

Getting Started with Zephyr API for JAVA

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 Java.....	5
2.2 Acquiring Eclipse IDE.....	6
2.3 Acquiring Apache CXF.....	7
3 - Using Zephyr API with JAVA	8
3.1 WSDL	8
3.2 Apache CXF.....	9
3.2.1 Using the Command Line.....	9
3.3 Eclipse IDE.....	11
3.3.1 Sample.java	14
3.3.2 Sample.java – Expected Output.....	16

1 - Overview

How this guide will help

The “Getting Started with Zephyr API for Java” guide will show you how one goes about accessing Zephyr’s API interfaces through your Java application typically through the following steps:

1. Generating client side stubs using wsdl2java
2. Compiling those interfaces
3. Consuming them in your application.

This guide assumes you already have a working knowledge of Java. This guide will attempt to get you started by going over the required software, in “Required Software”, which are necessary to perform the tasks detailed in the second half of the guide, “Using Zephyr API with Java”. 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 Java

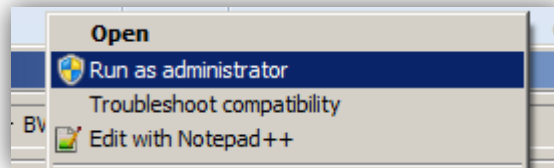
Before installing Java, make sure to uninstall all other unnecessary previous installations of Java.

Download from: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Steps:

1. Run the JDK 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. Running with Java Plug-in

You may need to close browser applications and some various programs in order to complete the installation process. You will need to also restart browser applications in order to enable Java for them.

3. Update the PATH and JAVA_HOME variables (Optional)

You can run Java without setting the PATH and JAVA_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 PATH 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 PATH
4. Add the location of the bin folder of the install for PATH in System variables (also User variables if present).

A typical path is: `C:\Program Files\Java\JDK_1.6.0_<version>\bin`

Setting up JAVA_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 JAVA_HOME, if it's not present Add it to the System variables.
4. Add the location of the JDK root folder to the value of JAVA_HOME in System variables (also User variables if present).

A typical path is: `C:\Program Files\Java\JDK_1.6.0_<version>`

Hints

- Variables are separated by a ";" are not case sensitive, and are read by Windows from left to right.
- Only one JDK variable is allowed
- Only applies to new command windows opened after committing

4. Installation Complete

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

2.2 Acquiring Eclipse IDE

Download from: <http://www.eclipse.org/>

Once downloaded, no installation is actually needed. Eclipse IDE can run immediately after you have finished downloading. It is recommended however, that you place the Eclipse folder in an easily accessible and secure place where it can't become compromised.

Upon first starting Eclipse IDE, you will be asked where you want to make your workspace. The workspace is another folder used by Eclipse to place all your projects, java files, and classes you will build. The workspace is not final. Once created, you can easily make a fresh new one (keeping the old one) or import a different one from another source.

2.3 Acquiring Apache CXF

Apache CXF is an open source services framework. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI.

You can acquire the Apache CXF BINARY DISTRIBUTION from:

<http://cxf.apache.org/index.html>

Apache CXF versions 2.3.4 and 2.4.0 are currently known to work with this information guide.

After downloading the binary zip file, unpack it and save it in a convenient location for later use.

3 – Using Zephyr API with JAVA

Reference URL of Zephyr API:

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

3.1 WSDL

The first step to using Zephyr API with JAVA 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.

