Warcat Documentation

Release 2.2.4

Christopher Foo

Apr 11, 2017

Contents

Contents:

CHAPTER 1

Readme

WARCAT: Web ARChive (WARC) Archiving Tool

Tool and library for handling Web ARChive (WARC) files.

Quick Start

Requirements:

- Python 3

Install stable version:

```
pip-3 install warcat
```

Or install latest version:

```
git clone git://github.com/chfoo/warcat.git
pip-3 install -r requirements.txt
python3 setup.py install
```

Example Run:

```
python3 -m warcat --help
python3 -m warcat list example/at.warc.gz
python3 -m warcat verify megawarc.warc.gz --progress
python3 -m warcat extract megawarc.warc.gz --output-dir /tmp/megawarc/ --progress
```

Supported commands

concat Naively join archives into one

extract Extract files from archive

help List commands available

list List contents of archive

pass Load archive and write it back out

split Split archives into individual records

verify Verify digest and validate conformance

Library

Example:

```
>>> import warcat.model
>>> warc = warcat.model.WARC()
>>> warc.load('example/at.warc.gz')
>>> len(warc.records)
8
>>> record = warc.records[0]
>>> record.warc_type
'warcinfo'
>>> record.content_length
233
>>> record.header.version
'1.0'
>>> record.header.fields.list()
[('WARC-Type', 'warcinfo'), ('Content-Type', 'application/warc-fields'), ('WARC-Date',
↪ '2013-04-09T00:11:14Z'), ('WARC-Record-ID', '<urn:uuid:972777d2-4177-4c63-9fde-
↪3877dacc174e>'), ('WARC-Filename', 'at.warc.gz'), ('WARC-Block-Digest',
↪'sha1:3C6SPSGP5QN2HNHKPTLYDHDPFYKYAOIX'), ('Content-Length', '233')]
>>> record.header.fields['content-type']
'application/warc-fields'
>>> record.content_block.fields.list()
[('software', 'Wget/1.13.4-2608 (linux-gnu)'), ('format', 'WARC File Format 1.0'), (
↪'conformsTo', 'http://bibnum.bnf.fr/WARC/WARC_ISO_28500_version1_latestdraft.pdf'),␣
↪('robots', 'classic'), ('wget-arguments', '"http://www.archiveteam.org/" "--warc-
↪file=at" ')]
>>> record.content_block.fields['software']
'Wget/1.13.4-2608 (linux-gnu)'
>>> record.content_block.payload.length
0
>>> bytes(warc)[:60]
b'WARC/1.0\r\nWARC-Type: warcinfo\r\nContent-Type: application/war'
>>> bytes(record.content_block.fields)[:60]
b'software: Wget/1.13.4-2608 (linux-gnu)\r\nformat: WARC File Fo'
```

Note: The library may not be entirely thread-safe yet.

About

The goal of the Warcat project is to create a tool and library as easy and fast as manipulating any other archive such as tar and zip archives.

Warcat is designed to handle large, gzip-ed files by partially extracting them as needed.

Warcat is provided without warranty and cannot guarantee the safety of your files. Remember to make backups and test them!

- Homepage: https://github.com/chfoo/warcat
- Documentation: http://warcat.readthedocs.org/
- Questions?: https://answers.launchpad.net/warcat
- Bugs?: https://github.com/chfoo/warcat/issues
- PyPI: https://pypi.python.org/pypi/Warcat/
- Chat: irc://irc.efnet.org/archiveteam-bs (I'll be on #archiveteam-bs on EFnet)

Specification

This implementation is based loosely on draft ISO 28500 papers `WARC_ISO_28500_version1_latestdraft.pdf` and `warc_ISO_DIS_28500.pdf` which can be found at http://bibnum.bnf.fr/WARC/ .

File format

Here's a quick description:

A WARC file contains one or more Records concatenated together. Each Record contains Named Fields, newline, a Content Block, newline, and newline. A Content Block may be two types: {binary data} or {Named Fields, newline, and binary data}. Named Fields consists of string, colon, string, and newline.

