- **pass_skiped** – if it is False, records for which some filter returned SKIP will not be passed to next filters;

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.filters.gen_rec module

General Filter that allows to generate new records. Every record is cloned by conifigurable number of times. This Filter is suitable for subclassing if more rebust generation strategies are required.

class datconv.filters.gen_rec.**DCFilter**(*n=1*, *fake_flg=None*)
Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters

- **n** – determines how many clones are generated for every record.
- **fake_flg** – if set (to string) a tag of set name is added to every generated cloned record with the value 1.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.filters.stat module

General Filter that allows to calculate and print required statistics about processed data. Filter prints first record number when given XPath expression is met and number of records in which it is met. Filter prints statistics at program exit as logger INFO messages.

class datconv.filters.stat.**DCFilter**(*retval=1*, *rectyp=True*, *printzero=False*, *fields=[]*)
Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters

- **retval** – value that filter returns (0 to skip records, 1 to write records);
- **rectyp** – if True, record type (root tag) is included into statistics; i.e. it is printed how many records are of particular types.
- **printzero** – if True, not found records (with count 0) are included into summary (except when groupping is used)
- **fields** – list of 2 elements' lists: - first element is absolute XPath expression to make statistics against (lxml.etree._element.xpath method compatible) - second element is a digit: 0 - if we test against element existance (i.e. not None and not []) 1 - if we are grouping against element value 2 - if given XPath expression returns boolean value.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.filters.statex module

General Filter that allows to calculate and print required statistics about processed data - extended version. Filter prints counts or sums of records that fulfill given expression with option to group by certain data. Filter prints statistics at program exit as logger INFO messages or to the file.

class datconv.filters.statex.**DCFilter**(*retval=1*, *fields=[]*, *statfile=None*, *statwriter=None*)

Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Filter:CArg key.

Parameters

- **retval** – value that filter returns (0 to skip records, 1 to write records);
- **fields** – list of 5 or 6 elements' lists that define calculated statistics.
- **statfile** – file to write final statistics
- **statwriter** – datconv writer module to write final statistics

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with filters contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
Filter:
    Module: datconv.filters.rectyp
    CArg:
        # If False, record types given in rectyp are excluded, otherwise only rectyp␣
↪records are included.
        # default: true
        inclusive: true

        # List of record types (root tags of records).
        # default: []
        rectyp: []

Filter:
    Module: datconv.filters.delfield
    CArg:
        # List of fields to remove.
        # Fields must be in form of XPaths understandable by lxml.etree._Element.find␣
↪method (relative paths)
        # default: []
        field: []

Filter:
    Module: datconv.filters.pipe
    CArg:
        # List of filters to be run in chain with their parameters (obligatory␣
↪parameter)
        flist:
            - Module: datconv.filters.rectyp
              CArg:
                  rectyp: []
            - Module: datconv.filters.delfield
              CArg:
                  field: []

        # If it is False, records for which some filter returned SKIP will not be␣
↪passed to next filters
```

(continues on next page)

(continued from previous page)

```
        # default: true
        pass_skiped: true

Filter:
    Module: datconv.filters.gen_rec
    CArg:
        # Determines how many clones are generated for every record.
        # default: 1
        n: 5

        # If set (to string) a tag of set name is added to every generated clone with␣
↪the value 1.
        # default: null
        fake_flg: FAKE

Filter:
    Module: datconv.filters.stat
    CArg:
        # Value that filter returns (0 to skip records, 1 to write records)
        # default: 1
        retval: 0

        # If true, record type (root tag) is included into statistics
        # i.e. it is printed how many records are of particular types.
        # default: true
        rectyp: true

        # If true, not found records (with count 0) are included into summary (except␣
↪when groupping is used)
        # default: false
        printzero: false

        # List of 2 elements' lists:
        # first element is absolute XPath expression to make statistics against
        # (lxml.etree._element.xpath method compatible)
        # second element is a digit:
        # 0 - if we test against element existance (i.e. not None and not [])
        # 1 - if we are grouping against element value
        # 2 - if given XPath expression returns boolean value.
        # default: []
        fields:
            - [/TT_COMMAND/PRODUCT, 1]
            - [/TT_WAGER/PRODUCT, 1]
            - [/TT_WAGER/PRODUCT=7, 2]
            - [/TT_COMMAND/WIN_CDC, 0]

Filter:
    Module: datconv.filters.statex
    CArg:
        # Value that filter returns (0 to skip records, 1 to write records)
        # default: 1
        retval: 1

        # List of 5 or 6 elements' lists:
        # 1st element is statistic name used only in output summary
        # 2nd element is record name for which evaluate statistic, if null - eveluate␣
↪for every record
```

(continues on next page)

(continued from previous page)

```
        # 3rd element is XPath expression or boolean; if it avaluate to non empty␣
↪list, text, non zero numeric or true, statistic is updated;
        #             if it is true statistic is updated unconditionally; if false -␣
↪never updated
        # 4th element is XPath expression used for grouping or null for global (all␣
↪data) grouping
        # 5th element is either 'c' (count) or 's' (sum) small letter
        # 6th element is XPath expression that returns numeric or value castable to␣
↪numeric;
        #             it determines what to sum if 5th element is 's', or has no␣
↪meaning (may be absent) otherwise
        # All XPath expressions must be absolute and lxml.etree._element.xpath method␣
↪compatible.
        # Note: only subset of full XPath specification is currently supported in␣
↪lxml - check your version of this package
        # default: []
        fields:
            - [BAL.WAGCNT, DEFRECBAL, true, number(//PROD_NUM), s, //WAGCNT]
            - [TMF.WAGCNT, TT_WAGER, //_UPDATE_MONEY=1, number(//PRODUCT), c]
            - [BAL.WAGAMT, DEFRECBAL, true, number(//PROD_NUM), s, //WAGAMT]
            - [TMF.WAGAMT, TT_WAGER, //_UPDATE_MONEY=1, number(//PRODUCT), s, number(/
↪/_AMOUNT)]

