pycounter Documentation

Release 3.0.0-a1

Geoffrey Spear

Apr 07, 2021

Contents

Release v3.0.0-a1

pycounter makes working with COUNTER usage statistics in Python easy, including fetching statistics with NISO SUSHI.

A simple command-line client for fetching JR1 reports from SUSHI servers and outputting them as tab-separated COUNTER 4 reports is included.

Developed by the Health Sciences Library System of the University of Pittsburgh to support importing usage data into our in-house Electronic Resources Management (ERM) system.

Licensed under the MIT license. See the file LICENSE for details.

pycounter is tested on Python 3.6, 3.7, 3.8, 3.9 and pypy3

Documentation is on Read the Docs and the code can be found on GitHub.

CHAPTER 1

Installing

From pypi:

> pip install pycounter

From inside the source distribution:

> pip install [-e] .

(use -e if you plan to work on the source itself, so your changes are used in your installation. Probably do all of this in a virtualenv. The PyPA has a good explanation of how to get started.)

COUNTER 5 Note

In this release, reports are output in COUNTER 4 format with COUNTER 5 data, which is wrong, and probably not a valid apples-to-apples comparison since, for example, TR_J1 excludes Gold Open Access counts that would be included in JR1, and also has HTML and PDF columns that will always be 0 because these are no longer reported.

Before the 3.0 release, it should be capable of producing actual COUNTER 5 reports, probably with an API for getting COUNTER 4 style data compatible with scripts that were making assumptions about the data received to pass it into another system.

Usage

Parsing COUNTER reports (currently supports 4 in .csv, .tsv, or .xlsx files, reports JR1, JR2, DB1, DB2, BR1, BR2, and BR3) and COUNTER 5:

```
>>> import pycounter.report
>>> report = pycounter.report.parse("COUNTER4_2015.tsv")  # filename or path to file
>>> print(report.metric)
FT Article Requests
>>> for journal in report:
...     print(journal.title)
Sqornshellous Swamptalk
Acta Mattressica
>>> for stat in report.pubs[0]:
...     print(stat)
(datetime.date(2015, 1, 1), 'FT Article Requests', 120)
(datetime.date(2015, 2, 1), 'FT Article Requests', 42)
(datetime.date(2015, 3, 1), 'FT Article Requests', 23)
```

Fetching SUSHI data:

```
>>> import pycounter.sushi
>>> import datetime
>>> report = pycounter.sushi.get_report(wsdl_url='http://www.example.com/SushiService
↪',
...     start_date=datetime.date(2015,1,1), end_date=datetime.date(2015,1,31),
...     requestor_id="myreqid", customer_reference="refnum", report="JR1",
...     release=4)
>>> for journal in report:
...     print(journal.title)
Sqornshellous Swamptalk
Acta Mattressica
```

Output of report as TSV:

```
>>> report.write_tsv("/tmp/counterreport.tsv")
```

Development

Our code is automatically styled using black. To install the pre-commit hook:

pip install pre-commit

pre-commit install

CHAPTER 5

API Docs

pycounter.report	COUNTER journal and book reports and associated functions.
pycounter.sushi	NISO SUSHI support.
pycounter.exceptions	Exception classes for pycounter.

CHAPTER 6

Internal APIs

Indices and tables

- genindex
- modindex
- search

Contents

8.1 pycounter API Docs

8.1.1 pycounter.report module

Commonly-used function

`pycounter.report.`**`parse`** (*filename*, *filetype=None*, *encoding='utf-8'*, *fallback_encoding='latin-1'*)
Parse a COUNTER file, first attempting to determine type.

Returns a *CounterReport* object.

> **Parameters**
>
> - **`filename`** – path to COUNTER report to load and parse.
> - **`filetype`** – type of file provided, one of "csv", "tsv", "xlsx". If set to None (the default), an attempt will be made to detect the correct type, first from the file extension, then from the file's contents.
> - **`encoding`** – encoding to use to decode the file. Defaults to 'utf-8', ignored for XLSX files (which specify their encoding in their XML)
> - **`fallback_encoding`** – alternative encoding to use to try to decode the file if the primary encoding fails. This defaults to 'latin-1', which will accept any bytes (possibly producing junk results. . .) Ignored for XLSX files.