A Record may be compressed with gzip. Filenames ending with `.warc.gz` indicate one or more gzip compressed files concatenated together.

Alternatives

Warcat is inspired by

- https://github.com/internetarchive/warc
- http://code.hanzoarchives.com/warc-tools

Development

Testing

Always remember to test. Continue testing:

```
python3 -m unittest discover -p '*_test.py'
nosetests3
```

To-do

- Smart archive join
- Regex filtering of records
- Generate index to disk (eg, for fast resume)
- Grab files like wget and archive them
- See TODO and FIXME markers in code
- etc.

CHAPTER 2

API Reference

Document model Model serialization and binary references

class `warcat.model.binary.`**`BytesSerializable`**
Metaclass that indicates this object can be serialized to bytes

`iter_bytes``()`
Return an iterable of bytes

class `warcat.model.binary.`**`StrSerializable`**
Metaclass that indicates this object can be serialized to str

`iter_str``()`
Return an iterable of str

class `warcat.model.binary.`**`BinaryFileRef`**
Reference to a file containing the content block data.

`file_offset`
When reading, the file is seeked to *file_offset*.

`length`
The length of the data

`filename`
The filename of the referenced data. It must be a valid file.

`file_obj`
The file object to be read from. It is important that this file object is not shared or race conditions will occur. File objects are not closed automatically.

Note: Either `filename` or `file_obj` must be set.

`get_file``(`*safe=True*, *spool_size=10485760*`)`
Return a file object with the data.

Parameters **`safe`** – If *True*, return a new file object that is a copy of the data. You will be responsible for closing the file.

Otherwise, it will be the original file object that is seeked to the correct offset. Be sure to not read beyond its length and seek back to the original position if necessary.

`iter_file`(*buffer_size=4096*)
Return an iterable of bytes of the source data

`set_file`(*file*, *offset=0*, *length=None*)
Set the reference to the file or filename with the data.

This is a convenience function to setting the attributes individually.

Content blocks and payload blocks

class `warcat.model.block.`**`ContentBlock`**

`iter_bytes`()
Return an iterable of bytes

classmethod **`load`**(*file_obj*, *length*, *content_type*)
Load and return *`BinaryBlock`* or *`BlockWithPayload`*

class `warcat.model.block.`**`BinaryBlock`**
A content block that is octet data

`get_file`(*safe=True*, *spool_size=10485760*)
Return a file object with the data.

Parameters **`safe`** – If *True*, return a new file object that is a copy of the data. You will be responsible for closing the file.

Otherwise, it will be the original file object that is seeked to the correct offset. Be sure to not read beyond its length and seek back to the original position if necessary.

`iter_bytes`()

`iter_file`(*buffer_size=4096*)
Return an iterable of bytes of the source data

classmethod **`load`**(*file_obj*, *length*)
Return a *`BinaryBlock`* using given file object

`set_file`(*file*, *offset=0*, *length=None*)
Set the reference to the file or filename with the data.

This is a convenience function to setting the attributes individually.

class `warcat.model.block.`**`BlockWithPayload`**(*fields=None*, *payload=None*)
A content block (fields/data) within a `Record`.

`fields`
`Fields`

`payload`
`Payload`

`binary_block`
If this block was loaded from a file, this attribute will be a *`BinaryBlock`* of the original file. Otherwise, this attribute is *None*.

`iter_bytes`()

`length`
Return the new computed length

classmethod **`load`**(*file_obj*, *length*, *field_cls*)
Return a *`BlockWithPayload`*

Parameters

- **`file_obj`** – The file object
- **`length`** – How much to read from the file
- **`field_cls`** – The class or subclass of `Fields`

class `warcat.model.block.`**`Payload`**
Data within a content block that has fields

`get_file`(*safe=True*, *spool_size=10485760*)
Return a file object with the data.

Parameters **`safe`** – If *True*, return a new file object that is a copy of the data. You will be responsible for closing the file.

Otherwise, it will be the original file object that is seeked to the correct offset. Be sure to not read beyond its length and seek back to the original position if necessary.

