bufarray: Working with C Arrays in Python

Release 1.2

April 2, 2002

Abstract

bufarray is a Python module which provides services to access contiguous C arrays of arbitrary C type and dimension, as Python sequences or Python buffer objects, in a flexible but type-safe and memory-safe manner.

Contents

List of Figures

List of Tables

```
#include <pythonplus.h>
#include <bufarray.h>
```

Figure 1: bufarray includes

1 Introduction

Most technical application C API's (scientific, engineering…) pass arguments by means of pointers to pre-allocated contiguous arrays.
`bufarray` provides the functionality to efficiently and safely work in Python with such objects, and it relies on the Python `buffer` interface to directly access arrays already referenced by other objects, without the overhead of memory duplication.

2 Package description

Once build, `bufarray` works as a standalone Python package, with no dependency; it is organized as a *tuttidolce component [1].*

`bufarray` builds from source as a tuttidolce package [2].

Header definitions for `bufarray` C programming are given in Figure 1.

3 `Ctype` objects

3.1 Python

`Ctype` is a C extension type, and provides a means to describe C types in Python , including casting rules.

class *`Ctype`* *`[Object]`*

`Ctype` instances are created with the C method C `ctype_export->PyCtypenew()` exported to Python (see `CtypeExport` in 5 and `struct ctype_export` in 3.2).

`alignment` *: integer readonly*
Alignment constraint on the C type.

`readonly` *: Ctype—None*
]arraytype `Ctype` of item array elements, if items are `bufarray`, or `None`.

`castclass` *: Ctype readonly*
5 Instance defining the cast class. `bufarray` instances can only be converted to `bufarray` with a `ctype` with equal `castclass` attribute.

`cellsmgt` *: CObject readonly*
Pointer to the C `struct ccellsmgt` object defining the method functions for collective management of array items.

`get` *: CObject readonly*
Pointer to the C `cget_f` function which will return a `PyObject_t` object from an array item.

`set` *: CObject readonly*
Pointer to the C `cset_f` function which will set an array item from a `PyObject_t`.

sizeof *: integer readonly*
Size of an array item (as returned from C `sizeof`*).*

array*(dim: integer,* name: string[=] *) : ctype*
Return a new `ctype` *instance describing an array of* `dim` `self`*.*

3.2 C

cget_f
`typedef PyObject_t (WSTDCALL* cget_f)( bufarray_t self, idx_v idx);`
Get item callback.

cset_f
`typedef int (WSTDCALL* cset_f)( bufarray_t self, idx_v idx, PyObject_t oval);`
Set item callback.

ccopycells_f
`typedef int (WSTDCALL* ccopycells_f)( bufarray_t self, idx_v first, idx_v ncell);`
Callback for copying a contiguous block of items..

cfreecells_f
`typedef void (WSTDCALL* cfreecells_f)( bufarray_t self, idx_v first, idx_v ncell);`
Callback for freeing resources associated to a contiguous block of items..

struct ccellsmgt
`struct ccellsmgt {ccopycells_f copy; cfreecells_f free; };`
`free` *can be NULL .*

ctype_t
`typedef struct ctype* ctype_t;`
C `Ctype` *object type*

struct ctype_export
```
struct ctype_export
{
  PyTypeObject* type;
  PyObject_t (WSTDCALL* new)( char const* name, int sizof, int alignment,
      cget_f get, cset_f set,
      struct ccellsmgt* cellsmgt,
      ctype_t castclass);
  PyObject_t (WSTDCALL* array)( ctype_t arraytype, int dim, char const* name);
};
```

Structure for export of the `Ctype` *static API features (see* `CtypeExport` *in 5).*

`type` *will point to the* `Ctype` *type.*
`new` *is the* `Ctype` *object factory.*

4 bufarray objects

4.1 Python

class ***bufarray*** *[sequence, buffer]*

bufarray objects provide type-safe access to contiguous C arrays.
They work as Python sequences of fixed length, that may or may not be writable depending upon their readonly attribute.
Slice operations are implemented so as to work efficiently (useful for copy or initialisation operations).
Raw memory is accessible through the buffer interface of the object.

len will return the number of items in the array.

ctype *: Ctype readonly*
Data type of the array.

cobject *: Cobject readonly*
C pointer to the array.

parent *: Object readonly*
Parent object from whom a reference to the memory has been borrowed. If parent is None , the object is the original owner of the memory, for Python .

readonly *: boolean readonly*
Is true if ond only if array items cannot be modified

setReadOnly *() : None*
set readonly to true .

