How to create a Python class from a SOAP WSDL

Using Python and SOAP to attach to a Web Service

You can create a Python class from a SOAP WSDL in 4 steps, in about 20 minutes.

1. Create your Python Project
2. Download and install a SOAP library for Python (this how-to will use ZSI’s soap library)
3. Create a base Python class from a WSDL (either a file or a URL)
4. Tweak it for your purposes!

Step One – Create your Python Class

This might be the easiest step of all- if you’re just using Python to create an application, then you’re set! If you’re using something fancier, such as a web application framework, there might some extra steps involved.

For a simple, vanilla python example, just create yourself a directory (let’s call it soapclient) where you can write files and access python.exe (or just python in a unix/unix-like system), and navigate to that directory.

Step Two – Download and install a SOAP library for Python

Here, I’m listing the steps for the ZSI (Zolera Soap Infrastructure) libraries- it has more comprehensive support for WSDL's than the other Python SOAP libraries. If you're in a hurry, the quick way is to use something like EasyInstall from PEAK (Python Enterprise Application Kit). NOTE: Currently, EasyInstall provides the production version of ZSI, which is 2.0rc3. If you're using a modern version of Python, with libraries provided by either ActiveState or OS X, you'll have problems due to dependencies. ZSI 2.0rc3 is based on the PyXML libraries, which are no longer maintained and conflict with those provided by packaged Python releases. If you are using a pre-installed or packaged version of Python, you might need to use the beta, ZSI 2.1a1, which is based on minidom and compatible with packaged Python releases. Eventually, this should work itself out!

To install from the web using EasyInstall, follow the directions to install EasyInstall from the PEAK website, then just type "easy_install ZSI". If you're installing ZSI by hand, download the tar.gz, and do the "configure", "make", "make install".

Using EasyInstall to install the ZSI 2.1a1 release, you'll need to just download the egg from sourceforge for your version of Python, then type "easy_install ZSI-2.1_a1-py2.5.egg", using Python 2.5 as an example.

Step Three – Create a base Python class from a WSDL

Now, here's the fun part! In a unix/unix-ish system, ZSI will install a small script file called wsld2py in your /usr/local/bin directory, or a wsdl2py.exe in your ActiveState Scripts directory. Using that, type "wsdl2py ".

Let's use the Braille Text Service from xmethods as an example, to find the Braille equivalent of UCLA.

wsdl2py —complexType "http://www.webservicex.net/braille.asmx?WSDL"
(no semi-colon, though.. that's just a display issue in this entry)

Using ZSI 2.1, this will give you a couple of classes: ndfdXML_client.py, ndfdXML_server.py, and ndfdXML_types.py. Using ZSI 2.0, you'll get a ndfdXML_service.py and a ndfdXML_types.py file.

Step Four – Tweak it for your purposes

Now I’m going to focus on ZSI 2.1, because it’s easier. (sorry!)

From a command line, you can start python, then you’ll get the “>>>” prompt.

Here’s the code to get the Braille text for UCLA (you just need the parts after the “>>>”):

```
>>> from Braille_client import *
>>> mylocator = BrailleLocator()
>>> myport = mylocator.getBrailleSoap()
>>> myrequest = BrailleTextSoapIn()
>>> myrequest.InText = “UCLA”
>>> myrequest.TextFontSize = 24
>>> myresponse = myport.BrailleText(myrequest)
>>> f = open(‘uclabraille.jpg’, ‘w’)
>>> f.write(myresponse.BrailleTextResult)
>>> f.close()
>>> quit()
```

With any luck, you’ll get a file named uclabraille.jpg in your project directory, with an image of UCLA in braille. Voila! You’re done!

If you’re working with your own web services, you’ll need to find what the request parameters and return parameters are… the easiest method I’ve found so far is to use the python prompt, make your requests or responses, and type “print dir(myrequest)”, where myrequest is the name of your request object! Then it’s a little bit of trial and error.

Next steps: Next, you’d want to incorporate that in something like a compiled python app if you’re building a gui, or into a website if you’re building a python based website!

Revision #1
Created Thu, Nov 8, 2007 8:32 PM by Garcia, Ricardo
Updated Thu, Nov 8, 2007 8:32 PM by Garcia, Ricardo