

## Getting Started with Zephyr API for Python

## Revision History

Version	Type	Description/Comment	Date	Author
1.0	Creation	Initial release of the document.	6/24/11	Daniel Gannon

## Table of Contents

1 - Overview .....	4
2 - Required Software.....	5
2.1 Installing Python.....	5
2.2 Install setuptools (Optional) .....	6
2.3 Installing SUDS .....	6
3 - Using Zephyr API with Python.....	8
3.1 WSDL.....	8
3.2 IDLE IDE .....	8
3.2.1 Sample.py.....	9
3.2.2 Sample.py Output .....	11

## 1 - Overview

### How this guide will help

The “Getting Started with Zephyr API for Python” guide will show you how to use Zephyr API within your Python program, typically through the following steps:

- Importing the Zephyr WSDL in Python SUDS to expose the interfaces
- Writing scripts in Python that utilize the Zephyr API

This guide assumes you already have a working knowledge of Python. This guide will attempt to get you started by going over the software, in the “Required Software” section, which are necessary to perform the tasks detailed in the second half of the guide, “Using Zephyr API with Python”. Versions of required software are subject to change, including Zephyr. It is recommended that you try to use the most up to date version of software listed, to ensure best compatibility with your system and security.

At the end of the document, sample code and output is provided to help get you started. After reading and understanding this guide, you will be able to understand and perform the basic operations, available to you through Zephyr’s API structure.

## 2 - Required Software



### 2.1 Installing Python

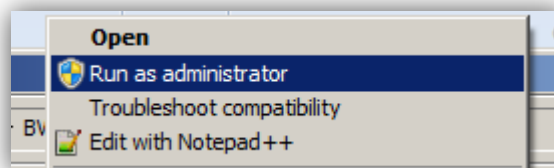
Before installing a newer version of Python, make sure to uninstall all other unnecessary previous installations.

Download from: <http://www.python.org/download/>

Steps:

#### 1. Run the Python Installer

Navigate to the directory where you downloaded the executable. You must have administrative rights in order to install on Windows. This can be done by right-clicking the icon and then clicking "Run As Administrator" from the menu.



#### 2. Update the PYTHON\_HOME variable (Optional)

You can run Python without setting the PYTHON\_HOME variable or you can set it for convenience. Setting the variables allows you to be able to conveniently run the executables from any directory without having to type the full command. Setting it this way will allow it to persist through reboot.

Setting up PYTHON\_HOME in Windows (steps vary with Windows version):

1. Click **Start > Control Panel > System**
2. Click **Advanced > Environment Variables**

3. Scroll through the fields until you find PYTHON\_HOME, if it's not present Add it to the System variables.
4. Add the location of the Python root folder to the value of PYTHON\_HOME in System variables (also User variables if present).

A typical path is: *C:\Python<version>*

#### Hints

- Only one variable is allowed
- Only applies to new command windows opened after committing

## 4. Installation Complete

You should now be ready to use Python on your machine.

### 2.2 Install setuptools (Optional)

Installing setuptools allows you to download, build, install, upgrade, and uninstalling Python packages easily.

Download from: <http://pypi.python.org/pypi/setuptools>

On Windows 32-bit systems, once download of the .exe file is complete, simply run the file to install setuptools. Python must be installed beforehand.

On Windows 64-bit systems, download *ez\_setup.py* and run it; it will download the appropriate .egg file and install it for you.

Additional installation help and further instructions can be found from the download site.

### 2.3 Installing SUDS

Suds is a lightweight SOAP python client for consuming Web Services. SUDS generates no classes, reads the wsdl at runtime, objectification of WSDL, and HTTP authentication among others.

There are two possible ways to install SUDS for Python that will be detailed.

First method is using setuptools detailed in the previous section (Recommended).

Open the command prompt:

### Windows 7/Vista

- Click on the start menu
- In the search bar, input: **CMD**
- Open CMD.EXE

### Windows XP/NT/2000

- Click on the start menu
- Select Run
- Input: **CMD**
- Open CMD.EXE

Once the command prompt is up, input these lines to install the latest version of SUDS.

```
cd C:\%PYTHON_HOME_DIRECTORY%\Scripts  
easy_install SUDS
```

```
C:\Python27\Scripts>easy_install SUDS
```

The other way to install SUDS is to do it manually.

Download from: <https://fedorahosted.org/suds/>

Download to a convenient location. Once downloaded, open the folder and look for a file named *setup.py*. Once found, go to the command prompt and type these lines:

```
cd C:\%DOWNLOAD_DIRECTORY%\<SUDS_VERSION>  
setup.py install
```

```
C:\python-suds-0.4>setup.py install
```

Now you have everything necessary to start working with API in Zephyr using Python.

## 3 - Using Zephyr API with Python

Reference URL of Zephyr API:

[http://support.yourzephyr.com/api\\_help/](http://support.yourzephyr.com/api_help/)

### 3.1 WSDL

The first step to using Zephyr API with Python is to know and understand the WSDL for Zephyr. Understanding WSDL requires a basic knowledge of XML because it is an XML-based language for describing Web services and how to access them. The WSDL for your Zephyr service can be viewed by going to its URL from any browser, while the service is running, just change the URL to include your server name.