4.2 C

bufarray_t
typedef struct bufarray bufarray_t;*
C bufarray object type

boolean ***PyBufarray_check*** *(PyObject_t obj)*
Macro: return true if obj is a bufarray object (and can be cast to bufarray_t.

int ***PyBufarray_sizeof*** *(bufarray_t obj)*
Macro: return the total size (in bytes) of the obj's array..

void ***PyBufarray_setzero*** *(bufarray_t obj)*
Macro: reset the array to 0

ctype_t ***PyBufarray_castclass*** *(bufarray_t obj)*
Macro: return cast class of obj.

int ***PyBufarray_cellsize*** *(bufarray_t obj)*
Macro: return byte size of a cell of obj.

typ ***PyBufarray_item*** *(bufarray_t obj, idx_v idx, typ)*
Macro: return pointer to the idx^{th} item of obj.

struct bufarray_export

```
struct bufarray_export
{
  PyTypeObject* type;
  PyObject_t (WSTDCALL* new)( ctype_t ctype, idx_v ncell);
  PyObject_t (WSTDCALL* copy)( ctype_t ctype, idx_v ncell, void* ptr);
  PyObject_t (WSTDCALL* from_cobject)( ctype_t ctype, idx_v ncell,
       PyObject_t cobj, int readonly);
  PyObject_t (WSTDCALL* from_buffer)( ctype_t ctype, PyObject_t buffer,
      int readonly, size_t offset, size_t siz);
};
```

`type` will point to the `bufarray` type.
`copy`, `from_cobject`, and `from_buffer` are `bufarray` object factory functions.

A pointer to a static instance of this structure is exported in `bufarray` (see `BufarrayExport` in 5).

5 `bufarray` module

readArray*(obj: sequence — buffer,* ctype: Ctype[= *None*] *) : bufarray*
`readArray` is the workhorse of the `bufarray` module. It will do its best to convert `obj` to a `bufarray` object, of the `ctype` type, if the latter argument is given.

If `obj` is already a `bufarray`, `readArray( obj)` will return `obj`.

Conversion rules from `Numeric` [1] and Python `array` objects are defined in `bufarray.bufarray_Numeric` and `bufarray.bufarray_array`, respectively.

All global rules are maintained in the `bufarray.castMap` dictionary.

Bufarray*(arg: sequence — buffer — integer,* ctype: Ctype[= *None*], readonly: boolean[= *0*], cobject: Cobject[= *None*], *) :*
This is a lower-level `bufarray` factory than `readarray()`.

`readonly` = true will turn on the `readonly` flag.

If `arg` is an `integer`, it will define the number of `ctype` items of the result, as created from `cobject`.

If `arg` is an `integer` and `cobject` is None , `Bufarray()` will do a new memory array allocation, initialise it, and return a `bufarray` object pointing to the new array.

castMap *: dictionary*
Python class to `ctype` compatibility class dictionary (see `castclass` member in)

exception *`CastError [TypeError]`*
Raised when `Ctype` cast rules cannot be resolved.