Classes

`class` `pycounter.report.`**`CounterReport`** (*report_type=None*, *report_version=4*, *metric=None*, *customer=None*, *institutional_identifier=None*, *period=(None, None)*, *date_run=None*, *section_type=None*)
a COUNTER usage statistics report.

Iterate over the report object to get its rows (each of which is a *CounterBook* or *CounterJournal* instance.

> **Parameters**
>
> - **metric** – metric being tracked by this report. For database reports (which have multiple metrics per report), this should be set to *None*.
>
> - **report_type** – type of report (e.g., "JR1", "BR2")
>
> - **report_version** – COUNTER version
>
> - **customer** – name of customer on report
>
> - **institutional_identifier** – unique ID assigned by vendor for customer
>
> - **period** – tuple of datetime.date objects corresponding to the beginning and end of the covered range
>
> - **date_run** – date the COUNTER report was generated
>
> - **section_type** – predominant section type used for this report. (applies to report BR2; should probably be None for any other report type)

as_generic()
> Output report as list of lists.
>
> Nested list will contain cells that would appear in COUNTER report (suitable for writing as CSV, TSV, etc.)

write_to_file(*path*, *format_*)
> Output report to a file.
>
> > **Parameters**
> >
> > - **path** – location to write file
> >
> > - **format** – file format. Currently supports 'tsv'
> >
> > **Returns**

write_tsv(*path*)
> Output report to a COUNTER 4 TSV file.
>
> > **Parameters path** – location to write file

class pycounter.report.**CounterEresource**(*period=None*, *metric=None*, *month_data=None*, *title=''*, *platform=''*, *publisher=''*)
> Base class for COUNTER statistics lines.
>
> Iterating returns (first_day_of_month, metric, usage) tuples.
>
> > **Parameters**
> >
> > - **period** – two-tuple of datetime.date objects corresponding to the beginning and end dates of the covered range
> >
> > - **metric** – metric tracked by this report. Should be a value from pycounter.report.METRICS dict.
> >
> > - **month_data** – a list containing usage data for this resource, as (datetime.date, usage) tuples
> >
> > - **title** – title of the resource
> >
> > - **publisher** – name of the resource's publisher
> >
> > - **platform** – name of the platform providing the resource

class pycounter.report.**CounterJournal**(*period=None*, *metric='FT Article Requests'*, *issn=None*, *eissn=None*, *month_data=None*, *title=''*, *platform=''*, *publisher=''*, *html_total=0*, *pdf_total=0*, *doi=''*, *proprietary_id=''*)

Statistics for a single electronic journal.

> **Parameters**
>
> - **period** – two-tuple of datetime.date objects corresponding to the beginning and end dates of the covered range
>
> - **metric** – the metric tracked by this statistics line. (Should probably always be "FT Article Requests" for CounterJournal objects, as long as only JR1 is supported.)
>
> - **issn** – eJournal's print ISSN
>
> - **eissn** – eJournal's eISSN
>
> - **month_data** – a list containing usage data for this journal, as (datetime.date, usage) tuples
>
> - **title** – title of the resource
>
> - **publisher** – name of the resource's publisher
>
> - **platform** – name of the platform providing the resource
>
> - **html_total** – total HTML usage for this title for reporting period
>
> - **pdf_total** – total PDF usage for this title for reporting period

as_generic()

Get data for this line as list of COUNTER report cells.

class pycounter.report.**CounterBook**(*period=None*, *metric=None*, *month_data=None*, *title=''*, *platform=''*, *publisher=''*, *isbn=None*, *issn=None*, *doi=''*, *proprietary_id=''*, *print_isbn=None*, *online_isbn=None*)

statistics for a single electronic book.

