

Building Hybrid Systems with Boost.Python

Author: David Abrahams
Contact: dave@boost-consulting.com
Organization: Boost Consulting
Date: 2003-03-13
Author: Ralf W. Grosse-Kunstleve
Copyright

Abstract

Boost.Python is an open source C++ library which provides a concise IDL-like interface for

insulate C++ users from low-level Python 'C' API, replacing error-prone 'C' interfaces like manual reference-count management and raw

```
}  
}
```

Now here's the wrapping code we'd use to expose it with Boost.Python:

```
#include <boost/python.hpp>  
using namespace boost::python;  
BOOST_PYTHON_MODULE(hello)  
{
```

Exposing Classes

This does *not* result in adding attributes to the World instance `__dict__`

Inheritance

C++ inheritance relationships can be represented to Boost.Python by adding an optional `bases<...>` argument to the `class_<...>`


```
>>> call_s_f(Derived(), 'forty-two')
9
```

Things to notice about the dispatcher class:

- The key element which allows overriding in Python is the `call_method` invocation, which uses the same global type conversion registry as the C++ function wrapping does to convert its arguments from C++ to Python and its return type from Python to C++.
- Any constructor signatures you wish to wrap must be replicated with an initial `PyObject*` argument
- The dispatcher must store this argument so that it can be used to invoke `call_method`
- The `f_default` member function is needed when the function being exposed is not pure virtual; there's no other way `Base::f` can be called on an object of type `BaseWrap`

```
};
```

```
#include <boost/python.hpp>
```

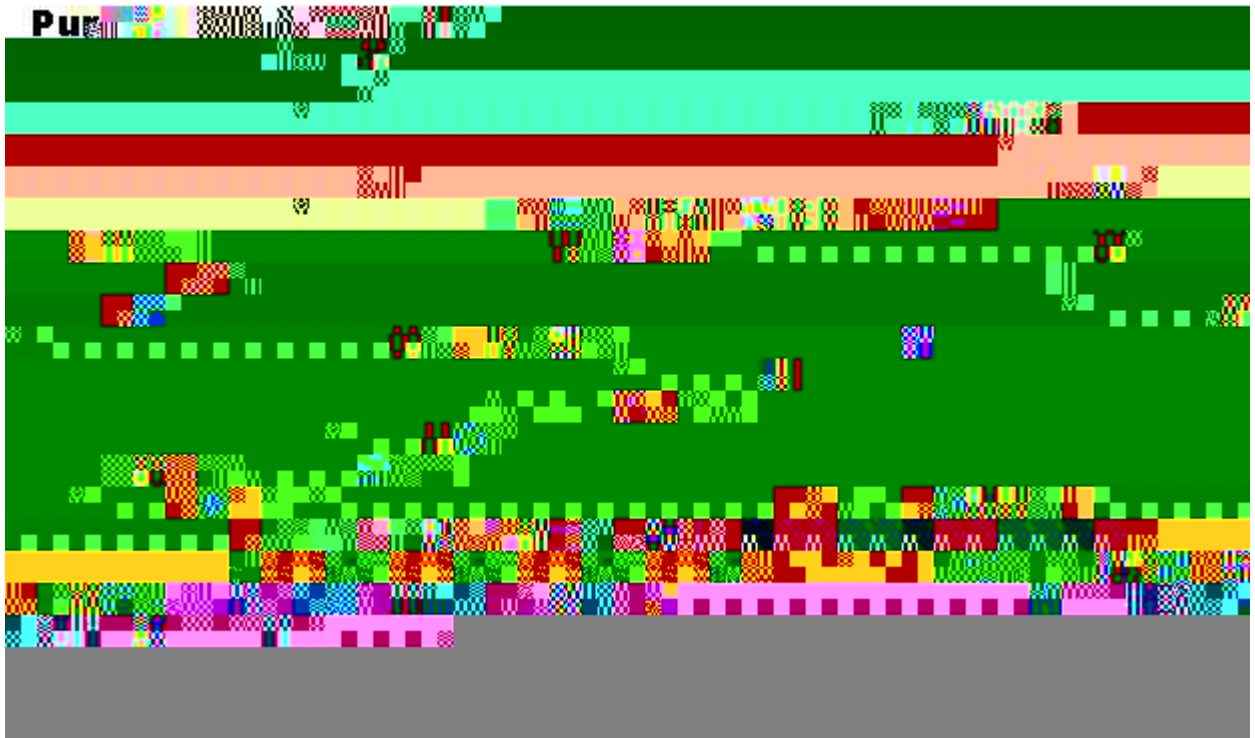
Boost.Python provides a class `object` which automates reference counting and provides conversion to Python from C++ objects of arbitrary type. This significantly reduces the learning effort for prospective extension module writers.

Creating an `object` from any other type is extremely simple:

```
object s('hello, world'); // s manages a Python string
```

`object` has templated interactions with all other types, with automatic to-python conversions. It happens so naturally that it's easily overlooked:

library of C++ classes with Boost.Python bindings, and for a while the growth was mainly concentrated on the C++ parts. However, as the toolbox is becoming more complete, more and more newly added functionality can be implemented in Python.



This figure shows the estimated ratio of newly added C++ and Python code over time as

By early 2001 development had stabilized and few new features were being added, however a disturbing new fact came to light: Ralf had begun testing Boost.Python on pre-release versions of a compiler using the **EDG** front-end, and the mechanism at the core of Boost.Python responsible for handling conversions between Python and C++ types was failing to compile.

[VELD1995] T. Veldhuizen, "Expression Templates," C++ Report, Vol. 7 No. 5 June 1995, pp. 26-31. <http://osl.iu.edu/~tveldhui/papers/Expression-Templates/exprtmpl.html>