Type_Ctype *: Type*
`Ctype` type

Type_Bufarray *: Type*
`Bufarray` type

charCtype *: Ctype*
C `char`

[1] Numeric version 21.0 or later is required (fix required on Numeric bug on Python buffer interface implementation).

castclass is charCtype.

ucharCtype *: Ctype*
C uchar
castclass is charCtype.

byteCtype *: Ctype*
C byte
castclass is charCtype.

ubyteCtype *: Ctype*
C ubyte
castclass is charCtype.

shortCtype *: Ctype*
C short
castclass is shortCtype.

ushortCtype *: Ctype*
C unsigned short
castclass is shortCtype.

intCtype *: Ctype*
C int
castclass is intCtype.

uintCtype *: Ctype*
C unsigned int
castclass is intCtype.

longCtype *: Ctype*
C long
castclass is longCtype.

ulongCtype *: Ctype*
C unsigned long
castclass is longCtype.

floatCtype *: Ctype*
C float
castclass is floatCtype.

doubleCtype *: Ctype*
C double
castclass is doubleCtype.

voidstarCtype *: Ctype*
*C void**
castclass is voidstarCtype.

ctypeExport *: CObject (struct ctype_export*)*
See the definition of struct ctype_export in 3.2. This data is designed to be retrieved in C with an import operation, wherever needed.*

BufarrayExport *: CObject (struct bufarray_export*)*
See the definition of struct bufarray_export in 4.2. This data is designed to be retrieved in C with an import operation, wherever needed.*

```
import mmap, bufarray, os

rgbpixelType = bufarray.ubyteCtype.array( 3, 'rgbpixel') # 3 bytes per pixel
line1024Type = rgbpixelType.array( 1024) # 1024 pixels per line
image1024x512Type = line1024Type.array( 512) # 512 lines per image

videopath = 'myvideo1024x512x500.rgb'
videofile = open( videopath, 'r+b')
videomap = mmap.mmap( videofile.fileno(), os.stat( videopath).st_size)
videofile.close()
del videofile

videoarray = readArray( videomap, image1024x512Type)
print len( videoarray) # number of 1024 x 512 images in the video

videoseq = videoarray[ 40 : 100] # images 40 to 99

# work on seq as series of pixels
videoseqpixels = readArray( videoseq, rgbpixelType)

redpixel = bufarray.readArray( (255, 0, 0), rgbpixelType)

videoseqpixels[ :] = redpixel # images 40 to 99 are now red

videoarray[ 400:450] = redpixel # same for images 400 to 449

videomap.close() # save the modified video sequence
```

Figure 2: RGB video: access and edition of a large digital video file

6 Examples

`bufarray` examples at both C and Python levels are available in regression tests, and in the source of JPE and OpenGLTK.

6.1 RGB video

See Figure 2.

6.2 Input Array

See Figure 3.

6.3 Output Array

See Figure 4.

Python wrapper:

```
def samplefun( inputpoint):
    '''
    inputpoint: sequence( 3 * float)
    '''
    import mylib, bufarray
    inputpoint = bufarray.readArray( inputseq, bufArray.floatCtype)
    assert len( inputpoint) == 3, len( inputpoint)
    mylib.samplefun( inputpoint)
```

C wrapper:

```
#include <pythonplus.h>
#include <bufarray.h>
...

void samplefun( float const point[ 3]);

static PyObject_t mylib_samplefun( PyObject_t self, PyObject_t args)
{
  void const* ptr;
  int sz;

  /* t#: read-only buffer */
  if (NOT PyArg_ParseTuple( args, "t#:samplefun", &ptr, &sz)) return NULL;
  if (sz != sizeof float * 3) /* check size */
      return PyErr_Format( PyExc_TypeError, __FILE__
                           ":%i %i-byte array of expected, %i-byte provided",
                           __LINE__, sz);

  samplefun( (float const*)point);

  return Py_INCREF( Py_None), Py_None;
};

...
 {"samplefun", mylib_samplefun, METH_VARARGS,
    "samplefun( buffer[ const float * 3])"},
...
```

Figure 3: input array wrapping

Python wrapper:

```
def samplefun():
    '''
    return bufarray( 3 * float)
    '''
    import mylib, bufarray
    result = Bufarray( 3, bufArray.floatCtype)
    mylib.samplefun( result)
    return result
```

