The Python Web services developer: Python SOAP libraries

Part 1

Level: Introductory

Mike Olson (mike.olson@fourthought.com), Principal Consultant, Fourthought, Inc.
Uche Ogbuji (uche.ogbuji@fourthought.com), Principal Consultant, Fourthought, Inc.

07 Nov 2001

In this first of a two-part series, Web services columnists Mike Olson and Uche Ogbuji discuss the various SOAP implementations available for Python, giving detailed code examples.

In the past 3 installments we have developed a Web services implementation using 4Suite Server, and taken advantage of the SOAP support of that product. (See Resources.) There have also been other SOAP implementations for Python; in fact, this seems to be quite a popular corner of open-source activity in Python. In this article, we shall take a look at Soap.py in action. For an update on the other open source SOAP projects, please see the sidebar. One immediate nit, though, is the naming of Python SOAP modules. It appears there hasn't been much talking between the different projects because there is a good deal of confusing similarity between the names. Recently, when explaining these choices to colleagues, we found ourselves at a loss to remember what was characteristic of SOAPy and what was characteristic of SOAP.py -- and that was after having spent a decent amount of time with both. This problem gets even worse, when one has to deal with module names within the actual libraries themselves, as we can see in the sidebar.

We covered 4Suite SOAP in the last three installments to this column; in this article and the next one, we shall present examples from the SOAP.py and SOAPy projects, which seem to have been the furthest along of this bunch at the time they became frozen. Note that although the W3C's XML protocol working group has produced a draft called SOAP 1.2, the common level of SOAP implementation across platforms and languages is still SOAP 1.1, with a strong representation of even earlier versions. The spread of SOAP versions these days causes complexity that might be greater than the simplicity promised by SOAP.

Clients and servers with SOAP.py

SOAP.py covers the basics. No Web Services Description Language (WSDL) or any other add-ons, but transparent support for implementing SOAP clients and servers in Python. Even the one nifty feature of the package relates to infrastructure: SOAP.py supports secure sockets layer (SSL), for encrypted SOAP transmissions. In order to use this feature you must have M2Crypto set up, which is a library covering a variety of cryptographic tools and formats, from RSA and DSA to HTTPs, S/MIME, and more. We shall not be examining the SSL support for SOAP.py in this installment.

Installation

Start by downloading the distribution (SOAPpy 0.9.7 was the latest at the time of writing), unpacking the files, changing to the resulting directory, and copying the file **SOAP.py** to an apt location. Of course this "apt" is the tricky bit. Because so many of these SOAP libs use the module name "soap.py" in some combination of case, one must be careful. Of course UNIX users need only worry if the case matches exactly, but Windows users can be bitten by a clash even between "SOAP.py" and "soap.py." Orchard's SOAP.py also has a clashing name, but it properly avoids any problems because its modules are sensibly tucked away under the Orchard package.

The short of all this is that we suggest ensuring that all your Python SOAP module installations use a differentiating package name. In our case, we found a suitable directory in our PYTHONPATH and created a WebServices package in which we placed SOAP.py.

SOAP opera digest
The story so far: SOAP utilities seem to be quite a popular corner of open-source activity in Python. Here is a rundown of the projects and their current status. First, the players:

- 4Suite SOAP, administered by Fourthought
- SOAPy, administered by Adam Elman
- SOAP.py, a project of the Web services for Python project
- soaplib, by Secret Labs
- Orchard, by Ken MacLeod
- PySOAP, administered by Dave

Therefore, in Linux:

```
$ mkdir ~/lib/python/WebServices
$ touch ~/lib/python/WebServices/__init__.py
$ cp SOAPpy097/SOAP.py ~/lib/python/WebServices
```

Note the important second command, which sets up the `__init__.py` file that marks the WebServices directory as a Python package. If you ever need to bundle this code to Windows, you might want to enter some comment into the empty file because some Windows tools refuse to create empty files.

You're soaking in it

There are already several active registries for publically-available SOAP servers. Probably the most popular is XMethods. Of course, it's also a pretty interesting guide to the state of SOAP reality, as opposed to its hype. Most of the public Web services out there are still trinkets, and hardly worth the noise our brave new model generates, but that is another story. As it is, we shall select a public service to demonstrate and test the use of SOAP.py as a SOAP client.

Or rather, we shall try to. The very first service the authors tried, a health care provider locator, showed up the pitfalls of the current state of SOAP interoperability when it choked with the following message:

```
WebServices.SOAP.faultType: <Fault soap:Client:
Server did not recognize the value of
HTTP Header SOAPAction: "".>
```

Uh oh. SOAPAction is an HTTP header which is supposed to signal the service being accessed. It is a mandatory header in a SOAP request, but even after setting the required header (just a pair of empty quotes), the above error persisted. The authors found this to be the case with most MS SOAP implementations. After trying out a bunch of these services, we determined that Delphi implementations seem to work best with SOAP.py, although when trying services -- even implemented in Delphi -- which returned complex types such as lists, SOAP.py had trouble with them, returning, say, a `WebServices.SOAP.typedArrayType` instance with no data.