> **Variables**
>
> - *isbn* – eBook's ISBN
>
> - **issn** – eBook's ISSN (if any)
>
> **Parameters**
>
> - **month_data** – a list containing usage data for this book, as (datetime.date, usage) tuples
>
> - **title** – title of the resource
>
> - **publisher** – name of the resource's publisher
>
> - **platform** – name of the platform providing the resource

as_generic()

Get data for this line as list of COUNTER report cells.

isbn

Return a suitable ISSN for the ebook.

The tabular COUNTER reports only report an "ISBN", while the SUSHI (XML) reports include both a Print_ISBN and Online_ISBN.

> This property will return a generic ISBN given in the constructor, if any. If the CounterBook was created with no "isbn" but with online_ISBN and/or print_ISBN, the online one, if any, will be returned, otherwise the print.

class `pycounter.report.`**`CounterDatabase`**(*period=None*, *metric=None*, *month_data=None*, *title="*, *platform="*, *publisher="*)

> a COUNTER database report line.

> **`as_generic`**()
>> Return data for this line as list of COUNTER report cells.

Other functions

These are mostly for internal use by the module, but are available to be called directly if necessary

`pycounter.report.`**`format_stat`**(*stat*)

> Turn numbers possibly with embedded commas into integers.

> Also accepts existing ints, which may be pre-converted from Excel.

>> **Parameters `stat`** – numeric value, possibly with commas

>> **Returns** int

`pycounter.report.`**`parse_generic`**(*report_reader*)

> Parse COUNTER report rows into a CounterReport.

>> **Parameters `report_reader`** – a iterable object that yields lists COUNTER data formatted as tabular lists

>> **Returns** CounterReport object

`pycounter.report.`**`parse_separated`**(*filename*, *delimiter*, *encoding='utf-8'*, *fallback_encoding='latin-1'*)

> Open COUNTER CSV/TSV report and parse into a CounterReport.

> Invoked automatically by *`parse()`*.

>> **Parameters**

>>> • **`filename`** – path to delimited COUNTER report file.

>>> • **`delimiter`** – character (such as ',' or '\t') used as the delimiter for this file

>>> • **`encoding`** – file's encoding. Default: utf-8

>>> • **`fallback_encoding`** – alternative encoding to try to decode if default fails. Throws a warning if used.

>> **Returns** CounterReport object

`pycounter.report.`**`parse_xlsx`**(*filename*)

> Parse a COUNTER file in Excel format.

> Invoked automatically by `parse`.

>> **Parameters `filename`** – path to XLSX-format COUNTER report file.

8.1.2 pycounter.sushi module

Note: Before pycounter 1.1, SUSHI requests were always made with SSL verification turned off. The default is now to verify certificates. If you must contact a SUSHI server without verification, please use the verify=False argument to request() or the –no-ssl-verify flag on sushiclient.

Commonly-used function

pycounter.sushi.**get_report**(**args*, ***kwargs*)
> Get a usage report from a SUSHI server.

> returns a *pycounter.report.CounterReport* object.

> parameters: see get_sushi_stats_raw

>> **Parameters no_delay** – don't delay in retrying Report Queued

Other functions

pycounter.sushi.**get_sushi_stats_raw**(*wsdl_url*, *start_date*, *end_date*, *requestor_id=None*, *requestor_email=None*, *requestor_name=None*, *customer_reference=None*, *customer_name=None*, *report='JR1'*, *release=4*, *sushi_dump=False*, *verify=True*, ***extra_params*)
> Get SUSHI stats for a given site in raw XML format.

> **Parameters**

>> - **wsdl_url** – URL to SOAP WSDL for this provider
>> - **start_date** – start date for report (must be first day of a month)
>> - **end_date** – end date for report (must be last day of a month)
>> - **requestor_id** – requestor ID as defined by SUSHI protocol
>> - **requestor_email** – requestor email address, if required by provider
>> - **requestor_name** – Internationally recognized organization name
>> - **customer_reference** – customer reference number as defined by SUSHI protocol
>> - **customer_name** – Internationally recognized organization name
>> - **report** – report type, values defined by SUSHI protocol
>> - **release** – report release number (should generally be *4*.)
>> - **sushi_dump** – produces dump of XML (or JSON, for COUNTER 5) to DEBUG logger
>> - **verify** – bool: whether to verify SSL certificates
>> - **extra_params** – extra params are passed to requests.post

8.1.3 pycounter.exceptions module

8.2 pycounter Internal APIs

8.2.1 pycounter.sushi5 module

COUNTER 5 SUSHI support.

pycounter.sushi5.**get_status**(*url: str*) → str
> Request SUSHI server status.