`iter_bytes`()

`iter_file`(*buffer_size=4096*)
Return an iterable of bytes of the source data

`set_file`(*file*, *offset=0*, *length=None*)
Set the reference to the file or filename with the data.

This is a convenience function to setting the attributes individually.

Constants and things

`warcat.model.common.`**`FIELD_DELIM_BYTES`** **= b'\r\n\r\n'**
Bytes CR LF CR LF

`warcat.model.common.`**`NEWLINE`** **= '\r\n'**
String CR LF

`warcat.model.common.`**`NEWLINE_BYTES`** **= b'\r\n'**
Bytes CR LF

Named fields

class `warcat.model.field.`**`Fields`**(*field_list=None*)
Name and value pseudo-map list

Behaves like a *dict* or mutable mapping. Mutable mapping operations remove any duplicates in the field list.

`add`(*name*, *value*)
Append a name-value field to the list

`clear`()

`count`(*name*)
Count the number of times this name occurs in the list

`get`(*name*, *default=None*)

`get_list`(*name*)
Return a list of values

index (*name*)
Return the index of the first occurance of given name

iter_bytes ()

iter_str ()

classmethod join_multilines (*value*, *lines*)
Scan for multiline value which is prefixed with a space or tab

keys ()

list ()
Return the underlying list

classmethod parse (*s*, *newline='\r\n'*)
Parse a named field string and return a *Fields*

values ()

class warcat.model.field.**HTTPHeader** (*field_list=None*, *status=None*)
Fields extended with a HTTP status attribute.

status
The *str* of the HTTP status message and code.

add (*name*, *value*)
Append a name-value field to the list

clear ()

count (*name*)
Count the number of times this name occurs in the list

get (*name*, *default=None*)

get_list (*name*)
Return a list of values

index (*name*)
Return the index of the first occurance of given name

iter_bytes ()

iter_str ()

join_multilines (*value*, *lines*)
Scan for multiline value which is prefixed with a space or tab

keys ()

list ()
Return the underlying list

classmethod parse (*s*, *newline='\r\n'*)

status_code

values ()

warcat.model.field.**HTTPHeaders**
Deprecated since version 2.1.1.

Name uses wrong inflection. Use *HTTPHeader* instead.

alias of *HTTPHeader*

class `warcat.model.field.`**`Header`**(*version='1.0'*, *fields=None*)

A header of a WARC Record.

`version`

A *str* containing the version

`fields`

The *`Fields`* object.

`VERSION` **= '1.0'**

`iter_bytes`()

`iter_str`()

classmethod **`parse`**(*b*)

Parse from *bytes* and return *`Header`*

A WARC record

class `warcat.model.record.`**`Record`**(*header=None*, *content_block=None*)

A WARC Record within a WARC file.

`header`

`Header`

`content_block`

A `BinaryBlock` or `BlockWithPayload`

`file_offset`

If this record was loaded from a file, this attribute contains an *int* describing the location of the record in the file.

`content_length`

`date`

`iter_bytes`()

classmethod **`load`**(*file_obj*, *preserve_block=False*, *check_block_length=True*)

Parse and return a *`Record`*

Parameters

- **`file_object`** – A file-like object.
- **`preserve_block`** – If *True*, content blocks are not parsed for fields and payloads. Enabling this feature ensures preservation of content length and hash digests.
- **`check_block_length`** – If *True*, the length of the blocks are checked to a serialized version by Warcat. This can be useful for checking whether Warcat will output blocks with correct whitespace.

`record_id`

`warc_type`

WARC model starting point

class `warcat.model.warc.`**`WARC`**

A Web ARChive file model.

Typically, large streaming operations should use *`open()`* and *`read_record()`* functions.

`iter_bytes`()

load (*filename*)
Open and load the contents of the given filename.

The records are located in `records`.

classmethod open (*filename*, *force_gzip=False*)
Return a logical file object.

Parameters

- **filename** – The path of the file. gzip compression is detected using file extension.
- **force_gzip** – Use gzip compression always.

read_file_object (*file_object*)
Read records until the file object is exhausted

classmethod read_record (*file_object*, *preserve_block=False*, *check_block_length=True*)
Return a record and whether there are more records to read.

