APPENDIX A

Use Case Examples

This chapter lists the use case sample code examples, and includes the following sections:

Use Case 1: SampleWSClient

The following example shows a java-based sample code to list the server farm or real servers on the ANM. After you import the ACE into the ANM, you can call the ANM Web Services API to get the list of ACE DeviceIDs and for each device ID, it displays the virtual context device IDs, and then for each context it displays the server farm and server farm real servers within the server farm:

```
/**
 * Title:       SampleWSClient
 * Description: Sample WS client code using java. It lists the devices imported in ANM,
the Virtual
 *             Contexts and the serverfarms/realservers configured in it.
 * Copyright:   Copyright(C) 2010 Cisco Systems. All Rights Reserved.
 * Company:     Cisco Systems
 * @author:     kamachan, Aug 2010
 * @version:    1.0
 */
package com.cisco.anm.client;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

import com.cisco.anm.DeviceID;
import com.cisco.anm.DeviceIDArray;
import com.cisco.anm.DeviceType;
import com.cisco.anm.OperationManager;
import com.cisco.anm.Serverfarm;
```

```
import com.cisco.anm.ServerfarmArray;
import com.cisco.anm.SessionToken;
import com.cisco.anm.SfRserver;
import com.cisco.anm.SfRserverArray;


/**
 * @author kamachan
 *
 *  Sample code to print all the serverfarms/realservers configured in all  imported ACE
modules and appliances.
 *  This code illustrate a basic usage of ANM WS code to get the ACE module/appliance
deviceid and its virtual context
 *  deviceid and then using the deviceid, it gets the list of serverfarms/realservers
configured in the deviceid.
 *
 *  Prerequiste:
 *  To compile the file, include the stubs generated from the wsdl file. It is already
packed in a jar (????) file - dcm-ws-client-<version>.jar ????
 *
 *
 */
public class SampleWSClient {
    static final String NAMESPACE = "http://anm.cisco.com";
    static String URL_TEMPLATE = "%s://%s:%s/anm/%s"; //
<protocol>://<IPAddress>:<Port>/anm/<service name>
    static final String WS_NAME = "OperationManager";

    public static BindingProvider getPort(String ws, String protocol, String host, String
port ) throws IOException {

        String url = String.format(URL_TEMPLATE, ""+ protocol,port, host, ws);
        String ns = NAMESPACE;

        BindingProvider bp;

        Service svc;
        try {
            QName qname = new QName(ns, ws + "Service");
            Class<Service> clazz = (Class<Service>) Class.forName("com.cisco.anm." + ws +
"Service");
            Constructor<Service> ctor = (Constructor<Service>) clazz.getConstructor(new
Class[]{URL.class, QName.class});
            svc =  ctor.newInstance(new Object[]{new URL(url+"?wsdl"), qname});
            Method m = svc.getClass().getMethod("get"+ws+"Port", null);
            bp = (BindingProvider)m.invoke(svc, null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return bp;
    }

    public static void main(String args[]){

    // args[0] = protocol , http or https
    // args[1] = IP Address or hostname of ANM server
    // args[2] = username,
    // args[3] = password.

    // java SampleWSClient https 10.77.241.54 admin admin
    if( args.length != 4 ) {
       System.out.println("Invalid arguments.");
       System.out.println("Usage: java com.cisco.anm.client.SampleWSClient
<protocol(http|https)> <ANM host name> <username> <password>");
```

```
        System.exit(0);
    }

    String proto=args[0], port="", host=args[1], user=args[2], pass=args[3] ;
    if( proto != null && proto.equals("https")){
        port = "8443";
    }else if( proto.equals("http")){
        port = "8080";
    }else{
        System.out.println("Invalid arguments.");
        System.out.println("Usage: java com.cisco.anm.client.SampleWSClient
<protocol(http|https)> <ANM host name> <username> <password>");
        System.exit(0);
    }
    OperationManager oprMgr = null;
    SessionToken st = null;
    try{
        oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

        /*
         * Invoke login() API to get session token.
         */
        st = oprMgr.login(user, pass);

        /*
         * Use the session token obtained from the previous call and get all ACE modules
         * imported in ANM.
         */
        DeviceIDArray deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.ACE_BLADE);
        List<DeviceID> listDeviceId = deviceIdArr.getItem();
            System.out.println("Device Type\t | Virtual Context Name\t | Module Slot\t |
Chassis IP address\t | Serverfarm\t | Real Server\t | Port\t | Real IP Address\t |
Weight\t | Admin State\t");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
