

Getting Started with Zephyr API for Perl

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 Padre on Strawberry	5
2.1.1 Alternatives.....	6
2.2 Acquiring SOAP::Lite	6
3 - Using Zephyr API with Perl.....	8
3.1 WSDL	8
3.2 SOAP::Lite	8
3.3 Padre IDE.....	9
3.3.1 Sample.pl.....	9
3.3.2 Sample.pl Output	14

1 - Overview

How this guide will help

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

- Utilizing SOAP::Lite to expose the Zephyr API interfaces
- Writing scripts in Perl that exercise the Zephyr API

This guide assumes you already have a working knowledge of Perl. 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 Perl”. 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 Padre on Strawberry

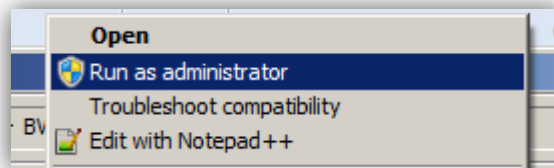
Before installing a newer version of Perl, make sure to uninstall all other unnecessary previous installations.

Download from: <http://padre.perlide.org/download.html>

Steps:

1. Run the Windows installer

Navigate to the directory where you downloaded the executable to. 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. Installation Complete

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

This Padre on Strawberry package comes with a few features. The first main feature is that the package includes Strawberry Perl. The second part is a Perl IDE called Padre. Since Perl has few IDEs available to it, this one will work with what we need. Plus, it is a good starting IDE when coding Perl. Look at the Padre website for more information about what features Padre on Strawberry includes.

2.1.1 Alternatives

An alternative to the Padre on Strawberry Perl package for Windows is ActivePerl. ActivePerl is the leading commercial-grade distribution of the open source Perl scripting language. Although it is primarily a commercial product, it does also provide a community edition of their software, supported by the ActivePerl community base.

Download from: <http://www.activestate.com/activeperl/downloads>

2.2 Acquiring SOAP::Lite

SOAP::Lite for Perl is a collection of Perl modules which provides a simple and lightweight interface to the Simple Object Access Protocol both on client and server side.

Download From: <http://www.soaplite.com/>

Save the folder into a convenient location. You must have installed Perl before SOAP::Lite can be installed.

Using Windows, 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 SOAP::Lite

```
cd C:\%SOAP::Lite_HOME_DIRECTORY\  
perl Makefile.PL  
  
nmake  
  
nmake test  
  
nmake install
```

Focus example:

```
C:\SOAP-Lite-0.712>perl Makefile.PL
```

Possible options for installation are:

--noprompt	Disable interactive dialog
---alltests	Perform extra testing
--help, -?	Display this help text

Hints

- Unix users will use the command 'make' instead of 'nmake'

You should now be able to use SOAP::Lite in your Perl applications.

For further instructions on installing SOAP::Lite, visit:

http://www.soaplite.com/2003/06/installation_in.html

3 - Using Zephyr API with Perl

Reference URL of Zephyr API:

http://support.yourzephyr.com/api_help/

3.1 WSDL

The first step to using Zephyr API with Perl 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: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:definitions>
```

3.2 SOAP::Lite

SOAP::Lite is meant to be used inside of Perl code but it has a useful utility called **stubmaker.pl**. This Perl file can take in a WSDL location and output a file of stubs for the WSDL file usable as a package in Perl. While this guide does not demonstrate using methods from this generated ZephyrSoapService.pm package, it is nonetheless a very convenient way of quickly starting up using Zephyr API services and could prove useful for your application.

Open the command prompt and type:

```
Cd C:\%SOAP::LITE_HOME_DIRECTORY%\bin
```

```
Perl stubmaker.pl http://<zephyr_server_name>/flex/services/soap/zephyrsoapservice-v1?wsdl
```

```
C:\SOAP-Lite-0.712\bin>perl stubmaker.pl http://localhost/flex/services/soap/ze[h  
yrsoapservice-v1?wsdl
```

This will generate the file in the bin folder of SOAP::Lite. It is recommended you move it to a more convenient location where Perl can automatically find it, such as the Perl /lib/ folder.

3.3 Padre IDE

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

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 Padre.**

3.3.1 Sample.pl

```
#!/usr/bin/perl -w
#
# ///////////////////////////////////////////////////////////////////
# //
# // 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.
```