        # Full path to file to write final statistics
        # default: null
        statfile: /tmp/statfile.xml

        # datconv writer module to write final statistics
        # All datconv compatible Writer modules can be used here.
        # The keys below are the same keys that you normaly have under Writer root␣
↪key in YAML file.
        # This key must be set together with above statfile key.
        # If this key is null final statistics are being sent to configured logger as␣
↪info messages.
        # default: null
        statwriter:
            Module: datconv.writers.dcxml
            CArg:
                pretty: false
```

datconv.writers package

This package contains datconv compatible Writer classes for common formats of files used in public.

Writer interface

This module contain Datconv Writer skeleton class suitable as starting point for new writers.

class datconv.writers._skeleton.**DCWriter**

Bases: object

This class must be named exactly DCWriter. It is responsible for:

- writing data to output file.

Additional constructor parameters may be added to this method, but they all have to be named parameters. Parameters are usually passed from YAML file as subkeys of Writer:CArg key.

setOutput(*out*)
Obligatory method that must be defined in Writer class. It is called by main Datconv.Run() function before conversion begin and before any write* function is being called.

Parameters **out** – is instance of datconv Output Connector class according to configuration file. In case Output Connector is not defined in configuration file there are two fallbacks checked: a) if Reader:PArg:outpath is defined, the file connector with specified path is used, b) standard output stream is used as output.

This method in some rare cases may be called multiply times (e.g. when convering set of files). Initialization of some variables related to output file (like output records counter etc.) should be done here.

writeHeader(*header*)
Obligatory method that must be defined in Writer class. Write header to output file (if it makes sense).

Parameters **header** – is instance of header as passed by Reader (always a list, but type of elements is up to Reader).

writeFooter(*footer*)
Obligatory method that must be defined in Writer class. Write footer to output file (if it makes sense).

Parameters **footer** – is instance of footer as passed by Reader (always a list, but type of elements is up to Reader).

getHeader()
Obligatory method that returnes header passed to `writeHeader`.

getFooter()
Obligatory method that returnes footer passed to `writeFooter`.

writeRecord(*record*)
Obligatory method that must be defined in Writer class. Transform passed record to its specific formet and pass to output connecte=or either as object or string.

Parameters **record** – is instance of lxml.etree.ElementTree class as passed by Reader.

Returns Transformed record that this writer passed to object type connector.

See *Filter interface* and package `lxml` documentation for information how to obtain structure and data from record.

datconv.writers.dccsv module

This module implements Datconv Writer which saves data in form of CSV file. Supports connectors of type: STRING, LIST, ITERABLE.

class datconv.writers.dccsv.**DCWriter**(*columns=None*, *simple_xpath=False*, *add_header=False*, *col_names=True*, *csv_opt=None*)
Bases: `object`

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of Writer:CArg key.

Parameters

- **columns** – this parameter may be one of 4 possible types or None: if it is a string, it should be the path to file that contain specification of columns in output file.

  if it is a list, it directly specifies columns in output file.

if it is a integer, add columns based on first record.

if it is None or dictionary, columns in output CSV file are being generated automatically based on contentents of input file. When this option is used number of columns in different records in CSV file may very because new columns are being added when discovered.

- **simple_xpath** – determines weather simple xpaths are used in column specification. See pdxpath Writer for more descripption.
- **add_header** – if True, generic header (as initialized by Reader) is added as first line of output file.
- **col_names** – if True, line with column names (fields) is added before data or after data (in case of auto option).
- **csv_opt** – dictionary with csv writer options. See documentation of csv standard Python library.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.writers.dcxml module

This module implements Datconv Writer which saves data in form of XML file.

class datconv.writers.dcxml.**DCWriter**(*pretty=True*, *encoding='unicode'*, *cnt_tag=None*, *cnt_attr=None*, *add_header=True*, *add_footer=True*)

Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Writer:CArg key.

Parameters

- **pretty** – this parameter is passed to lxml.etree.tostring function. If True, XML is formated in readable way (one tag in one line), otherwise full record is placed in one line (more compact,suitable for computers).
- **encoding** – this parameter is passed to lxml.etree.tostring function. It determines encoding used in output XML file. See documantation of codecs standard Python library for possible encodings. This parameter is ignored in Python3, where always unicode coding is used.
- **cnt_tag** – tag name to store records count, if not set record count will not be printed in output footer
- **cnt_attr** – attribute of cnt_tag tag to store records count, if not set record count will be printed as tag text
- **add_header** – if True, generic header (as initialized by Reader) is added as first tag of output file.
- **add_footer** – if True, generic footer (as initialized by Reader) is added as last tag of output file.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.writers.dcxpaths module

This module implements Datconv Writer which generates list of fields (tags that have text) in scanned document. This is helper Writer, it generates text file that can be used as configuration file (list of columns) for CSV Writer. It may be

also helpfull if you only want to extract (e.g. to compare) structure of XML file.

Format of output file is following:
Field Name, **Record Name**, **XPath**, **Default Value**
where:
Field Name - the name of tag with text
Record Name - the name of record (root tag) in which this field is contained
XPath - path to tag inside XML structure starting from record root (but not containing record name) in the forn of XPath expression
Default Value - placeholder to place default value, this writer leave it empty
Type - tye of data guessed from data (if add_type option is set)

Generated entries are unique and sorted by Record Name and XPath. Supports connectors of type: STRING, LIST, ITERABLE.

class datconv.writers.dcxpaths.**DCWriter**(*simple_xpath=False*, *ignore_rectyp=False*, *ignore_xpath=False*, *ignore_attr=False*, *add_header=True*, *add_type=False*, *rectyp_separator='_'*, *colno=0*)

Bases: object

Please see constructor description for more details.

Constructor parameters are usually passed from YAML file as subkeys of Writer:CArg key.

Parameters

- **simple_xpath** – if True, Wirter generate xpaths relatative to record tag, and will not generate separate fields for replicated data (repeated tags; arrays) and not generate fields for tag's attributes. The same setting must be applied in CSV Writer if it uses configuration file generated by this Writer.
- **ignore_rectyp** – if True, Writer join fields with the same name contained in different records. Generated Field Name does not contain record name prefix, and in place of record name '*' is placed.
- **ignore_xpath** – if True, Writer join fields with the same name contained in different paths of XML structure. Generated XPath is in form './/FieldName'.
- **ignore_attr** – if True, Writer will not generate fields for XML attbibutes. If simple_xpath is True, this option is automatically set to True.
- **add_header** – add header as first line of output.
- **add_type** – add data type information guessed from data.
- **rectyp_separator** – separator in generated column name between record type and calumn name (has effect if ignore_rectyp = false).
- **colno** – this parameter is for interface compatibility reason, it has no meaning in this class.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

checkXPath(*record*, *ret_new=False*)
Helper function - it scans record and finds new (not already known) xpaths to add to output.

Depending on constructor simple_xpath parameter it calls either _checkXPathSimple or _checkXPath.

resetXPaths()
Reset class internal structures (found xpaths' list).

Typically called in Writer.setOutput when we are about to read new file.

datconv.writers.dcjson module

This module implements Datconv Writer which saves data in form of JSON file. Supports connectors of type: STRING, OBJECT (dict()), ITERABLE.

class datconv.writers.dcjson.**DCWriter**(*add_header=True*, *add_footer=True*, *add_newline=True*, *convert_values=2*, *null_text='None'*, *preserve_order=False*, *text_key='text'*, *text_eliminate=True*, *with_prop=False*, *ignore_rectyp=False*, *json_opt=None*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of Writer:CArg key.

Parameters

- **add_header** – if True, generic header (as initialized by Reader) is added as first object of output file or stream - only in non-iteration mode.
- **add_footer** – if True, generic footer (as initialized by Reader) is added as last object of output file or stream - only in non-iteration mode.
- **add_newline** – if True, adds newline character after each record.
- **convert_values** – 0 - does not convert (all values are text); 1 - tries to convert values to int, bool or float (do not quote in json file) - little slower; 2 - like 1 but in addition checks if int values can be stored in 64 bits, if not place them as string value.
- **null_text** – text that is converted to JSON null value (apply if convert_values is > 0).
- **preserve_order** – if True, order of keys in json output match order in source.
- **text_key** – name of key to store XML text.
- **text_eliminate** – if True, XML text key will be eliminated if there are no other tag components.
- **with_prop** – if True, XML properties are being saved in JSON file.
- **ignore_rectyp** – if True, XML root tag for records (aka record type) will not be saved in JSON file (simplifies output layout in case there is one record type).
- **json_opt** – dictionary with json.dump() options. See documentation of json standard Python library.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with writers contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
Writer:
    Module: datconv.writers.dcxml
    CArg:
        # If True, XML is formated in readable way (one tag in one line),
        # otherwise full record is placed in one line (more compact, suitable for␣
↪computers).
        # default: true
        pretty:   true

        # Determines emcoding used in output XML file.
        # Note that if encoding is set to ascii some characters may be converted to␣
↪XML/HTML compatibe special codes
        # Most raliable way is to set unicode here (default option)
        # See documantation of codecs standard Python library for possible encodings.
        # Note: This option is ignored in Python3, where always unicode coding is␣
↪used what produce UTF-8 XML file.
        # default: utf8 (in Python2), unicode (in Python3 and above)
        encoding: unicode

        # Tag name to store records countin output footer, if not set records count␣
↪will not be printed
        # default: null
        cnt_tag: Footer

        # attribute of cnt_tag tag to store records count, if not set records count␣
↪will be printed as tag text
        # default: null
        cnt_attr: tranCount

        # if True, generic header (as initialized by Reader) is added as first tag of␣
↪output file
        # default: true
        add_header: true

        # if True, generic footer (as initialized by Reader) is added as last tag of␣
↪output file
        add_footer: true

Writer:
    Module: datconv.writers.dcxpaths
    CArg:
        # If true, Wirter generate xpaths relatative to record tag, and will not␣
↪generate
        # separate fields for replicated data (repeated tags; arrays) and not␣
↪generate fields for tag's attributes.
        # The same setting must be applied in writers.dccsv if it uses configuration␣
↪file generated by this Writer.
        # default: false
        simple_xpath: false

        # If true, Writer join fields with the same name contained in different␣
↪records.
        # Generated Field Name does not contain record name prefix, and in place of␣
↪record name '*' is placed.
        # default: false
        ignore_rectyp: false
```