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>
```

Zephyr's web service module is based on CXF from Apache, and you can use that to generate java stubs.

3.2 Apache CXF

Open the Apache CXF folder. Apache CXF has a number of different uses added into it by default but we are only interested in the WSDL to Java file.

This file can be located at:

```
%APACHE-CXF-<version>_HOME_DIRECTORY%\bin\wsdl2java.bat
```

3.2.1 Using the Command Line

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

There are many different ways to use the WSDL2]java.bat file in the command line:

-fe <front-end-name>
-db <data-binding-name>
-wv <wsdl-version>
-p <[wsdl-namespace =]package-name>*
-sn <service-name>
-b <binding-file-name>*
-reserveClass <class-name>*
-catalog <catalog-file-name>
-d <output-directory>
-compile
-classdir <compile-classes-directory>
-impl
-server
-client
-all
-autoNameReolutions
-allowElementReferences <=true>
-defaultValues <=class-name-for-DefaultValueProvider>
-ant
-nexclude <schema-namespace[=java-package-name]>*
Exsh <(true,false)>
-noTypes -dns <Default value is true>
-dex <(true,false)>

-validate
-keep
-wsdlLocation <wsdlLocation>
-xic<xic-arguments>*
-noAddressBinding
-useFQCNForFaultSerialUID
-mark
-generated
-h
-v
-verbose
-quiet
-wsdlList <wsdlurl>

The command line structure to generate the basic Java stubs:

```
%APACHE-CXF-<version>_HOME_DIRECTORY%\bin\wsdl2java.bat -impl -client -p  
<com.yourcompany.services> http://<zephyr_server_name>/flex/services/soap/zephyrsoapservice-  
v1?wsdl
```

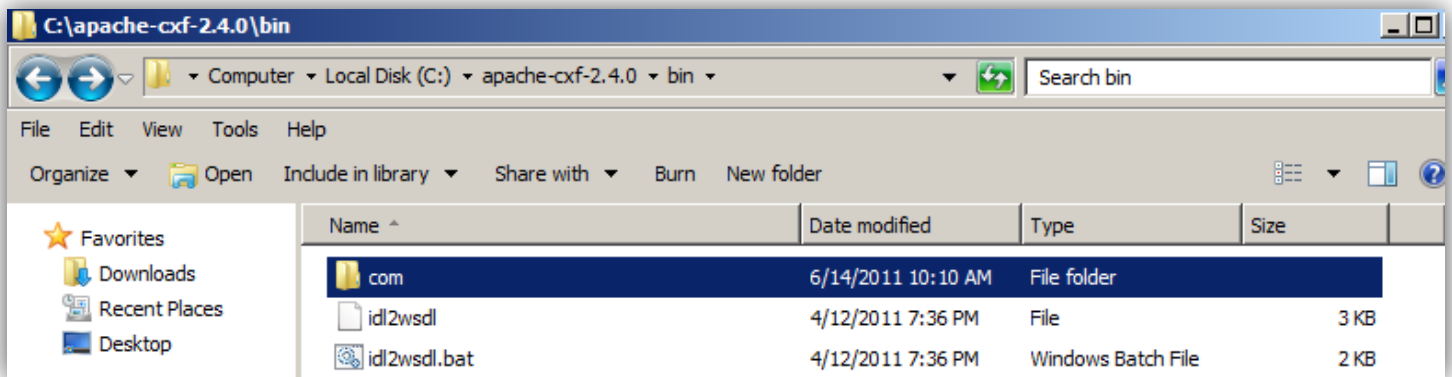
In this line, you can see com.yourcompany.services which is customizable. Ideally, you would want to put your own company name there and subfolders can be created by adding the '.' operator, IE: ".services"

IE, for stub generation from local installation:

```
%APACHE-CXF-<version>_HOME_DIRECTORY%\bin\wsdl2java.bat -impl -client -p  
com.yourcompany.services http://localhost:8080/flex/services/soap/zephyrsoapservice-v1?wsdl
```