C wrapper:

```
#include <pythonplus.h>
#include <bufarray.h>
...

void samplefun( float point[ 3]);

static PyObject_t mylib_samplefun( PyObject_t self, PyObject_t args)
{
  void const* ptr;
  int sz;

  /* w#: read-write buffer */
  if (NOT PyArg_ParseTuple( args, "w#:samplefun", &ptr, &sz)) return NULL;
  if (sz != sizeof float * 3) /* check size */
      return PyErr_Format( PyExc_TypeError, __FILE__
                           ":%i %i-byte array of expected, %i-byte provided",
                           __LINE__, sz);

  samplefun( (float const*)point);

  return Py_INCREF( Py_None), Py_None;
};

...
 {"samplefun", mylib_samplefun, METH_VARARGS,
    "samplefun( buffer[ float * 3])"},
...
```

Figure 4: output array wrapping

Python wrapper:

```
def samplefun( updatearray):
    import mylib, bufarray
    updatearray = readArray( updatearray, bufArray.floatCtype)
    mylib.samplefun( updatearray, len( updatearray))
```

Swig interface:

```
%{
#include <pythonplus.h>
#include <bufarray.h>
...
void samplefun( float* values, int nvalues);
%}

void samplefun( float* values, int nvalues);

%typemap( python, in) float*
{
  int buffer_len;
  if (PyObject_AsWriteBuffer( $source, (void**)&$target, &buffer_len))
    return NULL;
  if (! $target) return PyErr_Format( PyExc_ValueError,
                                      "NULL buffer not accepted");
}
```

Figure 5: Swig array wrapping

6.4 Swig

See Figure 5.

6.5 Creation a `Ctype` instance for packed YUV pixels

See Figures 6 to 7.

A Source Code and Informations

The source code can be obtained from the CVS repository associated to http://jpe.sourceforge.net*.*

Other informations can be obtained by contacting the author (Frédéric Giacometti) at mailto:fred@arakne.com*.*

B Copyright and Licence Information

The source code and this document are subject to the following copyright:

```
#include <pythonplus.h>
#include <bufarray.h>
...

struct yuv
{
  unsigned y:8; /*8-bit luminance*/
  unsigned u:4; /*4-bit first chrominance component*/
  unsigned v:4; /*4-bit second chrominance component*/
};

typedef struct yuv* yuv_t;

static PyObject_t yuv_get( bufarray_t self, idx_v idx)
{
  yuv_t yuv;
  yuv = (yuv_t)self->ptr + idx;
  return Py_BuildValue( "iii", (int)yuv->y, (int)yuv->u, (int)yuv->v);
}

static int yuv_set( bufarray_t self, idx_v idx, PyObject_t oval)
{
  int y, u, v;
  yuv_t yuv;

  if (NOT PyArg_ParseTuple( oval, "iii", &y, &u, &v)) return 1;

  yuv = (yuv_t)self->ptr + idx;

  yuv->y = y;
  yuv->u = u;
  yuv->v = v;
  return 0;
}
```

Figure 6: Defining a `Ctype` object for yuv pixels: callback functions

```
void inityuv( void)
{
  struct ctype_export* ctype_export;
  ctype_t yuvCtype;

  module = Py_InitModule( "yuv", yuv_methods);

  ctype_export = (struct ctype_export*)PypImport_ModuleCobjAttr( "bufarray",
                                                                 "ctypeExport");
  if (NOT ctype_export) return;

  yuvCtype = ctype_export->new( "yuv_pixel",         /*name*/
                                sizeof (struct yuv), /*sizof*/
                                1,                   /*alignment*/
                                yuv_get,             /*get*/
                                yuv_set,             /*set*/
                                NULL,                /*cellsmgt*/
                                NULL); /*castclass: NULL <=> self*/

  if (PyObject_SetAttrString( module, "yuvCtype", yuvCtype))
    {
      Py_DECREF( yuvCtype);
      return;
    }
  Py_DECREF( yuvCtype);
}
```

Figure 7: Defining a `Ctype` object for yuv pixels: module initialization function

C Acknowledgements

This work was funded in part under NIH grant P41 RR08605 to the National Biomedical Computation Resource (NBCR), and carried out under the direction of Michel Sanner, in the laboratory of Arthur Olson, The Scripps Research Institute.

References

[1] tuttidolce: organization reference*. Not written yet.*

[2] tuttidolce: configuration and build reference*. Not written yet.*

Index