

# PEP3118: Revising the Buffer Protocol

## Why it's the coolest thing ever<sup>1</sup>

Neil Muller

August 2, 2008

---

<sup>1</sup>May not be true for many values of coolest, or ever

## The Buffer interface in python 2.5

- ▶ A Python object can expose it's internal data to other objects, via the PyBufferProcs methods
- ▶ Allows access to large data chunks without unnecessary copies (good for file I/O, interfacing between modules, etc.)
- ▶ Fairly simple + limited interface

```
Py_ssize_t (*readbufferproc) (PyObject *self,  
    Py_ssize_t segment, void **ptrptr)  
Py_ssize_t (*writebufferproc) (PyObject *self,  
    Py_ssize_t segment, void **ptrptr)  
Py_ssize_t (*segcountproc) (PyObject *self,  
    Py_ssize_t *lenp)  
Py_ssize_t (*charbufferproc) (PyObject *self,  
    Py_ssize_t segment, const char **ptrptr)  
Py_TPFLAGS_HAVE_GETCHARBUFFER
```

- ▶ Via readbufferproc, caller can obtain a void pointer to the data, and the data length

## PIL's internal memory representation

```
char mode[4+1];      /* Band names ("1", "L", "P", "RGB", etc.) */
int type;           /* Data type */
int depth;         /* Depth */
int bands;         /* Number of bands */
int xsize, ysize;
ImagingPalette palette;
char **image;      /* raster data.*/ */
char *block;      /* Set if data is allocated in a single block */
int pixelsize;   /* Size of a pixel, in bytes (1, 2 or 4) */
int linesize;   /* Size of a line, in bytes (xsize * pixelsize) */
```

- ▶ Image is stored as a single array or an array of pointers, each pointer is a single line of in the image
- ▶ Maps nicely onto segment structure of buffer protocol, but no way of requesting other information like pixelsize

## NumPy's internal memory representation

```
char *data;          /* data buffer */
int nd;              /* number of dimensions */
numpy_intp *dimensions; /* size in each dimension */
numpy_intp *strides; /* bytes to jump to get to next */
                    /* element in each dimension */
PyArray_Descr *descr; /* Pointer to type structure for */
                    /* for contents of array */
int flags;           /* Flags describing array */
                    /* Flags include: NPY_C_CONTIGUOUS, */
                    /* NPY_F_CONTIGUOUS, NPY_NOTSWAPPED */
                    /* indicating data layout and endianness */
```

- ▶ Can't get anything other than a pointer to the data via buffer protocol

# Houston, We Have A Problem

- ▶ If interfacing PIL and NumPy, both sides need detailed knowledge of the internal layout to manage conversions back and forth
  - ▶ NumPy doesn't naively support the array of pointers memory layout, since may not be contiguous
  - ▶ PIL doesn't understand strides (can be fudged via array of pointers, but not ideal)
  - ▶ PIL doesn't understand different endianness, nor F\_CONTIGUOUS memory layouts
- ▶ Also need to pass information about what's in the buffer object (unsigned char, 32bit int, etc.) to be useful, but can't be done as part of the buffer protocol.

# Houston, We Have A Problem

- ▶ If interfacing PIL and NumPy, both sides need detailed knowledge of the internal layout to manage conversions back and forth
  - ▶ NumPy doesn't naively support the array of pointers memory layout, since may not be contiguous
  - ▶ PIL doesn't understand strides (can be fudged via array of pointers, but not ideal)
  - ▶ PIL doesn't understand different endianness, nor F\_CONTIGUOUS memory layouts
- ▶ Also need to pass information about what's in the buffer object (unsigned char, 32bit int, etc.) to be useful, but can't be done as part of the buffer protocol.

*The buffer protocol doesn't actually do anything useful in what should be an ideal use case.*

# Buffer Protocol Problems

- ▶ Can get pointer to part of the memory via segments (like PIL), but object needs to support this
  - ▶ caller needs to understand this as well, and specifically code for non-contiguous data in this form (and most don't)
- ▶ No type information - only type awareness is in `charbufferproc`, which raises an error if data can't be treated as characters
  - ▶ This is seldom very useful.
  - ▶ `charbufferproc` method often not be supported
  - ▶ Given need for detailed knowledge, benefits of `charbufferproc` over `(const char *) ptrptr` debatable.
- ▶ Data is still owned by the object, but no mechanism for establishing that the caller no longer needs the view.
  - ▶ Usually avoided by ensuring the views are short lived and the GIL is never released while the view is needed
  - ▶ This approach is hardly ideal.

## PEP3118: Fixing the buffer protocol

- ▶ Provide information about buffer contents
- ▶ Support common memory layouts better
- ▶ Provide support for releasing views
- ▶ Cleanup interface

## Details

Replace existing approach with 2 functions:

```
typedef int (*getbufferproc)(PyObject *obj,  
                             PyBuffer *view, int flags)
```

- ▶ Request a view of data from existing object obj
- ▶ Return codes no indicate success or failure, not size
- ▶ Details encoded in PyBuffer object
- ▶ No longer separate write + read requests, now handled by flags.
- ▶ Limitations on results (must be contiguous, etc.) also handled via flags.

```
typedef void (*releasebufferproc)(PyObject *obj,  
                                  PyBuffer *view)
```

- ▶ Releases the view
  - ▶ Implementation of view tracking left to implementations
  - ▶ But must make sure view is valid until releasebufferproc is called.

# PyBuffer

```
struct bufferinfo {  
    void *buf;  
    Py_ssize_t len;  
    int readonly;  
    const char *format;  
    int ndim;  
    Py_ssize_t *shape;  
    Py_ssize_t *strides;  
    Py_ssize_t *suboffsets;  
    Py_ssize_t itemsize;  
    void *internal;  
};
```

- ▶ NumPy style objects managed via ndim, shape + strides
- ▶ PIL style via ndim, shape + suboffsets
- ▶ Object structure describe by format using struct syntax

# Data format Description

- ▶ Some additions to the struct syntax
  - ▶ O - pointer to python object
  - ▶ T{} - nested struct definition
  - ▶ :name - named elements
  - ▶ (k1, k2, .. kn) - multi-dimensional arrays
  - ▶ Z <specifier> - complex number

'd' <--> Python float

'Zd' <--> Python complex

Nested structure:

```
struct {
    int ival;
    struct {
        unsigned short sval;
        unsigned char bval;
    } sub;
}
"""i:ival:
   T{
     H:sval:
     B:bval:
   }:sub:
"""
```

## Other cool things in PEP3118

- ▶ `memoryview` - Python object for views. Supports multi-dimensional slicing
- ▶ `PyMemoryView_GetContiguous` - return a contiguous view of the objects memory
  - ▶ Make a copy if data isn't already contiguous
  - ▶ Support for copying data back when `MemoryView` is deleted.
- ▶ `PyObject_CopyToObject` - copy from contiguous memory into the buffer of `obj`.
- ▶ Will also be back-ported to python 2.X

# Current status

- ▶ Py3k
  - ▶ Some changes to struct module outstanding, to support extended descriptions (issue 2395)
  - ▶ Memoryview is incomplete (issue 2394)
- ▶ Python trunk
  - ▶ Most of the back-port work targeted for 2.7
  - ▶ Most of the PyBuffer API is exposed via `abstract.c`, so objects can support it in python 2.6