Warner

4Suite SOAP is our own implementation, which we used in the last three installments of this column (see Resources for a link). It is under active development.

SOAPy was posted in April 2001, is currently in pre-alpha stages, and doesn't seem to be under active development.

SOAP.py development is frozen. SOAP.py as a project was being sponsored by a company called actzero, and actzero is no longer in business. New developers/maintainers are invited to volunteer.

soaplib' s development appears to have stalled, which is perhaps understandable given the huge body of work that Secret Labs undertakes these days. This Swedish company is led by Fredrik Lundh, famous in Python circles as the "eff-bot," and member of the Python Association board. Secret Labs also produces PythonWare, a kernel of Python and important add-on modules; PythonWorks, a leading Python IDE; the Python Imaging Library, and a host of other goodies (not least of which is the daily Python-URL Web log).

Orchard is a data management framework, basically a way to manage diverse data formats with a common interface. It implements a SOAP client as a basic way to send Orchard data items, known as Nodes, in a remote procedure call to a SOAP server.

PySOAP is a project intended as part of Dave Warner's Church-management suite, but has never released any files, and appears to be a dead project.

Appropriately enough, in the end, the authors chose a Web service that returns vintage curses by the Captain Haddock character in Tin Tin comics (yes, such are most Web services). Listing 1 (curse.py) is the program.

Listing 1: SOAP.py program to access Curse generator SOAP service

```
  #!/usr/bin/env python

#http://xmethods.net/detail.html?id=175
```

```
import sys

#Import the SOAP.py machinery
from WebServices import SOAP

remote = SOAP.SOAPProxy(
"http://www.tankebolaget.se/scripts/Haddock.exe/soap/IHaddock",
  namespace="urn:HaddockIntf-IHaddock",
  soapaction="urn:HaddockIntf-IHaddock#Curse"
)

try:
  lang = sys.argv[1]
except IndexError:
  lang = "us"

result = remote.Curse(LangCode=lang)

print "What captain Haddock had to say: "%s""%result
```

Putting it all together

After importing the library we set up the proxy object, `remote`. This object translates method invocations into remote SOAP messages. Its initializer takes the key parameters that govern remote requests: the URI of the server (known as the "endpoint"), the XML namespace of the request element (this is where SOAP-as-RPC gives lip service to its XML underpinnings), and the SOAPAction header value.

Next we determine the method argument, which in the case of this Web service is simply the language for Haddock's rantings, Swedish ("se") or English (strangely enough, "us" rather than "en").

Finally, we invoke the method of the right name, `Curse` on the proxy object to make the SOAP call, then we print the results. The following session illustrates the use of the program:

```
$ python curse.py
What captain Haddock had to say: "Ectoplasmic Byproduct!"
```

Our own SOAP server

Implementing a SOAP server in SOAP.py is quite easy. As an example, we shall emulate the field and also implement a trivial service: a program which, given a year and month, prints back a calendar as a string. The server program for this is Listing 2 (calendar-ws.py).

Listing 2: SOAP.py program to implement calendar server

```
  #!/usr/bin/env python

import sys, calendar

#Import the SOAP.py machinery
from WebServices import SOAP

CAL_NS = "http://uche.ogbuji.net/eg/ws/simple-cal"

class Calendar:
  def getMonth(self, year, month):
```

```
    return calendar.month(year, month)

  def getYear(self, year):
    return calendar.calendar(year)


server = SOAP.SOAPServer(("localhost", 8888))
cal = Calendar()
server.registerObject(cal, CAL_NS)

print "Starting server..."
server.serve_forever()
```

After the requisite imports, we define the namespace (`CAL_NS`) expected for SOAP request elements to our server. Next we define the class that implements all the methods which are to be exposed as SOAP methods. One can register individual functions as SOAP methods as well, but using the class approach is most flexible, especially if you wish to manage state between invocations. This `Calendar` class defines one method, `getMonth`, which uses Python's built-in calendar module to return a monthly calendar in text form, and another method which returns an entire year's calendar.

Then an instance of the SOAP server framework is created, with instructions to listen on port 8888. We must also create an instance of the `Calendar` class, which is registered to handle SOAP messages in the next line, with the relevant namespace indicated. Finally, we call the `serve_forever` methods, which doesn't return until the process is terminated.

In order to run the server, open up another command shell and execute `python calendar-ws.py`. Use ctrl-C to kill the process when you are done.

We could test the server with a client also written with SOAP.py, but that would be too obvious. Let us instead write a client in low-level Python to construct the SOAP response as an XML string and send an HTTP message. This program (testcal.py) is in Listing 3.