(continues on next page)

(continued from previous page)

```
        # If true, Writer join fields with the same name contained in different paths␣
↪of XML structure.
        # Generated XPath is in form './/FieldName' or '//FieldName' (depands on␣
↪simple_xpath property).
        # default: false
        ignore_xpath: false

        # If true, Writer will not generate fields for XML attbibutes.
        # If simple_xpath is true, this option is automatically set to true.
        # default: false
        ignore_attr: false

        # if True, generic header (as initialized by Reader) is added as first line␣
↪of output file
        # default: true
        add_header: true

        # If true, Writer will add data type information guessed from data.
        # default: false
        add_type: false

        # Separator in generated column name between record type and calumn name (has␣
↪effect if ignore_rectyp = false).
        # default: "_"
        rectyp_separator: "."

Writer:
    Module: datconv.writers.dccsv
    CArg:
        # This parameter specifies columns to be placed in outpot CSV file.
        # It may be one of 4 possible types or null:
        # - string: path to file that contain specification of columns in output file.
        #   This specification may be generated by (or based on file generated by)␣
↪writers.dcxpaths.
        #   See this module description for more details.
        #   Lines that begin with # sign in specification file are ignored.
        # - list: direct specification of columns in output file.
        #   It should be list of 4 element lists.
        #   Those 4 element lists are similar to lines in file specification␣
↪described above.
        #   This option is suitable if we want very few columns
        # - integer: assuming that all records have the same fields, add columns␣
↪based on first record
        # - dictionary or null: this runs writer in so called auto-mode.
        #   In this mode columns in output file are being added automatically as they␣
↪are being fond in input file.
        #   Columns fould in previous records are also placed, so number of columns␣
↪increase with consequtive records.
        #   This is possible to enforce certain number of columns from begin to␣
↪ensure equal number of columns (see below).
        #   In this option column names are added (if configured - see below) at end␣
↪of file.
        # default: null
        columns: out/CZEC8173.TMF.xpaths        # string: path to file that contain␣
↪specification of columns in output file
        # or
        columns:                                # list: direct specification of␣
↪columns in output file
```

(continues on next page)

(continued from previous page)

```
            - ['ISN','*','ISN',null]
            - ['TIME','*','TIME',null]
        # or
        columns: 1                                  # integer: assuming that all records␣
↪have the same fields, adds columns based on first record
                                                    # the number has currently no meaning␣
↪(we advice to place 1 here).
        # or
        columns:                                    # dictionary (auto) case
            ignore_rectyp: false                    # like in writers.pdxpaths (see above)
            ignore_xpath:  false                    # like in writers.pdxpaths (see above)
            ignore_attr:   false                    # like in writers.pdxpaths (see above)
            colno:         160                      # enforce this number of columns from␣
↪begin (they are filled with empty values (default: 0 - i.e. option not active)

        # Determines weather simple xpaths are used in column specification (see␣
↪option description in writers.pdxpaths above).
        # This option actually determines if function lxml.etree.Element.find or .
↪xpath is used (see lxml documentation).
        # find is less capable but about 25% faster than xpath - therefore this␣
↪option.
        # default: false
        simple_xpath: false

        # If True, generic header (as initialized by Reader) is added as first line␣
↪of output file.
        # default: false
        add_header: false

        # If True, line with column names (fields) is added before data or after data␣
↪(in case of auto option).
        # default: true
        col_names: true

        # Python csv writer class constructor options. See documantation of csv␣
↪standard Python library.
        # Caution: Escape characters must be contained in double quotes ('\n' will␣
↪not work).
        # default: null
        csv_opt:
            lineterminator: "\n"

Writer:
    Module: datconv.writers.dcjson
    CArg:
        # If True, generic header (as initialized by Reader) is added as first object␣
↪of output file.
        # default: true
        add_header: true

        # If True, generic footer (as initialized by Reader) is added as last object␣
↪of output file.
        # default: true
        add_footer: true

        # If True, adds newline character after each record.
        # default: true
```

(continues on next page)

(continued from previous page)

```
        add_newline: true

        # 0 - does not convert (all values are text)
        # 1 - tries to convert velues to int, bool or float (do not quote in json␣
↪file) - little slower
        # 2 - like 1 but in addition checks if int values fits in 64 bits, if not␣
↪place them as string value
        # default: 2
        convert_values: 2

        # Text that is converted to JSON null value (apply if convert_values is True)
        # default: 'None'
        null_text: ''

        # If True, order of keys in json output match order in source
        # default: false
        preserve_order: false

        # Name of key to store XML text
        # default: 'text'
        text_key: '_text_'

        # If True, XML text key will be eliminated if there are no other tag␣
↪components
        # default: true
        text_eliminate: true

        # If True, XML properties are being saved in JSON file
        # default: true
        with_prop: true

        # If True, XML root tag for records (aka record type) will not be saved in␣
↪JSON file
        # (simplifies output layout in case there is one record type)
        # default: false
        ignore_rectyp: true

        # Dictionary with json.dump() options. See documentation of json standard␣
↪Python library.
        # default: null
        json_opt:
            indent: 2
```

datconv.outconn package

Common data or structures to be used in Datconv Output Connectors.

datconv.outconn.**STRING = 1**
Bit meeaning that Connector accepts strings.

datconv.outconn.**OBJECT = 2**
Bit meeaning that Connector accepts objects.

datconv.outconn.**LIST = 4**
Bit meeaning that Connector accepts list objects.

datconv.outconn.**ITERABLE = 8**
Bit meeaning that Connector accepts Python iterable objects.

Output Connector interface

This module contain skeleton class suitable as starting point for new Datconv output connectors.

class datconv.outconn._skeleton.**DCConnector**
Bases: object

This class must be named exactly DCConnector. It is being called by Writter in order to write output data. It's main task is to:

- write data to destination storage (which can be file, database, some stream)

Additional constructor parameters may be added to this method, but they all have to be named parameters. Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

supportedInterfases()
Obligatory method that must be defined in Output Connector class. Informs Writer about kind of interface this connector implements. Connector should return one or combination of flags: STRING, OBJECT, LIST, ITERABLE. If is called by Writer before it pass any data to connector.

pushString(*strData*)
Obligatory method that must be defined if Connector returned STRING flag in supportedInterfases() method. It is called by Writer to pass data to be written to output. Connector must write passed data to output. Method does not return any value.

