Easy extending guide

Introduction

“... Boost.Python library is designed to wrap C++ interfaces non-intrusively, so that you shouldn't have to change the C++ code at all in order to wrap it. ...”

– Boost.Python tutorials

The previous statement is almost true. There are few use cases that the library doesn't support. This guide lists some of them.

Pointer to function

Boost.Python doesn't handle "pointer to function" functionality. You can't pass it as a function argument or keep it as a member variable.

Work-around: use “Command” design pattern (http://en.wikipedia.org/wiki/Command_pattern)

Problematic function arguments types

- native array - Boost.Python doesn't handle it. For example next function can't be exposed to Python as is: `int write( int* data, size_t size ){...}`
  Except technical reasons, there is a mental one: such interface is not intuitive for Python developers. They expect to pass single argument, for example look at built-in “file.write” method.
  Work-around:
  1. With a small help from the developer, Py++ generates code which feet well into Python developer mental model. Pure member functions is an exception – Py++ doesn't handle them right now.
  2. Use “std::vector<...>” or “std::basic_string<...>” class.
- immutable by reference. Python defines few fundamental types as "immutable". The value of instance of this type could not be changed after construction. Try to avoid passing immutable types by reference as function arguments or return type.
  C++ types mapped to immutable ones:
  - char
  - signed char
  - unsigned char
  - wchar_t
  - short int

- short unsigned int
- bool
- int
- unsigned int
- long int
- long unsigned int
- long long int
- long long unsigned int
- float
- double
- long double
- complex double
- complex long double
- complex float
- **std::string, std::wstring**
- **C++ enumeration**
- **smart pointer** – this is a special case

Work around:

- Just don't pass them by reference :-), thank you
- With small help from the developer, Py++ handles this issue, but the resulting interface is ugly.

- "void*". In most cases, we use "void*" when we want to deal with a memory block. Also Python provides support for this functionality, we steal haven't found an easy and intuitive way to expose it.
  Work around:
  - Please avoid "void*" usage
  - We do able to expose such functionality, but user will be required to use "ctypes" package.

Memory management

- If you want to return a new object and say that the user is responsible for it, life time – use "std::auto_ptr" class.
  We don't have, but using it we gain few things:
  - the memory management contract is obvious
  - automation – Py++ handles such functions automatically
- Same is true when you want to take a responsibility of object life-time. Use "std::auto_ptr" as a function argument, instead of raw pointer.

STL containers

Py++, via Indexing Suite version 2, supports next STL containers: vector, deque, list, map, multimap, hash_map, set, hash_set. Support for additional containers can be added on demand, but this is time consuming operation.

Multi-threading

Boost.Python is not thread-aware/compatible with multiple interpreters!