Subject: Re: PythonPP - tiny C++ wrappers for Python C API
Posted by Novo on Tue, 12 Apr 2011 14:48:55 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 12 April 2011 03:25hi novo

i'd like to compare the features / API invocation of PythonPP to that one of boost.Python. could you help here? since there is no API reference

yesterday i have added a BoostPyTest to bazaar. could you provide a similar one?

Hi kohait00,
I updated source code of PythonPP. I also added an example of a module, which unfortunately doesn't work yet.

```
#include <Core/Core.h>
#include <PythonPP/PythonPP.h>
#include <PythonPP/PythonPP_ext.h>

////////////////////////////////////////////////////////////////////////////
// Compatibility macros
//
// From Python 2.2 to 2.3, the way to export the module init function
// has changed. These macros keep the code compatible to both ways.
//
#if PY_VERSION_HEX >= 0x02030000
#  define PYDBAPI_MODINIT_FUNC(name)        PyMODINIT_FUNC name(void)
#else
#  define PYDBAPI_MODINIT_FUNC(name)        DL_EXPORT(void) name(void)
#endif

using namespace Upp;

namespace python
{

 class Test : public pythonpp::ExtObject<Test>
 {
 public:
  Test();

  pythonpp::Object Foo ( const pythonpp::Tuple& args );
  pythonpp::Object Boo ( const pythonpp::Tuple& args );
 };

 Test::Test()
 {
```

```
  ROAttr( "__class__", GetTypeObject() );
  PrepareForPython(this);
 }

 pythonpp::Object Test::Foo ( const pythonpp::Tuple& /*args*/ )
 {
  Cout() << "This is a UPP call !" << EOL;
 }

 pythonpp::Object Test::Boo ( const pythonpp::Tuple& /*args*/ )
 {
  Cout() << "This is Boo !" << EOL;
 }
}

////////////////////////////////////////////////////////////////////////////

static struct PyMethodDef methods[] = {
 { NULL, NULL }
};

////////////////////////////////////////////////////////////////////////////
// Module initialization
PYDBAPI_MODINIT_FUNC ( initPythonPPModule )
{
 pythonpp::ModuleExt::Declare ( "PythonPPModule", methods );
 PyObject *module = pythonpp::ModuleExt::GetPyModule();

 // Declare class Test.
 python::Test::
  Def ( "Foot", &python::Test::Foo, "Method #1" ).
  Def ( "Boo", &python::Test::Boo, "Method #2" );
 python::Test::Declare ( "PythonPPModule.Test" );

 if ( PyType_Ready ( &python::Test::GetType() ) == -1 )
 {
  return;
 }

 if ( PyModule_AddObject ( module, const_cast<char*> ( "Test" ), ( PyObject* )
&python::Test::GetType() ) == -1 )
 {
  return;
 }
}
```

I spent a couple of evenings trying to bring it back to life and understood that it is more complicated than I though.

Unless there is some kind of restriction the best way to go is Boost.Python.