See also:

`Record`

Returns A tuple. The first item is the `Record`. The second item is a boolean indicating whether there are more records to be read.

Archive process tools

class `warcat.tool.`**BaseIterateTool** (*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)
Base class for iterating through records

action (*record*)

postprocess ()

preprocess ()

process ()

class `warcat.tool.`**ConcatTool** (*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)

action (*record*)

postprocess ()

preprocess ()

process ()

class `warcat.tool.`**ExtractTool** (*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)

action (*record*)

postprocess ()

preprocess ()

process ()

class `warcat.tool.`**`ListTool`**(*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)

`action`(*record*)

`postprocess`()

`preprocess`()

`process`()

class `warcat.tool.`**`SplitTool`**(*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)

`action`(*record*)

`postprocess`()

`preprocess`()

`process`()

exception `warcat.tool.`**`VerifyProblem`**(*message*, *iso_section=None*, *major=True*)

`args`

`iso_section`

`major`

`message`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `warcat.tool.`**`VerifyTool`**(*filenames*, *out_file=None*, *write_gzip=False*, *force_read_gzip=None*, *read_record_ids=None*, *preserve_block=True*, *out_dir=None*, *print_progress=False*, *keep_going=False*)

`MANDATORY_FIELDS` = [’WARC-Record-ID’, ‘Content-Length’, ‘WARC-Date’, ‘WARC-Type’]

`action`(*record*)

`check_transfer_encoding`(*record*)

`postprocess`()

`preprocess`()

`process`()

`verify_block_digest`(*record*)

`verify_concurrent_to`(*record*)

`verify_content_type`(*record*)

`verify_filename`(*record*)

`verify_id_no_whitespace`(*record*)

`verify_id_uniqueness`(*record*)

`verify_mandatory_fields`(*record*)

`verify_payload_digest`(*record*)

`verify_profile`(*record*)

`verify_refers_to`(*record*)

`verify_segment_origin_id`(*record*)

`verify_segment_total_length`(*record*)

`verify_target_uri`(*record*)

`verify_warcinfo_id`(*record*)

Version info

`warcat.version.`**`short_version`** = **‘2.2’**
Short version in the form of N.N

Verification helpers

`warcat.verify.`**`parse_digest_field`**(*s*)
Return the algorithm name and digest *bytes*

`warcat.verify.`**`verify_block_digest`**(*record*)
Return *True* if the content block hash digest is valid

`warcat.verify.`**`verify_payload_digest`**(*record*)
Return *True* if the payload hash digest is valid

Utility functions

class `warcat.util.`**`DiskBufferedReader`**(*raw*, *disk_buffer_size=104857600*, *spool_size=10485760*)
Buffers the file to disk large parts at a time

`close`()
Flush and close the IO object.

This method has no effect if the file is already closed.

`closed`

`detach`()
Disconnect this buffer from its underlying raw stream and return it.

After the raw stream has been detached, the buffer is in an unusable state.

`fileno`()

`flush`()
Flush write buffers, if applicable.

This is not implemented for read-only and non-blocking streams.

`isatty`()

`mode`

`name`

`peek`(*n=0*)

`raw`

`read`(*n=None*)

read1()

Read and return up to n bytes, with at most one read() call to the underlying raw stream. A short result does not imply that EOF is imminent.

Returns an empty bytes object on EOF.

readable()

readinto()

readinto1()

readline()

Read and return a line from the stream.

If size is specified, at most size bytes will be read.

The line terminator is always b'n' for binary files; for text files, the newlines argument to open can be used to select the line terminator(s) recognized.

readlines()

Return a list of lines from the stream.

hint can be specified to control the number of lines read: no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds hint.

seek(*pos*, *whence=0*)

seekable()

tell()

truncate()

Truncate file to size bytes.

File pointer is left unchanged. Size defaults to the current IO position as reported by tell(). Returns the new size.

writable()

write()

Write the given buffer to the IO stream.

Returns the number of bytes written, which is always the length of b in bytes.

