Developer Walkthrough

WSDL SOAP Frameworks and CXF Overview

A Web Service Definition Language (WSDL) is an XML document that describes the functionality that is provided by a service. The WSDL includes all the operations that a service supports, and all the data types that are communicated through those operations. Cisco Hosted Collaboration Mediation Fulfillment provides WSDLs for all services that it supports.

There are many SOAP frameworks that help build SOAP clients and services. Some examples are Metro, Axis, and CXF. For all the examples in this documentation, it is assumed that the user is using CXF.

These frameworks, and CXF specifically, take the WSDL provided by Cisco Hosted Collaboration Mediation Fulfillment and generate code stubs that makes it easy to connect to Cisco Hosted Collaboration Mediation Fulfillment and configure it.

Apache CXF is an open-source services framework. CXF helps you build and develop services using front-end programming APIs, like JAX-WS and JAX-RS. These services have many protocols, such as SOAP, XML/HTTP, RESTful HTTP, or CORBA, and work over numerous transports, such as HTTP, JMS, or JBI.

You can download CXF here: http://cxf.apache.org

After you download CXF, you may need to set up your environment to use CXF. To do so, enter the following information at the command line: **export CXF_HOME=/Your_CXF_Download_Location/apache-cxf-2.X.X/**

Download WSDLs from Cisco HCM-F Platform

To download the WSDLs from the Cisco Hosted Collaboration Mediation Fulfillment platform, use the following example.

Procedure

Step 1 Create a directory for your project, and change into that directory.

```
In terminal:
$ pwd
/home/alice
$ mkdir myProject
$ cd myProject
```

Step 2 Create a directory for the WSDLs.

```
In terminal:
$ pwd
/home/alice/myProject
$ mkdir wsdl
$ cd wsdl
```

Step 3 Copy all WSDLs to the wsdl directory.

The WSDLs exist on the Cisco Hosted Collaboration Mediation Fulfillment platform. There is one WSDL for each service. You can retrieve them from the following locations:

Shared Data Repository Web Service

https://*HCM-F_Server_IPaddress*:8443/HCSWebServiceInterface/SharedDataRepositoryWebService?wsdl

FulFillment Web Service

https://*HCM-F_Server_IPaddress*:8443/HCSWebServiceInterface/FulfillmentWebService?wsdl

Service Inventory Web Service

https://*HCM-F_Server_IPaddress*:8443/HCSWebServiceInterface/ServiceInventoryWebService?wsdl

HCS License Manager Web Service API

https://*HCM-F_Server_IPaddress*:8443/HCSWebServiceInterface/HCSLicenseManagerWebService?wsdl

Step 4 Right-click the WSDL, select **Save Page as**, and save the WSDL in your development workspace. In this case, save the WSDLs in the `./wsdl/` folder.

Note When you save the WSDL file, name the saved file with a .wsdl suffix.

Use CXF to Autogenerate Code Stubs from WSDL

To generate autogenerated code stubs from the WSDL through CXF, use the following sample procedure.

Procedure

Step 1 Create a directory for the autogenerated source stubs.

```
In terminal:
$ pwd
/home/alice/myProject
$ mkdir autogen
```

```
$ mkdir autogen/src
$ mkdir autogen/build
```

Step 2 Generate autogenerated java code stubs using CXF.

Note This autogeneration step can take several minutes depending on the size of the WSDL and the processing power of your machine.

```
In terminal:
$ pwd
/home/alice/myProject
$ /Your_CXF_Download_Location/apache-cxf-2.X.X/bin/wsdl2java -classdir
autogen/build/
-d autogen/src/ -compile ./wsdl/SharedDataRepositoryWebService.wsdl
$ /Your_CXF_Download_Location/apache-cxf-2.X.X/bin/wsdl2java -classdir
autogen/build/
-d autogen/src/ -compile ./wsdl/HCSLicenseManagerWebService.wsdl
$ /Your_CXF_Download_Location/apache-cxf-2.X.X/bin/wsdl2java -classdir
autogen/build/
-d autogen/src/ -compile ./wsdl/ServiceInventoryWebService.wsdl
$ /Your_CXF_Download_Location/apache-cxf-2.X.X/bin/wsdl2java -classdir
autogen/build/
-d autogen/src/ -compile ./wsdl/FulfillmentWebService.wsdl
```