Parameters **strData** – string to be written to output.

getStreams()
Obligatory method that must be defined if Connector returned STRING flag in supportedInterfases() method. It may be called by Writer to obtain output stream to write to (instead of calling pushString. It may happen if Writer uses some library functions that require stream to be passed to. Connector must return array of its output streams (must be a list - typically one element).

Returns list of streams for writing data.

tryObject(*obj*)
Obligatory method that must be defined if Connector returned OBJECT flag in supportedInterfases() method. It is called by Writer before it pass any data to connector (but after supportedInterfases) to check if it is configured with compatible connector. Passed object is of the same type like it will be later passed to pushObject() method. Connector should call isinstance() and return True if object is of right type.

Parameters **obj** – object passed for hands shaking.

Returns boolean value indicating if passed object is of correct type.

pushObject(*obj*)
Obligatory method that must be defined if Connector returned OBJECT flag in supportedInterfases() method. It is called by Writer to pass data to be written to output. Connector must write passed data to output. Method does not return any value.

Parameters **obj** – object to be written to output.

onFinish(*bSuccess*)
Optional method that may be defined. If it is defined it is called by Writer at and of conversion process. Connector may use it to close its streams, commit data or just log proper message. Method does not return any value.

Parameters **bSuccess** – parameter that inform connector if process is going to end sucessfully.

datconv.outconn.dcnull module

This module implements Datconv Output Connector which discards output (like writing to /dev/null).

datconv.outconn.dcstdout module

This module implements Datconv Output Connector which writes data to standard output stream.

datconv.outconn.dcfile module

This module implements Datconv Output Connector which saves data to regular file.

class datconv.outconn.dcfile.**DCConnector**(*path*, *mode='w'*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **path** – relative or absolute path to output file.
- **mode** – output file opening mode.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.dcexcel module

This module implements Datconv Output Connector which writes data to MS Excel (*.xlsx) file. This connector should be used with Writer: *datconv.writers.dccsv module*. It requires Python package `openpyxl` to be installed. It does not require Excel program to be installed.

Note: This is very initial version of the package (beta quality):

- there is no support for cell value types: currently all data are placed as text
- output uses default font, sheet name etc.
- do not use this connector with very large data: all data are kept in memory before baing saved.

---

class datconv.outconn.dcexcel.**DCConnector**(*path*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters **path** – relative or absolute path to output file.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.dcmultiplicator module

This module implements Datconv Output Connector which sends data to multiply connectors.

class datconv.outconn.dcmultiplicator.**DCConnector**(*clist=[]*, *ilist=[]*)
Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **clist** – list of sub-connectiors that will get output records. Every item should be a dictionary with `Module:` key and optional `CArg:` key.
- **ilist** – list of sub-connectiors instances that will get output records. Every item should be a created instance of other output connector - option to use only in code.

Items passed in clist parameter will be instantiated by constructor, items passed in ilist parameter are already live instancies that will be added to sub-connectors list. For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with output connectors contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
OutConnector:
    Module: datconv.outconn.dcnull

OutConnector:
    Module: datconv.outconn.dcstdout

OutConnector:
    Module: datconv.outconn.dcfile
    CArg:
        # relative or absolute path to output file; obligatory
        path: "out/AcctAgentAdjustment_c5019_s38.xml"

        # output file opening mode (w or a); optional
        # default: w
        mode: a

OutConnector:
    Module: datconv.outconn.dcexcel
    CArg:
        # relative or absolute path to output file; obligatory
        path: "out/AcctAgentAdjustment_c5019_s38.xlsx"

OutConnector:
    Module: datconv.outconn.dcmultiplicator
    CArg:
        # list of sub-connectiors that will get output records;
        # every sub-connection may support different interface.
        # default: []
        clist:
```

(continues on next page)

(continued from previous page)

```
            - Module: datconv.outconn.dcfile
              CArg:
                path: "out/AddnDrawNbrs_c5019_s38_1.json"
            - Module: datconv.outconn.crate.json
              CArg:
                path: "out/AddnDrawNbrs_c5019_s38_2.json"
                cast:
                    - [['rec', 'value'], str]
                lowercase: True
```

datconv.outconn.sqlite package

Module supporting storing data to SQLIte database files.

General interface description: *Output Connector interface*.

datconv.outconn.sqlite.ddl module

This module implements Datconv Output Connector which generates CREATE TABLE cause in SQLite dialect. Output of this connector should be treated as starting point for further manual edition, it is not recommended to use it for automatic table generation. This connector should be used with Writer: *datconv.writers.dcxpaths module* with options `add_header: false` and `add_type: true`.

class datconv.outconn.sqlite.ddl.**DCConnector**(*table*, *path*, *mode='w'*, *schema='main'*, *check_keywords=True*, *lowercase=0*, *column_constraints={}*, *common_column_constraints=[]*, *table_constraints=[]*)

Bases: `object`

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **table** – name of the table.
- **path** – relative or absolute path to output file.
- **mode** – output file opening mode.
- **schema** – schema of the table.
- **check_keywords** – if true, prevents conflicts with SQL keywords. Data field names that are in conflict will be suffixed with undderscore.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **column_constraints** – dictionary: key=column name, value=column constraint.
- **common_column_constraints** – column constatins to be added after column definitions. Should be a list.
- **table_constraints** – table constatins and creation options. Should be a list.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.sqlite.jinsert module

This module implements Datconv Output Connector which directly inserts data to SQLite database. This connector should be used with Writer: *datconv.writers.dcjson module*.

class datconv.outconn.sqlite.jinsert.**DCConnector**(*connstring*, *table*, *schema='main'*, *dump_sql=False*, *autocommit=False*, *bulk_size=10000*, *check_keywords=True*, *lowercase=0*, *cast=None*, *on_conflict=None*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **connstring** – connection string to database (path to file in this case).
- **table** – table name where to insert records.
- **schema** – table schema name where to insert records.
- **dump_sql** – if true, insert statements are being saved to file specified as connstring with added '.sql' extension and not applied to database (option to be used for debugging).
- **bulk_size** – if consequtive records have similar structure (i.e. have the same fields) - they are groupped into one pack (up to the size specified as this parameter) and inserted in one command. If set value is 0 than every insert is done individaually - warning: it is quite slow operation.
- **autocommit** – if true, every insert is automatically commited (slows down insert operations radically); if false, chenges are commited at the end - i.e. if any insert fail everything is rolled back and no records are added.
- **check_keywords** – if true prevents conflicts with SQL keywords.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **cast** – array of arrays of the form: [['rec', 'value'], str], what means that record: {"rec": {"value": 5025}} will be treated as {"rec": {"value": "5025"}} - i.e. it is ensured that "value" will allways be string. First position determines address of data to be converted, last position specifies the type: str, bool, int, long or float. Field names shold be given after all other configured transformations (lowercase, check_keywords).
- **on_conflict** – specify what to do when record with given primary key exist in the table; one of strings 'ignore', 'update' (or 'replace' with the same effect), 'rollback', 'abort', fail' or None (raise error in such situation).

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with output connectors contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
OutConnector:
    Module: datconv.outconn.sqlite.ddl
    CArg:
      # name of the table; obligatory
      table: product

      # relative or absolute path to output file; obligatory
      path: "out/AddnDrawNbrs_c5019_s38_2.sql"

      # output file opening mode (w or a); optional
      # default: w
      mode: a

      # schema of the table
      # default: main
      schema: main

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 0
      lowercase: 1

      # dictionary: key=column name, value=column constraint; optional.
      # default: {}
      column_constraints:
        cdc: NOT NULL

      # column constatins to be added after column definitions. Should be a list
      # default: []
      common_column_constraints:
        - PRIMARY KEY(cdc, isn)

      # table constatins and creation options. Should be a list
      # default: []
      table_constraints: []

OutConnector:
    Module: datconv.outconn.sqlite.jinsert
    CArg:
      # connection string to database (path to file in this case); obligatory
      connstring: "out/AddnDrawNbrs_c5019_s38_2.sqlite"

      # name of the table; obligatory
      table: product

      # schema of the table
      # default: main
      schema: main

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
```