The URL for the WSDL is:

<http://<your zephyr server name>/flex/services/soap/zephyrsoapservice-v1?wsdl>

What you should see:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="ZephyrSoapService" targetNamespace="http://getzephyr.com/com/thed/services/soap/zephyrsoapservice">
  <wsdl:import location="http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl=ZephyrSoapService.wsdl" namespace="http://soap.service.thed.com" /> </wsdl:import>
  <wsdl:binding name="ZephyrSoapServiceSoapBinding" type="ns1:ZephyrSoapService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  </wsdl:binding>
  <wsdl:operation name="addPhaseToCycle">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="addPhaseToCycle">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="addPhaseToCycleResponse">
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ZephyrServiceException">
      <soap:fault name="ZephyrServiceException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="getAttachmentsByCriteria">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="getAttachmentsByCriteria">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getAttachmentsByCriteriaResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:definitions>
```

### 3.2 IDLE IDE

IDLE is the Python IDE built with a GUI toolkit. IDLE features include letting you code python in a GUI environment, work with Windows or Unix, multi-window text editing, use interactive interpreter, and debugging. IDLE is included with Python from version 2.3 onward.

Open Python IDLE. Since IDLE IDE is made to be used for Python, there is no need for creating projects, such as with other popular IDEs like Eclipse.

Select **File> New** to create a blank page for your code.

You now have everything necessary to start creating code that interacts with Zephyr API!

Here is a sample program to help you get started! **Be sure to read the comments to best understand the code. Read after the sample for an expected output sample and help on how to run the code in IDLE IDE.**

### 3.2.1 Sample.py

```
# ////////////////////////////////////////////////////////////////////
# //
# // D SOFTWARE INCORPORATED
# // Copyright 2007-2011 D Software Incorporated
# // All Rights Reserved.
# //
# // NOTICE: D Software permits you to use, modify, and distribute this file
# // in accordance with the terms of the license agreement accompanying it.
# //
# // Unless required by applicable law or agreed to in writing, software
# // distributed under the License is distributed on an "AS IS" BASIS,
# // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# // implied.
# //
# ////////////////////////////////////////////////////////////////////
#
# This is a sample of what can be done by using API's with Zephyr through the Python
# coding language.
# By importing the Zephyr WSDL, you can use the methods without building classes.
#
# IDLE IDE - Version: 2.7.2
# Python - Version: 2.7.2
#
# Author: Daniel Gannon, Technical Support Analyst, D Software Inc.

# Imports logging and from external SUDS installation
import logging
from suds.client import Client
```

```

# Setup logging for file
logging.basicConfig(level=logging.INFO)

# Setup logging for the SOAP client to show soap messages (in & out) and http
headers:
# logging.getLogger('suds.client').setLevel(logging.DEBUG)

# URL to WSDL file on server and setup client variable
url = 'http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl'
client = Client(url)

# To get a list of methods provided by the Zephyr service, uncomment:
# print client

# Set username and password used to log in. Must have privileges to perform
operations below.
username = 'test.manager'
password = 'test.manager'

# Creates variable for holding remoteCriteria information
sC = client.factory.create('ns0:remoteCriteria')

# Name, Operation, and Value of search can all be modified. Here we left them blank
to return all records.
sC.searchName = ""
sC.searchOperation = ""
sC.searchValue = ""

# Logs into Zephyr using predefined credentials (username and password)
# prints generated session token for confirmation
session = client.service.login(username, password)
print 'Logged In.'
print 'The session token is: %s\n' % session

# Performs search operation and holds the findings in RP variable
# Variables passed to method are the searchCriteria from above, a flag for returning all
or some of the project information, and the session token for authentication
RP = client.service.getProjectsByCriteria(sC, 'true', session)

# Prints list length of RP
print 'Returned project(s): %s\n' % len(RP)

```

```

# Goes through all list entries and prints the name and startDate for each entry
for key in RP:
    print 'name: %s' % key.name
    print 'startDate: %s' % key.startDate

# Goes through all User list entries and prints the username and email for each
# A simple loop is needed to find the specific key for each user item
RU = client.service.getUsersByCriteria(sC, 'true', session)

print 'User List: %s\n' % len(RU)

for key in RU:
    print 'username: %s' % key.username
    print 'email: %s\n' % key.email

# Logs out of current session
client.service.logout(session)
print('\nLogged Out.')

```

### 3.2.2 Sample.py Output

```

Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART
=====
>>>
Logged In.
The session token is: bc6b4522-9bb9-482b-ae34-c7f44b57061b

Returned project(s): 3

Name: Sample Project
Start date: 2010-12-20 00:00:00

Name: Project One
Start date: 2011-04-30 23:00:00

Name: NewPro

```

Start date: 2011-03-31 23:00:00

User List: 6

username: any.one  
email: any.one@yourcompany.com

username: test.manager  
email: test.manager@yourcompany.com

username: test.lead  
email: test.lead@yourcompany.com

username: tester.one  
email: tester.one@yourcompany.com

username: tester.two  
email: none@none.com

username: tester.three  
email: none2@none.com

Logged Out.

>>>

To run the sample.py from the IDLE interface, select **Run> Run Module** or hit **F5** on your keyboard. This will open a new Python shell or update a current one, which will show the results of the code.