Step 3 If you hit java.lang.OutOfMemoryError while generating the code, you have to increase the maximum heap memory size of wsdl2java. This can be done as follows:

a) Open `/Your_CXF_Download_Location/apache-cxf-2.X.X/bin/wsdl2java` using your favorite editor.

b) Change:

```
$JAVA_HOME/bin/java -Xmx128M <SNIP>
```

to

```
$JAVA_HOME/bin/java -Xmx256M <SNIP>
```

Writing a Base HCS Connector Web Client Using the Autogenerated Code Stubs

To write a base web client by using the autogenerated code stubs, use the following sample procedure:

Procedure

Step 1 Create a directory for your business logic source.

```
In terminal:
$ pwd
/home/alice/myProject
$ mkdir src
```

Step 2 For the purposes of this example exercise, create an example package directory structure.

```
In terminal:
$ pwd
/home/alice/myProject
$ mkdir src/com
$ mkdir src/com/example
```

All source code that you write is stored in `/home/alice/myProject/src/com/example`.

Step 3 Create a directory for your compiled business logic class files.

```
In terminal:
$ pwd
/home/alice/myProject
$ mkdir build
```

Step 4 Write a web base client that helps you to connect to the Cisco Hosted Collaboration Mediation Fulfillment SOAP Interface (call this base web client the **HCSConnector**).

The HCSConnector takes care of setting up the remote host, credentials, logging, and certificate management.

See the sample java code file HCSConnector.java in the SampleCode.zip file. You can download the sample code and store it as /home/alice/myProject/src/com/example/HCSConnector.java.

Note A client requesting HCMF SOAP API must send these headers against PUT, POST, DELETE HTTP methods for security.

Mandatory headers:

a. **Origin** or **Referer** (Specify any one):

(Value ***https://" <hostname of target> [":" <port>]***

b. **X-Requested-With**:

(Value ***XMLHttpRequest***)

```
/** Utility Method to add Http Mime Headers to Soap Request
     *
     * @param soapWebservicePortType : The Soap-Port Stub Object retrieved from
soapWebServiceObject#getPort(..)
     * @param wsUrl : webservice URL
     */
    protected void addHttpMandatoryHeaders(Object o, String wsUrl) {
        try {


            org.apache.cxf.endpoint.Client client =
org.apache.cxf.frontend.ClientProxy.getClient(o);
            Map<String, Object> reqContextHeaders = client.getRequestContext();
            Map standardHeaders = (Map)
reqContextHeaders.get(org.apache.cxf.message.Message.PROTOCOL_HEADERS);
                                    // Atual object type is Map<String,List<String>>


            if (standardHeaders != null) {
               System.out.println("#addHttpMandatoryHeaders standardHeaders is not
 null");
                standardHeaders.put("Referer", Arrays.asList(wsUrl));
                standardHeaders.put("X-Requested-With",
Arrays.asList("XMLHttpRequest"));
            } else {
                System.out.println("#addMandatoryHeaders standardHeaders is null,
now adding ");
                Map<String, List<String>> cxfheaders = new HashMap<String,
List<String>>();
                cxfheaders.put("Referer", Arrays.asList(wsUrl));
              cxfheaders.put("X-Requested-With", Arrays.asList("XMLHttpRequest"));

                reqContextHeaders.put(Message.PROTOCOL_HEADERS, cxfheaders);
            }
        } catch (Exception e) {
            System.out.println("exception while adding Http Headers " + e);
```

To add the addHttpMandatoryHeaders() method to HCSConnector, use it in HCSConnectorImpl as shown in the example.

Step 5 Extend the HCSConnector base web client and implement a version-specific Cisco Hosted Collaboration Mediation Fulfillment Connector.

The HCSConnector uses APIs to obtain the remote service handle for each of the web services. This class uses the settings from the base class (HCSConnector) and CXF autogenerated code stubs to connect to the Cisco Hosted Collaboration Mediation Fulfillment SOAP Interface.

See the sample java code file HCSConnectorImpl.java in the SampleCode.zip file. You can download the sample code and store it as `/home/alice/myProject/src/com/example/HCSConnectorImpl.java`.