(continues on next page)

(continued from previous page)

```
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 0
      lowercase: 1

      # if true, insert statements are being saved to file specified as connstring
      # with added '.sql' extension and not applied to database.
      # default: false
      dump_sql: true

      # if true, every insert is automatically commited (slows down insert operations␣
↪radically),
      # if false, chenges are commited at the end - i.e. if any insert fail
      # everything is rolled back and no records are added.
      # default: false
      autocommit: true

      # if consequtive records have similar structure (i.e. have the same fields),
      # they are groupped into one pack (up to the size specified as this parameter)
      # and inserted in one command.
      # If set value is 0 than every insert is done individaually - warning: it is␣
↪slow operation.
      # default: 10000
      bulk_size: 5000

      # array of arrays of the form: [['rec', 'value'], str], what means that record:
↪{"rec": {"value": 5025}}
      # will be writen as {"rec": {"value": "5025"}} - i.e. it is ensured that "value
↪" will allways be string.
      # First position determines address of data to be converted, last position␣
↪specifies the type: str, bool, int, long or float.
      # Field names shold be given after all other configured transformations␣
↪(lowercase,  check_keywords)
      # default: none
      cast:
          - [['rec', 'value'], str]

      # specify what to do when record with given primary key exist in the table;
      # one of strings 'ignore', 'update' (or 'replace' with the same effect),
↪'rollback', 'abort', fail' or None (raise error in such situation).
      # default: none
      on_conflict: update
```

datconv.outconn.postgresql package

Module supporting storing data to PostreSQL database.

General interface description: *Output Connector interface*.

datconv.outconn.postgresql.**obj2db**(*obj*)

Converts Python object to object acceptable as parameter replacement in execute() function

datconv.outconn.postgresql.**obj2str**(*obj*, *quota="'"*)
Converts Python object to string to be used in INSERT clause

datconv.outconn.postgresql.ddl module

This module implements Datconv Output Connector which generates CREATE TABLE cause in PostreSQL dialect. Output of this connector should be treated as starting point for further manual edition, it is not recommended to use it for automatic table generation.. This connector should be used with Writer: *datconv.writers.dcxpaths module* with options `add_header: false` and `add_type: true`.

class datconv.outconn.postgresql.ddl.**DCConnector**(*table*, *path*, *mode='w'*, *schema='public'*, *check_keywords=True*, *lowercase=0*, *column_constraints={}*, *common_column_constraints=[]*, *table_constraints=[]*)

Bases: `object`

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **table** – name of the table.
- **path** – relative or absolute path to output file.
- **mode** – output file opening mode.
- **schema** – schema of the table.
- **check_keywords** – if true, prevents conflicts with SQL keywords. Data field names that are in conflict will be suffixed with undderscore.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **column_constraints** – dictionary: key=column name, value=column constraint.
- **common_column_constraints** – column constatins to be added after column definitions. Should be a list.
- **table_constraints** – table constatins and creation options. Should be a list.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.postgresql.jinsert module

This module implements Datconv Output Connector which directly inserts data to PostreSQL database. This connector should be used with Writer: *datconv.writers.dcjson module*. It requires Python package `psycopg2` to be installed.

class datconv.outconn.postgresql.jinsert.**DCConnector**(*connstring*, *table*, *schema='public'*, *dump_sql=False*, *autocommit=False*, *check_keywords=True*, *lowercase=0*, *cast=None*, *on_conflict=None*, *options=[]*)

Bases: `object`

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **connstring** – connection string to database.
- **table** – table name name where to insert records.
- **schema** – table schema name where to insert records.
- **dump_sql** – if true, insert statements are being saved to file specified as `connstring` and not inserted to database (option to be used for debugging).
- **autocommit** – parameter passed to connection, if true every insert is automatically commited (slows down insert operations radically), if false chenges are commited at the end - i.e. if any insert fail everything is rolled back and no records are added.
- **check_keywords** – if true, prevents conflicts with SQL keywords.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **cast** – array of arrays of the form: [[‘rec’, ‘value’], str], what means that record: {“rec”: {“value”: 5025}} will be writen as {“rec”: {“value”: “5025”}} - i.e. it is ensured that “value” will allways be string. First position determines address of data to be converted, last position specifies the type: str, bool, short, int, long, float or array. Where short stands for 16 bit integer, int - 32 bit integer and long - 64 bit integer. Field names shold be given after all other configured transformations (lowercase, no_underscore, check_keywords).
- **on_conflict** – specify what to do when record with given primary key exist in the table; one of strings ‘ignore’, ‘update’ or None (raise error in such situation).
- **options** – array or additional options added to INSERT caluse (see Posgresql documentation).

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with output connectors contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
OutConnector:
    Module: datconv.outconn.postgresql.ddl
    CArg:
      # name of the table; obligatory
      table: product

      # relative or absolute path to output file; obligatory
      path: "out/AddnDrawNbrs_c5019_s38_2.sql"

      # output file opening mode (w or a); optional
      # default: w
      mode: a

      # schema of the table
```

(continues on next page)

(continued from previous page)

```
      # default: public
      schema: public

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 0
      lowercase: 1

      # array of arrays of the form: [['rec', 'value'], str], what means that record:
↪{"rec": {"value": 5025}}
      # will be writen as {"rec": {"value": "5025"}} - i.e. it is ensured that "value
↪" will allways be string.
      # First position determines address of data to be converted, last position␣
↪specifies the type: str, bool, short, int, long or float.
      # Where short stands for 16 bit integer, int - 32 bit integer and long - 64 bit␣
↪integer.
      # Field names shold be given after all other configured transformations␣
↪(lowercase,  check_keywords)
      # default: none
      cast:
        - [['rec', 'value'], str]

      # dictionary: key=column name, value=column constraint; optional.
      # default: {}
      column_constraints:
        cdc: NOT NULL

      # column constatins to be added after column definitions. Should be declared as␣
↪string
      # default: []
      common_column_constraints: |
        PRIMARY KEY("cdc", "isn")
        UNIQUE("id")

      # table constatins and creation options. Should be declared as string
      # default: []
      table_constraints: |
        PARTITION BY RANGE ("cdc")
        TABLESPACE diskvol1

OutConnector:
    Module: datconv.outconn.postgresql.jddl
    # This module has the same parameters than ddl (see above).