```

# //
# ///////////////////////////////////////////////////////////////////

# For troubleshooting and debugging purposes you can use +trace => qw( debug )
# use SOAP::Lite +trace => qw( debug ) ;
use SOAP::Lite ;

# Convenient subroutine for printing all SOAP faults
sub print_fault {
    my ($result) = @_;

    if ($result->fault) {
        print "faultcode=" . $result->fault->{'faultcode'} . "\n";
        print "faultstring=" . $result->fault->{'faultstring'} . "\n";
        print "detail=" . $result->fault->{'detail'} . "\n";
    }
}

# Variable to hold URI of WSDL location
# Change localhost to the name of your Zephyr Server
my $url = 'http://localhost/flex/services/soap/zephyrsoapservice-v1';
# Variables to hold login parameters
my $user = 'test.manager';
my $pass = 'test.manager';

# Initilize soap variable object
my $soap = SOAP::Lite
-> uri('http://soap.service.thed.com/')
-> on_action( sub { join '/', 'http://soap.service.thed.com/', $_[1] } )
-> proxy($url);

# Initialized serializer
my $serializer = $soap->serializer();
$serializer->register_ns( 'http://soap.service.thed.com/', 'thed' );

# Initializes a method for logging in from WSDL's call: login
my $login = SOAP::Data->name('thed:login');

# Previously defined login params are saved in @params
my @params = ( SOAP::Data->name("username" => $user),
              SOAP::Data->name("password" => $pass) );

```

```

# Passes the parameters to the method and calls the method to be run
# Saves the result of the call in '$result'
my $result = $soap->call($login => @params);

# If statement to test if login was a success
# If the statement has a fault, it will 'pass' the If statement and print failure
# If the statement fails, it will print successful login and session token
if ($result->fault)
{
    print 'Login Failed, Try again: ';
    print_fault($result);
}
else
{
    print 'Logged In.';
    print "\n\n";
    print 'The session token is: ';
    print $result->result();
}

# Saves token into session variable
my $session = $result->result();

# Creates a getProjectsByCriteria method from WSDL's call: getProjectsByCriteria
my $getProjectsByCriteria = SOAP::Data->name('thed:getProjectsByCriteria');

# Creates and initializes variables for getProjectsByCriteria call
# searchCriteria is a value made up of: searchName, searchOperation, and
searchValue
# See WSDL documentation to see what these should be
# returnAllDataFlag is true, so all data in projects returned will be sent
# token is set to the session token created at login
@params = ( SOAP::Data->name("searchCriteria" =>
    \SOAP::Data->value(
        SOAP::Data->name("searchName" => SOAP::Data->value('id')),
        SOAP::Data->name("searchOperation" => SOAP::Data-
>value('EQUALS')),
        SOAP::Data->name("searchValue" => SOAP::Data->value('1'))
    )),
    SOAP::Data->name("returnAllDataFlag" => 'true'),
    SOAP::Data->name("token" => $session) );

```

```

$result = $soap->call($getProjectsByCriteria => @params);

if ($result->fault)
{
    print 'getProjectsByCriteria Failed, Try again: ';
    print_fault($result);
}
else
{
    # First array entry will be in result
    my @listings1 = $result->result();
    # Second and subsequent array entries will be saved in paramsout
    my @listings2 = $result->paramsout();

    # @listing1 is the array of structs
    foreach my $listing1 (@listings1) {
        print "\n-----\n";
        # print description for every listing
        foreach my $key (keys %{$listing1}) {
            print $key, ": ", $listing1->{$key} || ", "\n";
        }
    }
    # $listing2 is an array of structs
    foreach my $listing2 (@listings2) {
        print "\n-----\n";
        # print description for every listing
        foreach my $key (keys %{$listing2}) {
            print $key, ": ", $listing2->{$key} || ", "\n";
        }
    }
}

print "\n";

# Creates a getUsersByCriteria method from WSDL's call: getUsersByCriteria
my $getUsersByCriteria = SOAP::Data->name('thed:getUsersByCriteria');

# Creates and initializes variables for getUsersByCriteria call
@params = ( SOAP::Data->name("searchCriteria" =>
    \SOAP::Data->value(
        SOAP::Data->name("searchName" => SOAP::Data->value('id')),
        SOAP::Data->name("searchOperation" => SOAP::Data-

```

```

>value('EQUALS')),
    SOAP::Data->name("searchValue" => SOAP::Data->value('3'))
    ),
    SOAP::Data->name("returnAllDataFlag" => 'true'),
    SOAP::Data->name("token" => $session) );