Listing 3: A client written using the Python core libraries to access the calendar service

```
  import sys, httplib

SERVER_ADDR = "127.0.0.1"
SERVER_PORT = 8888
CAL_NS = "http://uche.ogbuji.net/ws/eg/simple-cal"

BODY_TEMPLATE = """<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:s="http://uche.ogbuji.net/eg/ws/simple-cal"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <s:getMonth>
      <year xsi:type="xsd:integer">%s</year>
      <month xsi:type="xsd:integer">%s</month>
    </s:getMonth>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>"""


def GetMonth():
    year = 2001
    month = 12
    body = BODY_TEMPLATE%(year, month)
    blen = len(body)
    requestor = httplib.HTTP(SERVER_ADDR, SERVER_PORT)
    requestor.putrequest("POST", "cal-server")
    requestor.putheader("Host", SERVER_ADDR)
    requestor.putheader("Content-Type", "text/plain; charset="utf-8"")
```

```
    requestor.putheader("Content-Length", str(blen))
    requestor.putheader("SOAPAction", "http://uche.ogbuji.net/eg/ws/simple-car")
    requestor.endheaders()
    requestor.send(body)
    (status_code, message, reply_headers) = requestor.getreply()
    reply_body = requestor.getfile().read()

    print "status code:", status_code
    print "status message:", message
    print "HTTP reply body:\n", reply_body


if __name__ == "__main__":
    GetMonth()
```

The following session illustrates the running of this test.

```
$ python testcal.py
status code: 200
status message: OK
HTTP reply body:
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:SO
AP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<getMonthResponse SOAP-ENC:root="1">
<Result xsi:type="xsd:string">    December 2001
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
</Result>
</getMonthResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Bytes under scrutiny

One thing useful to note is that you can get details of the actual SOAP messages being exchanged and other key data for debugging and tracing, if you look for the line `self.debug = 0` and change the "0" to "1" (this is line 210 in SOAP.py version 0.9.7.) As an example, here is a session with the earlier curses.py program with debugging information turned on:

```
$ python curse.py
*** Outgoing HTTP headers *****************************************
POST /scripts/Haddock.exe/soap/IHaddock HTTP/1.0
Host: www.tankebolaget.se
User-agent: SOAP.py 0.9.7 (actzero.com)
Content-type: text/xml; charset="UTF-8"
Content-length: 523
SOAPAction: "urn:HaddockIntf-IHaddock#Curse"
************************************************************************
```

```
*** Outgoing SOAP ****************************************************
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:SO
AP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:Curse xmlns:ns1="urn:HaddockIntf-IHaddock" SOAP-ENC:root="1">
<LangCode xsi:type="xsd:string">us</LangCode>
</ns1:Curse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
**********************************************************************
*** Incoming HTTP headers *********************************************
HTTP/1.? 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 11 Sep 2001 16:40:19 GMT
Content-Type: text/xml
Content-Length: 528
Content:
**********************************************************************
*** Incoming SOAP *****************************************************
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xml
soap.org/soap/encoding/"><SOAP-ENV:Body><NS1:CurseResponse xmlns:NS1="urn:HaddockIntf-
IHaddock" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><NS1:return
xsi:type="xsd:string">Anacoluthons!</NS1:return></NS1:CurseRespon
se></SOAP-ENV:Body></SOAP-ENV:Envelope>
**********************************************************************
What captain Haddock had to say: "Anacoluthons!"
```

To compare, you would get this same information in a traditional Python script or program with the following code:

```
import calendar
return calendar.month(2001, 10)
```

SOAP.py concentrate

As we've noted, there are some hiccups with interoperability of SOAP.py, but hopefully the debugging data available is of help -- and among the developers who have signed up to keep this project moving along is Mike Olson, co-author of this column. In the next installment of this column we shall look at one of the other Python SOAP implementations.

Resources

- Participate in the discussion forum.
- XMethods: a SOAP service registry.

- The Daily Python-URL, edited by Fredrik Lundh of Secret Labs AB.
- M2Crypto: an cryptological library for Python.
- SOAPy: A SOAP/XML Schema Library for Python. See also the Source Forge project page for SOAPy.
- SOAP.py, a project of the Web services for Python project. See also the notice that development is frozen.
- PySOAP, intended as an implementation of the SOAP v1.1 standard in Python.
- soaplib, by Secret Labs.
- See the Orchard Sourceforge home page.

IBM resources

- IBM's Web services site has links to other resources.
- Check out two of the previous columns of the Python Web services developer:

About the authors

Mike Olson is a consultant and co-founder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management applications. Fourthought develops 4Suite, and 4Suite Server, open source platforms for XML middleware. You can contact Mr. Olson at mike.olson@fourthought.com.

Uche Ogbuji is a consultant and co-founder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management applications. Fourthought develops 4Suite, and 4Suite Server, open source platforms for XML middleware. Mr. Ogbuji is a Computer Engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can cont act Mr. Ogbuji at uche.ogbuji@fourthought.com.