`pycounter.sushi5.`**`get_sushi_stats_raw`**(*wsdl_url=None*, *start_date=None*, *end_date=None*, *requestor_id=None*, *customer_reference=None*, *report='TR_J1'*, *release=5*, *sushi_dump=False*, *verify=True*, *url=None*, *api_key=None*, ***kwargs*)

    Get SUSHI stats for a given site in dict (decoded from JSON) format.

        Parameters

- **`wsdl_url`** – (Deprecated; for backward compatibility with COUNTER 4 SUSHI code. Use *url* instead.) URL to API endpoint for this provider

- **`start_date`** – start date for report (must be first day of a month)

- **`end_date`** – end date for report (must be last day of a month)

- **`requestor_id`** – requestor ID as defined by SUSHI protocol

- **`customer_reference`** – customer reference number as defined by SUSHI protocol

- **`report`** – report type, values defined by SUSHI protocol

- **`release`** – COUNTER release (only 5 is supported in this module)

- **`sushi_dump`** – produces dump of JSON to DEBUG logger

- **`verify`** – bool: whether to verify SSL certificates

- **`url`** – str: URL to endpoint for this provider

- **`api_key`** – str: API key for SUSHI provider (not used by all vendors; see vendor instructions to determine if this is needed)

`pycounter.sushi5.`**`raw_to_full`**(*raw_report*)

    Convert a raw report to CounterReport.

        Parameters **`raw_report`** – raw report as dict decoded from JSON

        Returns a *`pycounter.report.CounterReport`*

8.2.2 pycounter.constants module

Constants used by pycounter.

8.2.3 pycounter.csvhelper module

Read CSV as unicode from both python 2 and 3 transparently.

class `pycounter.csvhelper.`**`UnicodeReader`**(*filename*, *dialect=<class 'csv.excel'>*, *encoding='utf-8'*, *fallback_encoding='latin-1'*, ***kwargs*)

    CSV reader that can handle unicode.

    Must be used as a context manager:

with UnicodeReader('myfile.csv') as reader: pass # do things with reader

        Parameters

- **`filename`** – path to file to open

- **`dialect`** – a csv.Dialect instance or dialect name

- **`encoding`** – text encoding of file

- **fallback_encoding** – encoding to fall back to if default encoding fails; gives warning if it's used.

All other parameters will be passed through to csv.reader()

class pycounter.csvhelper.**UnicodeWriter**(*filename*, *dialect=<class 'csv.excel'>*, *encoding='utf-8'*, *lineterminator='n'*, ***kwargs*)

CSV writer that can handle unicode.

Must be used as a context manager:

with UnicodeWriter('myfile.csv') as writer: pass # do things with writer

> **Parameters**
>
> - **filename** – path to file to open
> - **dialect** – a csv.Dialect instance or dialect name
> - **encoding** – text encoding of file

All other parameters will be passed through to csv.writer()

writerow(*row*)

Write a row to the output.

> **Parameters** **row** – list of cells to write to the file

writerows(*rows*)

Write many rows to the output.

> **Parameters** **rows** – list of lists of cells to write

8.2.4 pycounter.helpers module

Helper functions used by pycounter.

pycounter.helpers.**convert_covered**(*datestring*)

Convert coverage period string to datetimes.

> **Parameters** **datestring** – the string to convert to a date. Format as 'YYYY-MM-DD to YYYY-MM-DD'
>
> **Returns** tuple of datetime.date instances

(Will also accept MM/DD/YYYY format, ISO 8601 timestamps, or existing datetime objects; these shouldn't be in COUNTER reports, but they do show up in real world data. . .)