Use the addHttpMandatoryHeaders method in the following manner:

```
public FulfillmentWebService getFulfillmentWebService (String fulfillmentWebServiceUrl) {
        FulfillmentWebService_Service fulfillmentWebService_ss = null;
        FulfillmentWebService fulfillmentWebService = null;
        URL fulfillmentWSDLUrl = null;

        /*
         * Set up and return a Fulfillment Web Client handle
         */
        logger.info("Using URL: " + fulfillmentWebServiceUrl + " for HCS Fulfillment Web
Service");
        fulfillmentWSDLUrl =
getClass().getClassLoader().getResource("v9_1/FulfillmentWebService.wsdl");
        logger.info("Using WSDL URL: " + fulfillmentWSDLUrl);
        if (fulfillmentWSDLUrl == null) {
            logger.error("Could not get WSDL for Fulfillment Web Service");
            return null;
        }

       fulfillmentWebService_ss = new FulfillmentWebService_Service(fulfillmentWSDLUrl);

fulfillmentWebService_ss.addPort(FulfillmentWebService_Service.FulfillmentWebServiceImplPort,

                                        javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING,
                                        fulfillmentWebServiceUrl);
        fulfillmentWebService =
fulfillmentWebService_ss.getPort(FulfillmentWebService_Service.FulfillmentWebServiceImplPort,

                                                                FulfillmentWebService.class);

        addHttpMandatoryHeaders(fulfillmentWebService,fulfillmentWebServiceUrl);
        addAuthentication(fulfillmentWebService);
        addTrustAllSecurity(fulfillmentWebService);
        addLoggingInterceptors(fulfillmentWebService);
        enableMaintainSessions(fulfillmentWebService);
        return fulfillmentWebService;
}
```

Step 6 Write a web client that uses the version-specific HCSConnector web client and automates a sample workflow on Cisco Hosted Collaboration Mediation Fulfillment (call this web client the **WorkflowHCSWebClient**).

As part of the sample workflow, do the following:

a. Configure some default credentials

b. Configure a Unified Communications Domain Manager

Note Cisco HCM-F will deprecate the support of Cisco Unified Communications Domain Manager in the upcoming releases with limited support for existing integration, Cisco HCS partners and customers are advised to take necessary steps to align their requirements.

c. Configure a Unified Operations Manager

d. Configure a Data Center

- Configure a UCSManager
- Configure a vCenter

e. Synchronize all service providers

See the sample java code file WorkflowHCSWebClient.java in the SampleCode.zip file. You can download the sample code and store it as `/home/alice/myProject/src/com/example/WorkflowHCSWebClient.java`.

Step 7 Build the workflow web client.

This process involves writing a build.xml file that performs the following tasks:

- Compile the autogenerated code stubs.
- Archive all the autogenerated code and class files into a jar file (call this jar file "HCSWebClientGeneratedClasses.jar").
- Compile the business logic class files.
- Archive the business logic code and the class files into a jar file. (This jar file depends on the HCSWebClientGeneratedClasses.jar, and calls WorkflowHCSWebClient.jar.)
- Archive the downloaded WSDL files.
- Use the main class of WorkflowHCSWebClient as the class that is executed when the jar is run.

See the sample java code file build.xml in the SampleCode.zip file that does all the above. You can download the sample code and store it as /home/alice/myProject/build.xml.

The autogenerated code depends on CXF and the base web client and the workflow client depends on the autogenerated code, the build file has to find CXF. To do so, enter the following information:

```
export CXF_HOME=/Your_CXF_Download_Location/apache-cxf-2.X.X/
```

Build the workflow web client as follows:

```
In terminal:
    $ pwd
    /home/alice/myProject
    $ ant all
```

Step 8 Run the workflow web client as follows:

Example:

Note The client in this example uses the following system properties to connect to the Cisco HCS Web Server:

- hcs: Remote IP address or fully qualified domain name of the Cisco HCS Web Server
- Username: Username to use for authentication
- Password: Password to use for authentication

```
In terminal:
$ pwd
/home/alice/myProject
```

```
$ java -Dhcs="my-hcs-box" -Dusername="my_username"
-Dpassword="my_password" -jar ./jar/WorkflowHCSWebClient.jar
```