PicklingTools: A Library for Connecting C++ and Python. PyCon 2010, Atlanta GA

Richard T. Saunders

Rincon Research Corporation and University of Arizona, Tucson AZ, USA

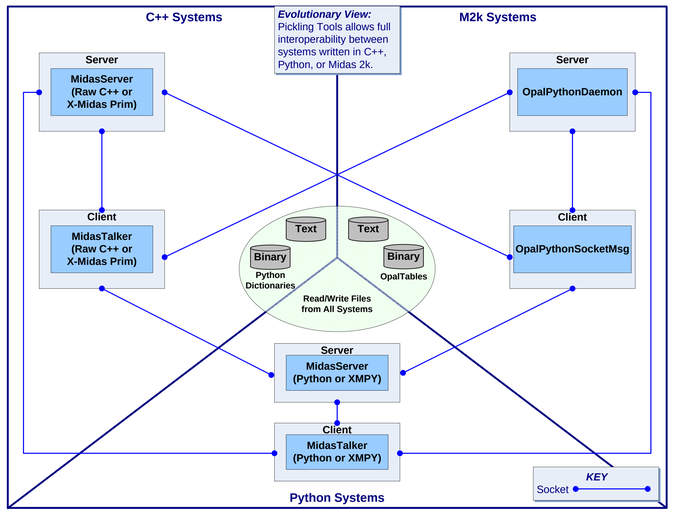
Abstract

The PicklingTools library is a open-source collection of C++ and Python code allowing C++ and Python processes (applications, subprograms, etc.) to communicate easily[Sau10a]. All communication centers on the *Python Dictionary*: it is the currency of the PicklingTools library.

```
{
  'REQUEST': {
    'type' : 'ping',
    'ports' : [8801, 8802],  # list of ports
    'timeout' : 5.5,         # in m.n seconds
    'error' : {
        'message' : 'no host',
        'retry' : 0
    }
  }
}
```

FIGURE 1: A Sample Python Dictionary: Within the PicklingTools, all exchanges between C++ and Python use the Python Dictionary

Python Dictionaries can be read from *files* or *sockets* in both C++ and Python. X-Midas and M2k are legacy systems using the PicklingTools[Sau10b].



The PicklingTools support multiple serializations of Python Dictionaries across sockets and files:

- Text Serialization: Human-readable format (like Figure 1). This format is easy to read and write and edit, but tends to be slow to serialize.
- Binary Serialization: Machine-readable format. This format is quick to serialize into files/sockets, but tends to be hard to read without special editors.

The term *PicklingTools* comes from the `pickle` module, where pickling is the nomenclature for (binary) serialization in Python. One of the PicklingTools greatest strengths is that it supports multiple serializations. In order from fastest to slowest:

- OpenContainers Binary: The native serialization of the C++ library OpenContainers. Tends to be 15 percent faster than the others.
- Midas 2k Binary: The native serialization of the Midas 2k framework.
- Python Pickling Protocol 2: The build-in serialization for Python. Works with Python 2.2 and above.
- Python Pickling Protocol 0: The built-in serialization for Python. Works with most versions of Python. It is 7-bit ASCII clean.
- Text: Textual Python Dictionaries (like Figure 1) or Textual OpalTables. Can be 10x slower than binary formats.

References

[Sau10a] Richard T. Saunders. The picklingtools web site. *http://www.picklingtools.com*, 2001-2010.

[Sau10b] Richard T. Saunders. Complex software systems in legacy and modern environments: A case study of the picklingtools library. *IEEE Hawaii International Conference on Systems Sciences*, 2010.

Python Libraries

All Python PicklingTools code is implemented with standard Python built-ins:

- Python Dictionaries: Built-in to the language
- Socket Libraries: Standard library: `import socket`
- Serialization Libraries: Standard library: `import cPickle`

There are no dependencies on external libraries (C or otherwise). Sample code for a client.

```
from * import midastalker   # Pure Python code
mt = MidasTalker('host', 9711)   # host, port for TCP/IP socket

request = { 'DATA': 'data to send' }
mt.send(request)

response = mt.recv(5.5)   # Block for 5.5 seconds if no response
print response   # some table
```

From Python, the choice of binary serializations is limited to Pickling Protocol 0 or 2: This limitation is to try to preserve the constraint of PURE Python code (no C module dependencies).

There has been extensive testing with Python 2.2 up to Python 2.6. Python 3.x should work, but hasn't been extensively tested: we only deliver products on RedHat Enterprise, which only comes with (until very recently) Python 2.x.

C++ Libraries

The C++ PicklingTools code has one major goal: When dealing with Python Dictionaries, make the C++ experience as pleasant as the Python experience. To accomplish this, we have to implement the Python abstractions in C++.

- Python Dictionaries: Not built-in to C++. PicklingTools uses the OpenContainers library to emulate the dynamic Python types. The OpenContainers comes with the PicklingTools distribution.
- Socket Libraries: Included on most UNIXes
- Serialization Libraries: Not built-in to C++. Implemented from scratch and comes with the PicklingTools distribution

We *do not* use the Boost library because it does *NOT* feel like Python. Sample code for a client. Note the similarity to the Python interface!

```
#include "midastalker.h"
MidasTalker mt("host", 9711); // host, port for TCP/IP socket

Tab request = "{ 'DATA': 'data to send' }";
mt.send(request);

Val response = mt.recv(5.5);  // Block for 5.5 seconds if no response
cout << response << endl;  // some table
```

The C++ portion has been tested thoroughly with the GNU C++ 3.X and 4.X series as well as the Intel 10.X Series. There are sample Makefiles for Linux and DEC OSF.

Dynamic Types in C++

The `Tab`, `Arr` and `Val` comprise the basic types to manipulate complex C++ dynamic data structures like Python dynamic data structures.

OpenContainers emulates the Python Dictionary with the `Tab` (think table) type.

```
Tab dict;  // Empty dictionary
Tab d1 = "{ 'a': 1, 'b': 2.2, 'c':'three' }";
d1["newkey"] = 3.14159265;
```

OpenContainers emulates Python List with the `Arr` (think array) type.

```
Arr lst;   // Empty list
Arr l1 = "[ 1, 2.2, 'three' ]";
l1.append(3.14159265);
```

C++ is a statically-typed language, so we simulate dynamic types by making a new type called `Val`.

```
Val a; // a can have ANY type!  Initialially None (like Python None type)
a = 6;               // a is an int
a = "some string";   // a is a string
a = 5.5;             // a is real-valued

// Use out-casting to get the value out appropriately
// a is current 5.5 ....
int outint = a;   // cast 5.5 to int as C would -> 5
string  s1 = a;   // stringize a -> '5.5'
real    r1 = a;   // r1 -> 5.5
```

For more documentation, see the web site at `http://www.picklingtools.com`