```
C:\apache-cxf-2.4.0\bin>wsdl2java.bat -impl -client -p com.yourcompany.services  
http://localhost/fex/services/soap/zephyrsoapservice-v1?wsdl
```

A new folder, com, and subsequent subfolders will be created in:
`%APACHE-CXF-<version>_HOME_DIRECTORY%\bin\`



This folder contains all the subfolders and Java files generated from the WSDL2Java.bat file

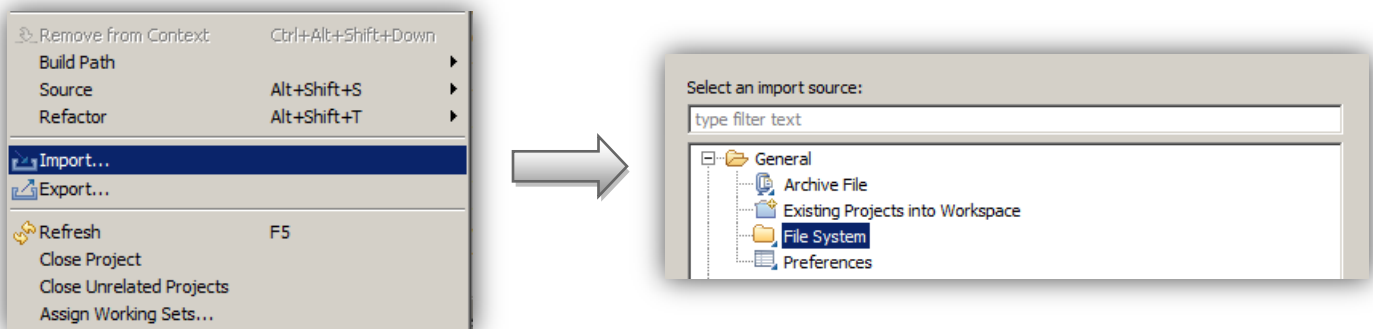
The com folder will be used in Eclipse, so you may, optionally, move the folder to a more convenient location.

3.3 Eclipse IDE

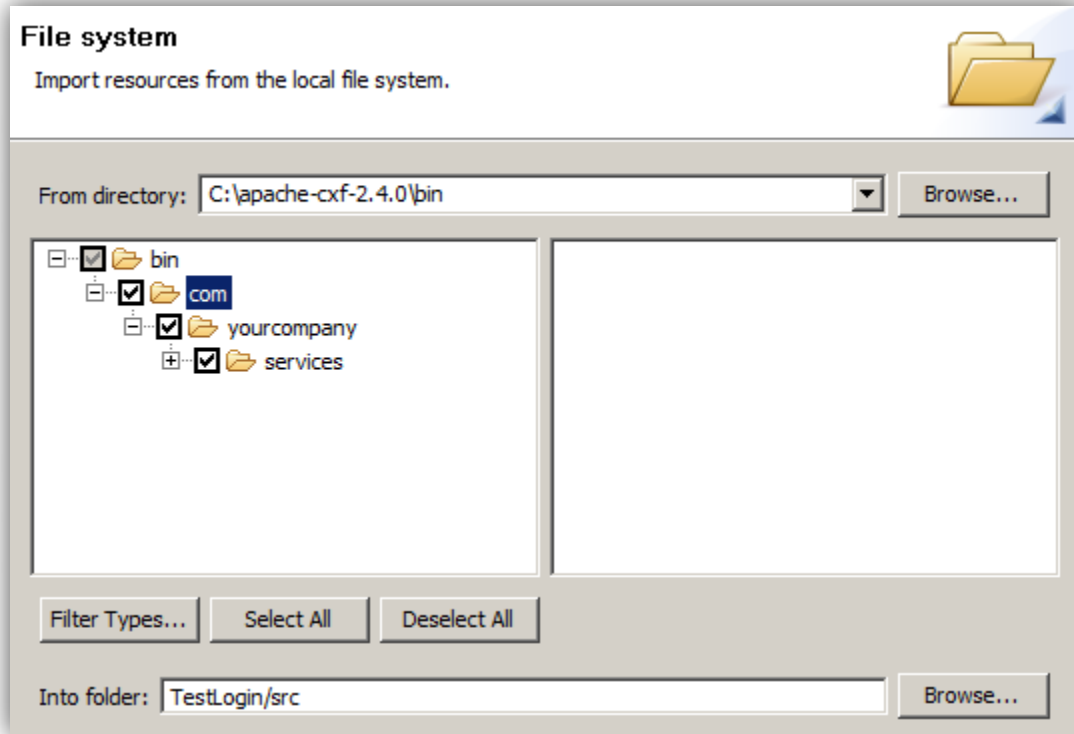
Open Eclipse IDE and assign a workspace, if necessary. You should now be on the welcome screen. From here you can go anywhere and do anything within Eclipse IDE.

Find the Package Explorer tab and right click inside of it.

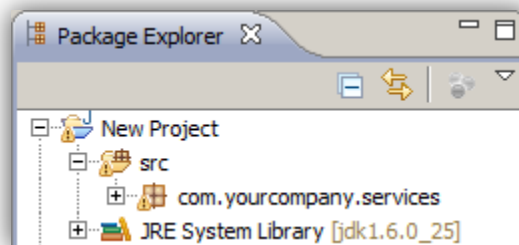
Select **New> Java Project**. This will bring up the New Java Project wizard. Give your projects a name and create it. Create a package in that project. Remember the com folder we created before? Now is the time to reference it. Right click your src folder in your Java project and click "Import". The import screen gives you many options to choose from but we want to import from the "File System" selection under the "General" tab.



Clicking next will bring you to a screen that asks for a directory. Manually input or browse to the com folder in your file system. If correct, the com folder should show up just underneath the directory. Click the check mark next to the com folder will add everything in the folder without needing to manually select each file.



In your src folder you should now see a new package:
com.<yourcompany>.services



Hint

You may need to comment out a few lines in the imported files to get them to stop showing error symbols in Eclipse IDE, if you don't have JAX-WS2.2.x installed. This is because some default constructors in the ZephyrSoapService_Service class file are optional for those with JAX-WS but not commented out by default.

In the package *com.<yourcompany>.services* the class file *ZephyrSoapService_Service* has three default constructors that can be commented out:

```
public ZephyrSoapService(WebServiceFeature ... features) {
    super(WSDL_LOCATION, SERVICE, features);
}

public ZephyrSoapService(URL wsdlLocation, WebServiceFeature ...
features) {
    super(wsdlLocation, SERVICE, features);
}

public ZephyrSoapService(URL wsdlLocation, QName serviceName,
WebServiceFeature ... features) {
    super(wsdlLocation, serviceName, features);
}
```

Additionally, in the package, in the *ZephyrSoapService_ZephyrSoapServiceImplPort_Client* class there, you will need to make two corrections:

```
Before:
public static void main(String args[]) throws Exception {

After:
public static void main(String args[]) throws MalformedURLException {
```

Create a new class by right-clicking the *src* folder and selecting **New> Class**. This will let you create a new package and class all at once. Give the package and class a name. Change any of the other settings as needed then click **Finish**.

You now have everything necessary to start creating programs that interact 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 Eclipse IDE.**

3.3.1 Sample.java

```
/*  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 JAVA coding language.
 * By creating the .java files from the Zephyr WSDL, you can import them
into your workspace and then
 * call them in your custom program.
 *
 * Eclipse IDE for Java Developers- Version: Helios Service Release 2.
Build id: 20110218-0911
 * Java- Java JDK 1.6.0_25
 *
 * Author: Daniel Gannon, Technical Support Analyst, D Software Inc.
 */

//Change the package to the package in your workspace
package New Project;

//com.yourcompany.service.* is imported to the package from outside
Eclipse, the package name will be whatever you put in the command line
run.
//The imported files are the result of the WSDL2JAVA.bat run
import com.yourcompany.services.*;
import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;

public class Sample {

    public static void main(String[] args) throws ZephyrServiceException
{

        //String URL variable that holds the location of your Zephyr
WSDL file
        //This example has http://localhost but yours may be different
```