$result = $soap->call($getUsersByCriteria => @params);

if ($result->fault)
{
    print 'getUsersByCriteria Failed, Try again: ';
    print_fault($result);
}
else
{
    my @listings1 = $result->result();
    my @listings2 = $result->paramsout();
    # @listings is the array of structs
    foreach my $listing1 (@listings1) {
        print "\n-----\n";
        # print description for every listing
        foreach my $key (keys %{$listing1}) {
            print $key, ": ", $listing1->{$key} || ", "\n";
        }
    }
    foreach my $listing2 (@listings2) {
        print "\n-----\n";
        # print description for every listing
        foreach my $key (keys %{$listing2}) {
            print $key, ": ", $listing2->{$key} || ", "\n";
        }
    }
}

print "\n";

# Creates a logout method from WSDL's call: logout
my $logout = SOAP::Data->name('thed:logout');

@params = ( SOAP::Data->name("token" => $session) );

$result = $soap->call($logout => @params);

```

```

if ($result->fault)
{
    print 'Logout Failed, Try again: ';
    print_fault($result);
}
else
{
    print 'Logged out.';
    print "\n\n";
    print $result->result();
}

print "\n"

```

3.3.2 Sample.pl Output

Logged In.

The session token is: da0365da-714c-4087-a1ed-c2b9018b60f9

members: ARRAY(0x2efeb8c)

newItem: true

status: 2

showItem: true

name: Sample Project

id: 1

startDate: 2010-12-20T00:00:00-08:00

description: This is a sample project shipped with Zephyr. Please modify this one or create a new one for your projects.

status: 2

showItem: false

name: Project One

description: Test

members: ARRAY(0x2efcd8c)

endDate: 2011-05-31T00:00:00-07:00

newItem: false

id: 2

startDate: 2011-05-01T00:00:00-07:00

status: 2
showItem: false
name: NewPro
description: project
members: ARRAY(0x2efc054)
endDate: 2011-06-30T00:00:00-07:00
newItem: false
id: 3
startDate: 2011-04-01T00:00:00-07:00

firstName: Anyone
credentialsExpired: true
location: Sunnyvale
username: any.one
email: any.one@yourcompany.com
accountEnabled: false
accountExpired: false
id: -10
lastName:
address1:

website: http://www.thedinc.com
loginName: test.manager
workPhoneNumber: +1-408-555-1212
state: CA
email: test.manager@yourcompany.com
city: Sunnyvale
id: 1
lastName: Manager
firstName: Test
country: US
credentialsExpired: false
location: Sunnyvale
username: test.manager
remoteRole: HASH(0x2ef6eec)
accountExpired: false

accountEnabled: true

address1:

title: Manager

type: 1

postalCode:

website: http://www.thedinc.com

loginName: test.lead

workPhoneNumber: +1-408-555-1212

state: CA

email: test.lead@yourcompany.com

city: Sunnyvale

id: 2

lastName: Lead

firstName: Test

country: US

credentialsExpired: false

location: Sunnyvale

username: test.lead

remoteRole: HASH(0x2ef66e4)

accountExpired: false

accountEnabled: true

address1:

title: Lead

type: 1

postalCode:

website:

loginName: tester.one

workPhoneNumber: +91-80-5551212

state:

email: tester.one@yourcompany.com

city: Bangalore

id: 3

lastName: One

firstName: Tester

country: India

credentialsExpired: false

location: Bangalore

username: tester.one

remoteRole: HASH(0x2ef5f04)
accountExpired: false
accountEnabled: true
address1:
title: Tester
type: 1
postalCode:

firstName: Tester
credentialsExpired: true
location: Fremont
loginName: tester.two
workPhoneNumber: 345-345-3456
username: tester.two
remoteRole: HASH(0x2e1b0ac)
email: none@none.com
accountExpired: false
accountEnabled: true
id: 5
title: Tester
lastName: Two
type: 4

firstName: Tester
credentialsExpired: true
location: Fremont
loginName: tester.three
workPhoneNumber: 124-123-1234
username: tester.three
remoteRole: HASH(0x2e1aaec)
email: none2@none.com
accountExpired: false
accountEnabled: true
id: 6
title: Tester
lastName: Three
type: 4

Logged out.

Press any key to continue . . .

To run the `sample.pl` from Padre interface, select the play button on the top toolbar. This will run the Perl code and show the output in a new window.