OutConnector:
    Module: datconv.outconn.postgresql.jinsert
    CArg:
      # connection string to database; obligatory
      connstring: host='192.168.1.15' dbname='postgres' user='postgres' password=
↪'postgres'
```

(continues on next page)

(continued from previous page)

```
      # name of the table; obligatory
      table: product

      # schema of the table
      # default: public
      schema: public

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 0
      lowercase: 1

      # if true, insert statements are being saved to file specified as connstring.
      # default: false
      dump_sql: true

      # parameter passed to connection, if true every insert is automatically␣
↪commited (slows down insert operations radically),
      # if false, chenges are commited at the end - i.e. if any insert fail
      # everything is rolled back and no records are added.
      # default: false
      autocommit: true

      # array of arrays of the form: [['rec', 'value'], str], what means that record:
↪{"rec": {"value": 5025}}
      # will be writen as {"rec": {"value": "5025"}} - i.e. it is ensured that "value
↪" will allways be string.
      # First position determines address of data to be converted, last position␣
↪specifies the type: str, bool, int, long or float.
      # Field names shold be given after all other configured transformations␣
↪(lowercase,  check_keywords)
      # default: none
      cast:
        - [['rec', 'value'], str]

      # specify what to do when record with given primary key exist in the table;
      # one of strings 'ignore', 'update' or None (raise error in such situation).
      # default: none
      on_conflict: update

      # array or additional options added to INSERT caluse (see Posgresql␣
↪documentation).
      # default: []
      options: []
```

datconv.outconn.crate package

Module supporting storing data to Crate database.

General interface description: *Output Connector interface*.

Note: In this module all options `lowercase` and `no_underscore` are enabled by default, because Crate (versions 2.2 and 2.3) sometimes silently converts field names to lower case and does not load data (using COPY command) if fields begin with underscore.

datconv.outconn.crate.ddl module

This module implements Datconv Output Connector which generates CREATE TABLE cause in database crate.io dialect. Output of this connector should be treated as starting point for further manual edition, it is not recommended to use it for automatic table generation. This connector should be used with Writer: *datconv.writers.dcxpaths module* with options `add_header: false` and `add_type: true`.

class datconv.outconn.crate.ddl.**DCConnector**(*table*, *path*, *mode='w'*, *schema='doc'*, *check_keywords=True*, *lowercase=1*, *no_underscore=1*, *column_constraints={}*, *common_column_constraints=[]*, *table_constraints=[]*)

Bases: `object`

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **table** – name of the table.
- **path** – relative or absolute path to output file.
- **mode** – output file opening mode.
- **schema** – schema of the table.
- **check_keywords** – if true, prevents conflicts with SQL keywords. Data field names that are in conflict will be suffixed with undderscore.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **no_underscore** – if >1, leading _ will be removed from all JSON keys; if =1, only from first level of keys; if =0, option is disabled.
- **column_constraints** – dictionary: key=column name, value=column constraint.
- **common_column_constraints** – column constatins to be added after column definitions. Should be declared as string.
- **table_constraints** – table constatins and creation options. Should be declared as string.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.crate.json module

This module implements Datconv Output Connector which saves data to file that contain one JSON object per line. Such files can serve as imput files for COPY FROM caluse used in crate.io database. This connector should be used with Writer: datconv.writers.dcjson

class datconv.outconn.crate.json.**DCConnector**(*path*, *check_keywords=True*, *lowercase=1*, *no_underscore=1*, *move_to_front=[]*, *cast=None*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **path** – relative or absolute path to output file.
- **check_keywords** – if true, prevents conflicts with SQL keywords.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **no_underscore** – if >1, leading _ will be removed from all JSON keys; if =1, only from first level of keys; if =0, option is disabled.
- **move_to_front** – those first level keys will be placed at begin of record. This option requires dcjson Writer option preserve_order to be set.
- **cast** – array of arrays of the form: [[‘rec’, ‘value’], str], what means that record: {“rec”: {“value”: 5025}} will be writen as {“rec”: {“value”: “5025”}} - i.e. it is ensured that “value” will allways be string. First position determines address of data to be converted, last position specifies the type: str, bool, int, long or float. Field names shold be given after all other configured transformations (lowercase, no_underscore, check_keywords).

For more detailed descriptions see *conf_template.yaml* file in this module folder.

datconv.outconn.crate.insert module

This module implements Datconv Output Connector which directly inserts data to crate.io database. This connector should be used with Writer: *datconv.writers.dcjson module* called with option preserve_order: true. It requires Python package crate to be installed. TODO: Add suport for ON CONFLICT

class datconv.outconn.crate.insert.**DCConnector**(*table*, *connstring*, *user=None*, *password=None*, *schema='doc'*, *dump_sql=False*, *bulk_size=10000*, *check_keywords=True*, *lowercase=1*, *no_underscore=1*, *cast=None*, *on_conflict=None*, *options=[]*)

Bases: object

Please see constructor description for more details.

Parameters are usually passed from YAML file as subkeys of OutConnector:CArg key.

Parameters

- **table** – table name where to insert records.
- **connstring** – connection string to database.
- **user** – user name for databse connection.
- **password** – password for databse connection.
- **schema** – table schema name where to insert records.

- **dump_sql** – if true, insert statements are being saved to file specified as `connstring` and not inserted to database (option to be used for debugging).
- **bulk_size** – if consequtive records have similar structure (i.e. have the same fields) - they are groupped into one pack (up to the size specified as this parameter) and inserted in one command. If set value is 0 than every insert is done individaually - warning: it is slow operation.
- **check_keywords** – if true prevents conflicts with SQL keywords.
- **lowercase** – if >1, all JSON keys will be converted to lower-case; if =1, only first level keys; if =0, no conversion happen.
- **no_underscore** – if >1, leading _ will be removed from all JSON keys; if =1, only from first level of keys; if =0, option is disabled.
- **cast** – array of arrays of the form: [[‘rec’, ‘value’], str], what means that record: {“rec”: {“value”: 5025}} will be writen as {“rec”: {“value”: “5025”}} - i.e. it is ensured that “value” will allways be string. First position determines address of data to be converted, last position specifies the type: str, bool, int, long or float. Field names shold be given after all other configured transformations (lowercase, no_underscore, check_keywords).
- **on_conflict** – specify what to do when record with given primary key exist in the table; one of strings ‘update’ or None (raise error in such situation).
- **options** – array or additional options added to INSERT caluse (see Crate documentation), it may be also ON DUPLICATE KEY phrase with non default settings.

For more detailed descriptions see *conf_template.yaml* file in this module folder.

Configuration keys

Listing of all possible configuration keys to be used with output connectors contained in this package.

There are sample values given, if key is not specified in configuration file, than default value is assumed.

```
OutConnector:
    Module: datconv.outconn.crate.ddl
    CArg:
      # name of the table; obligatory
      table: product

      # relative or absolute path to output file; obligatory
      path: "out/AddnDrawNbrs_c5019_s38_2.sql"

      # output file opening mode (w or a); optional
      # default: w
      mode: a

      # schema of the table
      # default: doc
      schema: doc

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
```

(continues on next page)

(continued from previous page)

```
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 1
      lowercase: 1

      # if >1, leading ``_`` will be removed from all JSON keys;
      # if =1, only from first level of keys;
      # if =0, option is disabled.
      # default: 1
      no_underscore: 1

      # dictionary: key=column name, value=column constraint; optional.
      # default: {}
      column_constraints:
        cdc: NOT NULL

      # column constatins to be added after column definitions.  Should be a list
      # default: []
      common_column_constraints:
        - PRIMARY KEY(cdc, isn)

      # table constatins and creation options.  Should be a list
      # default: []
      table_constraints:
        - PARTITION BY (cdc)

OutConnector:
    Module: datconv.outconn.crate.json
    CArg:
      # relative or absolute path to output file; obligatory
      path: "out/AddnDrawNbrs_c5019_s38_2.json"

