Zero Buffer Documentation

Release

Alex Gaynor and David Reid

March 02, 2017

Contents

zero_buffer is a high-performance, zero-copy, implementation of a byte-buffer for Python.

```
from zero_buffer import Buffer

# Create a buffer which has space for 8192 bytes.
b = Buffer.allocate(8192)
with open(path, "rb") as f:
    # Read up to 8192 bytes from the file into the buffer
    b.read_from(f.fileno())
# Create a read-only view of the buffer, this performs no copying.
view = b.view()
# Split the view on colons, this returns a generator which yields sub-views
# of the view.
for part in view.split(b":"):
    print(part)
```

zero_buffer works on Python 2.6, 2.7, 3.3+, and PyPy.

CHAPTER 1

Installation

Install it with `pip`:

```
$ pip install zero_buffer
```

If you are installing `zero_buffer` on Ubuntu, you may need to run the following before installing it with `pip`.

```
$ sudo apt-get install build-essential libffi-dev python-dev
```

CHAPTER 2

Contents

API Reference

class zero_buffer.**Buffer**

A buffer is a fixed-size, append only, contigious region of memory. Once data is in it, that data cannot be mutated, however data can be read into the buffer with multiple calls.

classmethod allocate(*size*)

Parameters **size** (*int*) – Number of bytes.

Return Buffer The new buffer.

Allocates a new buffer of size bytes.

capacity

Returns the size of the underlying buffer. This is the same as what it was allocated with.

writepos

Returns the current, internal writing position, this increases on calls to *read_from()* and *add_bytes()*.

free

Returns the remaining space in the *Buffer*.

read_from(*fd*)

Parameters **fd** (*int*) – A file descriptor.

Return int Number of bytes read.

Raises

- **OSError** – on an error reading from the file descriptor.
- **EOFError** – when the read position of the file is at the end.
- **BufferFull** – when the buffer has no remaining space when called

Reads from the file descriptor into the *Buffer*. Note that the number of bytes copied may be less than the number of bytes in the file.

add_bytes(*b*)

Parameters **b** (*bytes*) – Bytes to copy into the buffer.

Return int Number of bytes copied into the buffer.

Raises **BufferFull** – when the buffer has no remaining space when called

Copies the bytes into the *Buffer*. Note that the number of bytes copied may be less than `len(b)` if there isn't space in the *Buffer*.

view(*start=0*, *stop=None*)

Parameters

- **start** (*int*) – The byte-offset from the beggining of the buffer.
- **stop** (*int*) – The byte-offset from start.

Return BufferView

Raises **ValueError** – If the stop is before the start, if the start is negative or after the writepos, or if the stop is after the writepos.

Returns a view of the buffer's data. This does not perform any copying.

class zero_buffer.**BufferView**

A buffer view is an immutable, fixed-size, view over a contigious region of memory. It exposes much of the same API as `bytes`, except most methods return BufferViews and do not make copies of the data. A buffer view is either a view into a *Buffer* or into another *BufferView*.

__bytes__()

Returns a copy of the contents of the view as a `bytes`.

__len__()

Returns the length of the view.

__eq__(*other*)

Checks whether the contents of the view are equal to `other`, which can be either a `bytes` or a *BufferView*.

__contains__(*needle*)

Returns whether or not the `needle` exists in the view as a contigious series of bytes.

__getitem__(*idx*)

If `idx` is a `slice`, returns a *BufferView* over that data, it does not perform a copy. If `idx` is an integer, it returns the ordinal value of the byte at that index.

Unlike other containers in Python, this does not support slices with steps (`view[::2]`).

__add__(*other*)

Parameters **other** (BufferView) –

Returns a *BufferView* over the concatenated contents. If `other` is contigious with `self` in memory, no copying is performed, otherwise both views are copied into a new one.

find(*needle*, *start=0*, *stop=None*)

The same as `bytes.find()`.

index(*needle*, *start=0*, *stop=None*)

The same as `bytes.index()`.

rfind(*needle*, *start=0*, *stop=None*)

The same as `bytes.rfind()`.

rindex(*needle*, *start=0*, *stop=None*)

The same as `bytes.rindex()`.

split(*by*, *maxsplit=-1*)

Similar to `bytes.split()`, except it returns an iterator (not a `list`) over the results, and each result is a *BufferView* (not a `bytes`).

splitlines(*keepends=False*)
Similar to `bytes.splitlines()`, except it returns an iterator (not a `list`) over the results, and each result is a *BufferView* (not a `bytes`).

isspace()
The same as `bytes.isspace()`.

isdigit()
The same as `bytes.isdigit()`.

isalpha()
The same as `bytes.isalpha()`.

strip(*chars=None*)
The same as `bytes.strip()` except it returns a *BufferView* (and not a `bytes`).

lstrip(*chars=None*)
The same as `bytes.lstrip()` except it returns a *BufferView* (and not a `bytes`).

rstrip(*chars=None*)
The same as `bytes.rstrip()` except it returns a *BufferView* (and not a `bytes`).

write_to(*fd*)

Parameters **fd** (*int*) – A file descriptor.

Return int Number of bytes written.

Raises **OSError** – on an error writing to the file descriptor.

Writes the contents of the buffer to a file descriptor. Note that the number of bytes written may be less than the number of bytes in the buffer view.

class zero_buffer.**BufferCollator**
A buffer collator is a collection of *BufferView* objects which can be collapsed into a single *BufferView*.

__len__()
Returns the sum of the lengths of the views inside the collator.

append(*view*)

Parameters **view** (BufferView) –

Adds the contents of a view to the collator.

collapse()
Collapses the contents of the collator into a single *BufferView*. Also resets the internal state of the collator, so if you call it twice successively, the second call will return an empty *BufferView*.

Symbols

A

B

C

F

I

L

R

S

V

W