Also accepts strings of the form 'Begin_Date=2019-01-01; End_Date=2019-12-31' for better compatibility with some (broken) COUNTER 5 implementations.

pycounter.helpers.**convert_date_column**(*datestring*)

Convert human-readable month to date of first day of month.

> **Parameters** **datestring** – the string to convert to a date. Format like "Jan-2014".
>
> **Returns** datetime.date

pycounter.helpers.**convert_date_run**(*datestring*)

Convert a date of the format 'YYYY-MM-DD' to a datetime.date object.

(Will also accept MM/DD/YYYY format, ISO 8601 timestamps, or existing datetime objects; these shouldn't be in COUNTER reports, but they do show up in real world data. . .)

> **Parameters** `datestring` – the string to convert to a date.
>
> **Returns** datetime.date object

`pycounter.helpers.`**`format_stat`**(*stat*)

> Turn numbers possibly with embedded commas into integers.
>
> Also accepts existing ints, which may be pre-converted from Excel.
>
> > **Parameters** `stat` – numeric value, possibly with commas
> >
> > **Returns** int

`pycounter.helpers.`**`guess_type_from_content`**(*file_obj*)

> Guess type of a spreadsheet-like file.
>
> Defaults to assuming it's CSV, if it doesn't appear to be XLSX or TSV.
>
> > **Parameters** `file_obj` – file-like object of which to determine type.
> >
> > **Returns** string, one of "xlsx", "tsv", "csv"

`pycounter.helpers.`**`is_first_last`**(*period*)

> **Args:** period: a tuple of datetime.date objects
>
> **Returns: bool, whether the period starts on the 1st of a month and ends on** the last of a month

`pycounter.helpers.`**`last_day`**(*orig_date*)

> Find last day of a month from any day in the month.
>
> > **Parameters** `orig_date` – the date within the month for which we want the last day as date-time.date
> >
> > **Returns** datetime.date of last day of the month

`pycounter.helpers.`**`next_month`**(*dateobj*)

> Find the first day of the next month after the given date.
>
> > **Parameters** `dateobj` – the date within the month for which we want the next month's first day as datetime.date
> >
> > **Returns** datetime.date of the first day of the next month

`pycounter.helpers.`**`prev_month`**(*dateobj*)

> Find the first day of the previous month before the given date.
>
> > **Parameters** `dateobj` – the date within the month for which we want the previous month's first day as datetime.date.
> >
> > **Returns** datetime.date of first day of the previous month.

8.3 The sushiclient

pycounter comes with a rudimentary SUSHI command line client.

Note: Before pycounter 1.1, SUSHI requests were always made with SSL verification turned off. The default is now to verify certificates. If you must contact a SUSHI server without verification, please use the verify=False argument to request() or the –no-ssl-verify flag on sushiclient.

8.3.1 Invocation

sushiclient [OPTIONS] <URL>

URL
    The SUSHI endpoint/WSDL URL to use

Options:

-r, --report
    report name (default JR1)

-l, --release
    COUNTER release (default 4)

-s, --start_date
    Start Date (default first day of last month) in 'YYYY-MM-DD' format

-e, --end_date
    Ending Date (default last day of last month) in 'YYYY-MM-DD' format

-i, --requestor_id
    Requestor ID as defined in the SUSHI standard

--requestor_email
    Email address of requestor

--requestor_name
    Internationally recognized organization name

-c, --customer_reference
    Customer reference number as defined in the SUSHI standard

--customer_name
    Internationally recognized organization name

-f <format>, **--format** <format>
    Output format (currently only allows the default, tsv)

-o <output_file>, **--output_file** <output_file>
    Path to write output file to. If file already exists, it will be overwritten.

-d, --dump
    Dump raw request and response to logger.

--no_ssl_verify
    Skip SSL certificate verification.

--no-delay
    Do not wait 60 seconds before retrying a request in case of failure. This is provided mainly for testing; it's not recommended to skip the delay when talking to someone else's server...

--status
    Request the status of the (COUNTER 5 only at the moment) SUSHI server, print it, and exit. Ignores all other options except –release, –dump, and the URL.

Python Module Index

p

Symbols

A

C

F

G

I

L