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 1
      lowercase: 1

      # if >1, leading ``_`` will be removed from all JSON keys;
      # if =1, only from first level of keys;
      # if =0, optionj is disabled.
      # default: 1
      no_underscore: 1

      # list of first level keys that will be placed at begin of record.
      # This option requires ``dcjson`` Writer option ``preserve_order`` to be set.
      # default: []
      move_to_front: ['cdc', 'recno']

      # array of arrays of the form: [['rec', 'value'], str], what means that record:
↪{"rec": {"value": 5025}}
      # will be writen as {"rec": {"value": "5025"}} - i.e. it is ensured that "value
↪" will allways be string.
```

(continues on next page)

(continued from previous page)

```
      # First position determines address of data to be converted, last position␣
↪specifies the type: str, bool, int, long or float.
      # Field names shold be given after all other configured transformations␣
↪(lowercase, no_underscore, check_keywords)
      # default: none
      cast:
          - [['rec', 'value'], str]

OutConnector:
    Module: datconv.outconn.crate.insert
    CArg:
      # name of the table; obligatory
      table: product

      # connection string to database; obligatory
      connstring: http://192.168.1.15:4200

      # user name for databse connection
      # default: none
      user: crate

      # connection string to database; obligatory
      # default: none
      password: none

      # schema of the table
      # default: doc
      schema: doc

      # if true, prevents conflicts with SQL keywords;
      # data field names that are in conflict will be suffixed with undderscore.
      # default: true
      check_keywords: true

      # if >1, all JSON keys will be converted to lower-case;
      # if =1, only first level keys;
      # if =0, no conversion happen.
      # default: 1
      lowercase: 1

      # if >1, leading ``_`` will be removed from all JSON keys;
      # if =1, only from first level of keys;
      # if =0, optionj is disabled.
      # default: 1
      no_underscore: 1

      # if true, insert statements are being saved to file specified as connstring.
      # default: false
      dump_sql: true

      # if consequtive records have similar structure (i.e. have the same fields)
      # they are groupped into one pack (up to the size specified as this parameter)␣
↪and inserted in one command.
      # If set value is 0 than every insert is done individaually - warning: it is␣
↪slow operation.
      # default: 10000
      bulk_size: 5000
```

(continues on next page)

(continued from previous page)

```
      # array of arrays of the form: [['rec', 'value'], str], what means that record:
↪{"rec": {"value": 5025}}
      # will be writen as {"rec": {"value": "5025"}} - i.e. it is ensured that "value
↪" will allways be string.
      # First position determines address of data to be converted, last position␣
↪specifies the type: str, bool, int, long or float.
      # Field names shold be given after all other configured transformations␣
↪(lowercase, no_underscore, check_keywords)
      # default: none
      cast:
          - [['rec', 'value'], str]

      # specify what to do when record with given primary key exist in the table;
      # one of strings 'update' or None (raise error in such situation).
      # default: none
      on_conflict: update

      # array or additional options added to INSERT caluse (see Crate documentation),
      # it may be also ON DUPLICATE KEY phrase with non default settings.
      # default: []
      options: []
```

7.4 Upgrade instructions

7.4.1 From versions 0.6.x to Datconv 0.7.x

- To support iterators Reader and Writer interfaces were extended. Following changes should be done in Reader classes: * If possible, write `Iterate` method which will work in similar way than `Process` but should call yield on object returned by writer's `writeRecord` method. Typically `Iter` method is a clone of `Process` with `writeRecord(...)` call replaced with `yield writeRecord(...)`. Following changes should be done in Writer classes: * Remember header and footer passed in funtions: `writeHeader` / `writeFooter`, and return them in new functions: `getHeader` / `getFooter`. * `writeRecord` class method should now return an object which it pass to OBJECT type output connector (prevoiusly this method did not returned any value.

7.4.2 From versions 0.6.0 to Datconv 0.6.1

- Output connectors `datconv.outconn.*.ddl` were changed in non-compatible way: parameters `primary_key` and `not_null` were replaced by `column_constraints`, `common_column_constraints` and `table_constraints`.

7.4.3 From versions 0.5.x to Datconv 0.6.x

- The type of `datconv.writers.dcjson` parameter `convert_values` was changed from boolean to int. If you had false change to 0, true - change to 1 or 2. Also default value changed to make conversion by default.
- Name of of `datconv.writers.dcjson` parameter `ign_rectyp` was changed to `ignore_rectyp` to unify names with xpath writer.

- Old fashion parameter `Reader:PArg:outpath` is still supported as fall-back, however it is recommended to explicitly specify output connectior for better clarity. I.e. replace:

```
Reader:
    Module: datconv.readers.dcxml
    # ...
    PArg:
        outpath: "out/AcctAgentAdjustment_c5019_s38.xml"
```

  with:

```
OutConnector:
    Module: datconv.outconn.dcfile
    CArg:
        path: "out/AcctAgentAdjustment_c5019_s38.xml"
```

- Default debugging level is now INFO (was WARNING) so some information messages are printer to console when program is run without any special logger configuration. To restore previous (silent) behaviour add `DefLogLevel: WARNING` root key in your configuration file.

7.4.4 From earlier versions to Datconv 0.5.1

New Filter optional functions `setHeader()` and `setFooter()`:

- If you managed to write custom Reader, it must be updated in following way: call filter optional functions before calling Writer `writeHeader()` and `writeFooter()` functions as shown below:

```
if self._flt is not None:
    if hasattr(self._flt, 'setHeader'):
        self._flt.setHeader(self._header)
self._wri.writeHeader(self._header)
```

  and in similar way for Footer.

- If you have a filter that does some work at the end of converion process (like writing some statistics) and you implemented it in `__del__()` fumction, this finalization code can be now moved to `setFooter()`.

7.4.5 From Datconv version 0.2.x to Datconv 0.3.x

Changes in parameters default values

Changed default value for parameter `encoding` of `datconv.writers.dcxml`. It was `ascii` and is now `unicode`. Note that this parameter is ignored while run against Python3.

7.4.6 From Pandata version 0.1 to Datconv 0.2.x

Changes in files' layout

Pandata version 0.1 did not had its own installer. Everything was placed in one folder. Now Datconv (renamed from Pandata) modules included in package are being installed in special Python subfolder intended for 3-rd party modules. So all old Pandata core modules may be deleted from current localisation. Recomended Datconv layout is now to keep user own modules in subfolders of one folder, and run datconv script from that folder or add user's "projects' folder"

to PYTHONPATH environment variable. Datconv core modules should be kept in Python sub-folders where installer placed them.

Modules and classes' name changes

Names of standard Reader and Writer modules has changed: pdxml -> dcxml, pdcsv -> dccsv, pdxpaths -> dcxpaths. Appropriate changes should be done in user's YAML configuration files.

Fixed names of Reader, Filter and Writer classes has changed: PDReader -> DCReader, PDFilter -> DCFilter, PDWriter -> DCWriter. Appropriate changes should be done in user's filters' definition files.

All version 0.1 modules were moved to common datconv meta package as subpackages. So in user's YAML configuration files and in import statements in filters datconv. specification should be added.

Example 1 (YAML file):

```
Module:  writers.pdcsv
```

should be changed to:

```
Module:  datconv.writers.dccsv
```

Example 2 (filter definition):

```
from filters import WRITE
```

should be changed to:

