# Writing extensions

All Versions

Improvements requested:

## Examples

### Hello World with C Extension

The following C source file (which we will call hello.c for demonstration purposes) produces an extension module named hello that contains a single function greet() :

```
#include <Python.h>
#include <stdio.h>

#if PY_MAJOR_VERSION >= 3
#define IS_PY3K
#endif

static PyObject *hello_greet(PyObject *self, PyObject *args)
{
    const char *input;
    if (!PyArg_ParseTuple(args, "s", &input)) {
        return NULL;
    }
    printf("%s", input);
    Py_RETURN_NONE;
}

static PyMethodDef HelloMethods[] = {
    { "greet", hello_greet, METH_VARARGS, "Greet the user" },
    { NULL, NULL, 0, NULL }
};

#ifdef IS_PY3K
static struct PyModuleDef hellomodule = {
    PyModuleDef_HEAD_INIT, "hello", NULL, -1, HelloMethods
};

PyMODINIT_FUNC PyInit_hello(void)
{
    return PyModule_Create(&hellomodule);
}
#else
PyMODINIT_FUNC inithello(void)
{
    (void) Py_InitModule("hello", HelloMethods);
}
#endif
```

To compile the file with the gcc compiler, run the following command in your favourite terminal:

`gcc /path/to/your/file/hello.c -o /path/to/your/file/hello`

To execute the greet() function that we wrote earlier, create a file in the same directory, and call it hello.py

```
import hello          # imports the compiled library
hello.greet("Hello!") # runs the greet() function with "Hello!" as an argument
```

### C Extension Using c++ and Boost

This is a basic example of a *C Extension* using C++ and Boost .

#### C++ Code

C++ code put in hello.cpp:

```
#include <boost/python/module.hpp>
#include <boost/python/list.hpp>
#include <boost/python/class.hpp>
#include <boost/python/def.hpp>

// Return a hello world string.
std::string get_hello_function()
{
   return "Hello world!";
}

// hello class that can return a list of count hello world strings.
class hello_class
```

```
{
public:

   // Taking the greeting message in the constructor.
   hello_class(std::string message) : _message(message) {}

   // Returns the message count times in a python list.
   boost::python::list as_list(int count)
   {
      boost::python::list res;
      for (int i = 0; i < count; ++i) {
         res.append(_message);
      }
      return res;
   }

private:
   std::string _message;
};


// Defining a python module naming it to "hello".
BOOST_PYTHON_MODULE(hello)
{
```

To compile this into a python module you will need the python headers and the boost libraries. This example was made on Ubuntu 12.04 using python 3.4 and gcc. Boost is supported on many platforms. In case of Ubuntu the needed packages was installed using:

```
sudo apt-get install gcc libboost-dev libpython3.4-dev
```

Compiling the source file into a .so-file that can later be imported as a module provided it is on the python path:

```
gcc -shared -o hello.so -fPIC -I/usr/include/python3.4 hello.cpp -lboost_python-py34 -lboost_sys
```

The python code in the file example.py:

```
import hello

print(hello.get_hello())

h = hello.Hello("World hello!")
print(h.as_list(3))
```

Then `python3 example.py` will give the following output:

```
Hello world!
['World hello!', 'World hello!', 'World hello!']
```

## Passing an open file to C Extensions

Pass an open file object from Python to C extension code.

You can convert the file to an integer file descriptor using `PyObject_AsFileDescriptor` function:

```
PyObject *fobj;
int fd = PyObject_AsFileDescriptor(fobj);
if (fd < 0){
    return NULL;
}
```

To convert an integer file descriptor back into a python object, use `PyFile_FromFd` .

```
int fd; /* Existing file descriptor */
PyObject *fobj = PyFile_FromFd(fd, "filename","r",-1,NULL,NULL,NULL,1);
```

## Syntax

## Parameters

Remarks