Raises BlockingIOError if the buffer is full and the underlying raw stream cannot accept more data at the moment.

writelines()

class warcat.util.**FileCache**(*size=4*)

A cache containing references to file objects.

File objects are closed when expired. Class is thread safe and will only return file objects belonging to its own thread.

get(*filename*)

put(*filename*, *file_obj*)

class warcat.util.**HTTPSocketShim**

close()

Disable all I/O operations.

closed
True if the file is closed.

detach()
Disconnect this buffer from its underlying raw stream and return it.

After the raw stream has been detached, the buffer is in an unusable state.

fileno()
Returns underlying file descriptor if one exists.

OSError is raised if the IO object does not use a file descriptor.

flush()
Does nothing.

getbuffer()
Get a read-write view over the contents of the BytesIO object.

getvalue()
Retrieve the entire contents of the BytesIO object.

isatty()
Always returns False.

BytesIO objects are not connected to a TTY-like device.

makefile(**args*, ***kwargs*)

read()
Read at most size bytes, returned as a bytes object.

If the size argument is negative, read until EOF is reached. Return an empty bytes object at EOF.

read1()
Read at most size bytes, returned as a bytes object.

If the size argument is negative or omitted, read until EOF is reached. Return an empty bytes object at EOF.

readable()
Returns True if the IO object can be read.

readinto()
Read bytes into buffer.

Returns number of bytes read (0 for EOF), or None if the object is set not to block and has no data to read.

readinto1()

readline()
Next line from the file, as a bytes object.

Retain newline. A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then). Return an empty bytes object at EOF.

readlines()
List of bytes objects, each a line from the file.

Call readline() repeatedly and return a list of the lines so read. The optional size argument, if given, is an approximate bound on the total number of bytes in the lines returned.

seek()
Change stream position.

Seek to byte offset pos relative to position indicated by whence: 0 Start of stream (the default). pos should be >= 0; 1 Current position - pos may be negative; 2 End of stream - pos usually negative.

Returns the new absolute position.

seekable()
Returns True if the IO object can be seeked.

tell()
Current file position, an integer.

truncate()
Truncate the file to at most size bytes.

Size defaults to the current file position, as returned by tell(). The current file position is unchanged. Returns the new size.

writable()
Returns True if the IO object can be written.

write()
Write bytes to file.

Return the number of bytes written.

writelines()
Write lines to the file.

Note that newlines are not added. lines can be any iterable object producing bytes-like objects. This is equivalent to calling write() for each element.

warcat.util.**append_index_filename**(*path*)
Adds `_index_xxxxxx` to the path.

It uses the basename aka filename of the path to generate the hex hash digest suffix.

warcat.util.**copyfile_obj**(*source*, *dest*, *bufsize=4096*, *max_length=None*, *write_attr_name='write'*)
Like `shutil.copyfileobj()` but with limit on how much to copy

warcat.util.**file_cache = <warcat.util.FileCache object>**
The `FileCache` instance

warcat.util.**find_file_pattern**(*file_obj*, *pattern*, *bufsize=512*, *limit=4096*, *inclusive=False*)
Find the offset from current position of pattern

warcat.util.**parse_http_date**(*s*)

warcat.util.**parse_http_response**(*file_obj*)
Parse and return `http.client.HTTPResponse`

warcat.util.**printable_str_to_str**(*s*)

warcat.util.**rename_filename_dirs**(*dest_filename*)
Renames files if they conflict with a directory in given path.

If a file has the same name as the directory, the file is renamed using `append_index_filename()`.

warcat.util.**sanitize_str**(*s*)
Replaces unsavory chracters from string with an underscore

warcat.util.**split_url_to_filename**(*s*)
Attempt to split a URL to a filename on disk

warcat.util.**strip_warc_extension**(*s*)
Removes `.warc` or `.warc.gz` from filename

`warcat.util.`**`truncate_filename_parts`**(*path_parts*, *length=160*)

Truncate and suffix filename path parts if they exceed the given length.

If the filename part is too long, the part is truncated and an underscore plus a 6 letter hex (_xxxxxx) suffix is appended.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index