```
from datconv.filters import WRITE
```

Change in main program invocation

Datconv main script is now included into system PATH. On Linux .py file extension was removed, so it can be called like (without ./ prefix): datconv [options]. On Windows script was renamed, so it can be called like (without python explicit invocation): datconv-run.py [options].

Note: Windows Python installer should create file-type entries that allow user to directly call python script (without python explicit invocation).

To check that, run following connads from command box: C:\>assoc .py should give: .py=Python.File

and: C:\>ftype Python.File should give: Python.File="c:\python27\python.exe" "%1" %*

Important is %* at end — what allows to pass additional arguments to program.

7.5 Changelog

7.5.1 Development plans for future

Points are specified in priority (and probably implementation) order:

- Add more options to Excel Output connector
- Add cinsert (working with csv writer) Output connectiors to sqlite and postgresql modules.

- Introduce option to run connectors as separate process with queue between writer and connector for better performance (espacially with database connectors).
- Output connectors: Couchbase, MongoDB, MySQL, zip-file, snappy-file, Avro, Kafka.
- Introduce Filter dispatchers to split data flow to few streams (e.g. to optimize database inserts), option to run them in separate processes.
- Add input comnnectors layer. Imput connectors for databases (query as source of data).
- Readers/Writers: Python pickle, Pandas.
- Output connector: Postgresql binary file (for COPY clause).
- Better support for running Datconv as paralell proceses e.g. convering big files in paralell processes (using rfrom/rto settings). Support for skipping headers footers etc.
- Create Windows binary form of program (with cx_Freeze package) that does not require Python installation and upload to github.
- I'm considerating rewriting program in Julia language.

7.5.2 Notes about versioning schema

- Major number will be changed when changes breaks backward compatibility, i.e. users may have to slightly change their own modules or configuration in order to work with new release. However if this number is zero, API is considerated unstable and may change with any feature release. This is called Major Release, and in this case midle and minor numbers are reset to zero.
- Midle number will be changed when new features or options will be introduced but without API break. This is called Feature Release, and in this case minor number is reset to zero.
- Minor number will be changed when fixes or very small, non-risky features are introduced. This is called Fix Release.

7.5.3 0.8.1-2 (2019.12.28)

Improvements

- Verified package against Python 3.7
- Fixes in package long description (separate file for sphinx)

7.5.4 0.8.0 (2018.08.21)

Improvements

- Added `columns: 1` parameter to `datconv.writers.dccsv`. It allows to place headers in first line of output when all records have the same structure.
- Added new output connector: `datconv.outconn.dcexcel` to be used with CSV writer for writing MS Excel (.xlsx) files.

7.5.5 0.7.3 (2018.08.07)

Improvements

- Added support for schema names to database output connectors.

7.5.6 0.7.0-2 (2018.04.19)

Improvements

- Added possibility to run Datconv as iterator (i.e. obtain output records directly in Python code).

Fixes

- Default log level is set to INFO when run from command line and ERROR when run from Python.
- Added additional range checks when cast option is used in `datconv.outconn.postgresql.jinsert` to prevent INSERT errors.
- Fixed bug in `datconv.writers.dcjson` which produced invalid output `key: NaN` while converting "Nan" string.

7.5.7 0.6.1 (2018.03.17)

Improvements

- Improved command line option `--default`.
- `datconv.outconn.postgresql.jddl`: new connector
- `datconv.outconn.postgresql.jinsert`: added support for JSON types.
- `datconv.outconn.postgresql.jinsert`: added support for casting to ARRAY.
- More flexible configuration of ddl connectors (not backword compatible changes - see *Upgrade instructions*)

Fixes

- `datconv.outconn.postgresql.jinsert`: when `autocommit: false` no records are saved in case of error.

7.5.8 0.6.0 (2018.02.17)

Improvements

- Added output connectors layer - see *Output Connector interface*.
- Added parameters `rectyp_separator` and `add_type` to XPath Writer to better support database ddl connectors.
- Introduced *Default Configuration*.

Fixes

- Fix program crash with json readers when status reporting was enabled.

7.5.9 0.5.1 (2018.01.20)

Improvements

- Added optional filter method `setFooter()` to inform filter about contents of data footer and give it a chance to change it.
- Convert standard filters pipe, stat, statex to use `setFooter()` instead of `__del__()`.

Fixes

- Fix program crash when dcjson writer was used with `with_prop: true` option.

7.5.10 0.5.0 (2018.01.06)

Improvements

- Added new standard filters: pipe, gen_rec.
- Added optional filter method `setHeader()` to inform filter about contents of data header and give it a chance to change it.

7.5.11 0.4.1 (2017.08.16)

Fixes

- Small fixes in documentation.

7.5.12 0.4.0 (2017.08.15)

Improvements

- XML Reader: added parameter `foottags`.
- XML Reader: parameter `rectags` can be empty (see documentation).
- XML Writer: added parameters `add_header`, `add_footer`.
- Added JSON Writer.
- Added JSON Readers.
- Added CSV Reader.
- Added command line option: `--help`.

7.5.13 0.3.4 (2017.05.12)

Fixes

- Small fixes after documentation was published on-line.

7.5.14 0.3.3 (2017.05.06)

Improvements

- Adopted pydoc descriptions in sources to Sphinx.
- Created first version of documentation using Sphinx Project.

7.5.15 0.3.2 (2016.06.01)

Improvements

- Extended method `Datconv().Version()` for possibility to display version of external module.

7.5.16 0.3.1 (2016.05.27)

Fixes

- Fixed exceptions being logged only to console (stderr, not by configured logger).
- Fixed duplicated log entries to console (bug introduded by 0.3.0 version).

Improvements

- Added method `Datconv().Version()`.

7.5.17 0.3.0 (2016.05.24)

Fixes

- Fixed value returned to shell by `datconv` script.

Improvements

- Port to Python 3.
- Add option to inherit logger (to use when datconv is called from Python script that already has its own logger).
- Created basic test scripts - available as separate `datconv_tests` package.
- New filter: `datconv.filters.statex`.

7.5.18 0.2.4 (2016.03.06)

Fixes

- Fixed bug that caused writers/dcxml.py to write multiply XML closing tags in case when the same writer class instance was used to process multiply files.

7.5.19 0.2.3 (2016.01.20)

Fixes

- Fixed exception when user press `Ctrl-C` before script finish.

Improvements

- Added command line option: `--version`.

7.5.20 0.2.2 (2016.01.15)

Fixes

- Fixed `conf_template.yaml` files.

7.5.21 0.2.1 (2016.01.06)

Fixes

- Installation script no longer require `PyYAML` to be installed.
- Corrected import statements in `_skeleton.py` files.

7.5.22 0.2.0 (2015.12.29)

Fixes

- Ensure that XML Output is correct (i.e. have one root element).

Improvements

- Project/program/package rename due to conflicts with existing projects: Pandata -> Datconv.
- As consequence of above, renamed some modules and classes. See included Upgrade.md file for more information - changes in user files are needed.
- Added Datconv class - i.e. data conversion can be run as stand alone script: `datconv [options]` or from python code:

```
import datconv
dc = datconv.Datconv()
conf = {...}
dc.Run(conf)
```

This also implies that all subpackages were moved to one, root `datconv` package.

- Separated common and IGT specific modules into two separate packages. Datconv is now distributed as 2 packages created according python standard (`datconv` and `datconv-igt`).
- Added standard `setup.py` installation script. This means that package files are being installed in Python 3rd party package standard location.
- Licensed `datconv` under Python Software Foundation like license.

7.5.23 0.1 (2015.10 - 2015.12.04)

- Initial not-public release. Delivered only to IGT coworkers.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

E

F

G

I

L

O

P

R

S

T

V

W