```

String strURL =
"http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl";
//Name of user performing operation. Test.manager is a default
user name in a standard Zephyr installation
String username = "test.manager";
//Password for named user performing the operation. Test.manager
is a default password for the test.manager user name
String password = "test.manager";
//Variable for token created in login process. Initialized to
null value
String token = null;
//Creates project instance from RemoteProject object
RemoteProject project;
//Variable for searchCriteria collection list and initialization
java.util.List<RemoteCriteria> sC = null;
//Variable for RemoteProject collection list and initialization
java.util.List<RemoteProject> RP = null;

// This specifies the URL directly. The code will try to do the
operations and throw an exception if not
try {
    //Initializes the URL data type with strURL created
above
    URL url = new URL(strURL);
    //Create an instance of ZephyrSoapService. And initialize
it with namespaceUri and LocalPart
    ZephyrSoapService_Service serviceWithUrl = new
ZephyrSoapService_Service(url, new QName(
        "http://getzephyr.com/com/thed/services/soap/zephyrsoapservice",
        "ZephyrSoapService"));
    //servicePortWithUrl is used for API calls to retrieve and
add information in Zephyr
    ZephyrSoapService servicePortWithUrl =
serviceWithUrl.getZephyrSoapServiceImplPort();

    //Login to Zephyr and pass the returned token into the
token variable for later use
    //Simple IF/ELSE statement to confirm.
    token = servicePortWithUrl.login(username, password);
    if(token == null)
        System.out.println("Login Failed!");
    else
        System.out.println("Successfully Logged In. Your Token
is: " + token);

    //Uses the RP variable to save the list of project IDs.
Since sC is null and the return flag is true, it returns all projectIDs
    //Token created at login passed for process authentication
    RP = servicePortWithUrl.getProjectsByCriteria(sC , true,
token);

```

```

        /*
        * IF statement looks at the RP list collection, if it sees
nothing in the list it will skip the loop
        * Loop that uses the RemoteProject RP list to go through
all the projects
        * System.out.println for confirmation and example
        */
        if(RP.isEmpty() == false)
        {
            for(int i = 0; i < RP.size(); i++)
            {
                project = RP.get(i);
                System.out.println("\n" + "This name is: " +
project.getName());
                System.out.println("The project start date is: "
+ project.getStartDate());
            }
        }
        else
            System.out.println("\n No projects to display");

        //Call logout while passing sessions 'token'
        //System.out.println for confirmation
        servicePortWithUrl.logout(token);
        System.out.println("\n" + "Logged Out");

    } catch (MalformedURLException e) {
        e.printStackTrace(); //Prints exception(s)
    }
}
}

```

3.3.2 Sample.java - Expected Output

```

Successfully Logged In. Your Token is: fd21b89e-0830-4395-9f18-
d3015e897d61

```

```

This name is: Sample Project
The project start date is: 2010-12-20T00:00:00-08:00

```

```

This name is: Project One
The project start date is: 2011-05-01T00:00:00-07:00

```

```

This name is: NewPro
The project start date is: 2011-04-01T00:00:00-07:00


```

```

Logged Out

```

Please be aware that your project names and start dates (if you kept the sample intact) will likely be different from this sampled output. The idea behind giving the output is to allow you to look at the code and the output and connect what parts are doing what. Also, to check your output for comparison if need be.

With your code in place, be sure to save your work. After saving, make sure that no error symbols  in your code and then you can debug, compile, and run your code. With this example, debugging is not necessary but is a good practice to understand and use. Also, chances are, if you're using Eclipse IDE, your code is compiling automatically. A tell-tale sign of this is when you're writing code and error/ warning symbols appear without you clicking anything. This means that Eclipse has auto-compile enabled. Lastly, you will want to run your code to be sure that it performs expectedly and outputs correctly before possibly putting it into production. To run your code from the project explorer, **Select Class>Right-Click>Run-As>Java Application.**