        for( DeviceID dev : listDeviceId ){
            /*
             * Invoke listVirtualContext() to get all the virtual contexts for each ACE
module
             * imported in ANM.
             */
            DeviceIDArray vcIds = oprMgr.listVirtualContexts(st, dev);
            for( DeviceID vcId : vcIds.getItem() ){
                /*
                 * Invoke listServerFarms() API to get all the serverfarms configured for
each VC
                 *
                 */
                ServerfarmArray sfArr = oprMgr.listServerFarms(st, vcId);
                for(Serverfarm sf: sfArr.getItem() ){
                    /*
                     * Invoke listServerfarmRservers() API to get all the reals configured
in a serverfarm.
                     */
                    SfRserverArray sfRsArr =  oprMgr.listServerfarmRservers(st, vcId,
sf.getName());
                    for( SfRserver sfRs: sfRsArr.getItem() ){
                        System.out.println(vcId.getDeviceType() + "\t | " + vcId.getName()
+ "\t\t | "
                                           + vcId.getSlot() + "\t\t | " +
vcId.getChassisIPAddr() + "\t\t | "
```

```
                                        + sfRs.getServerfarmName() + "\t | " +
sfRs.getRealserverName() + "\t | "
                                        + sfRs.getPort() + "\t | " + sfRs.getIpAddr() + "\t
| " + sfRs.getWeight() + "\t | "
                                        + sfRs.getAdminState()
                                       );
                    }
                    if( sfRsArr.getItem().size() == 0) {
                        System.out.println(vcId.getDeviceType() + "\t | " + vcId.getName()
+ "\t\t | "
                                        + vcId.getSlot() + "\t\t | " +
vcId.getChassisIPAddr() + "\t\t | "
                                        + sf.getName() +  "\t\t "
                                          );
                    }
                }
            }
        }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");

        System.out.println("\n\n\n");
        /*
         * Use the session token obtained from the login() API call and get all ACE
modules
         * imported in ANM.
         */
        deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.ACE_4710);
        listDeviceId = deviceIdArr.getItem();
            System.out.println("Device Type\t | Virtual Context Name\t | Appliance IP
address\t | Serverfarm\t | Real Server\t | Port\t | Real IP Address\t | Weight\t | Admin
State\t");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
        for( DeviceID dev : listDeviceId ){
            /*
             * Invoke listVirtualContext() to get all the virtual contexts for each ACE
module
             * imported in ANM.
             */
            DeviceIDArray vcIds = oprMgr.listVirtualContexts(st, dev);
            for( DeviceID vcId : vcIds.getItem() ){
                /*
                 * Invoke listServerFarms() API to get all the serverfarms configured for
each VC
                 *
                 */
                ServerfarmArray sfArr = oprMgr.listServerFarms(st, vcId);
                for(Serverfarm sf: sfArr.getItem() ){
                    /*
                     * Invoke listServerfarmRservers() API to get all the reals configured
in a serverfarm.
                     */
                    SfRserverArray sfRsArr =  oprMgr.listServerfarmRservers(st, vcId,
sf.getName());
                    for( SfRserver sfRs: sfRsArr.getItem() ){
                        System.out.println(vcId.getDeviceType() + "\t | " + vcId.getName()
+ "\t\t | "
                                        + vcId.getIpAddr() + "\t\t | " +
sfRs.getServerfarmName() + "\t | "
```

```
                                        + sfRs.getRealserverName() + "\t | " +
sfRs.getPort() + "\t | "
                                        + sfRs.getIpAddr() + "\t | " + sfRs.getWeight() +
"\t | "
                                        + sfRs.getAdminState()
                                       );
                    }
                    if( sfRsArr.getItem().size() == 0) {
                        System.out.println(vcId.getDeviceType() + "\t | " + vcId.getName()
+ "\t\t | "
                                        + vcId.getIpAddr() + "\t\t | " + sf.getName() +
"\t\t "
                                          );
                    }
                }
            }
        }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");

    }catch(Exception ex){
        ex.printStackTrace();
    }finally{
        try{
            /*
             * Close the ws session using logout() API.
             */
            oprMgr.logout(st);
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }



    }
}
```

Use Case 2: ActivateSuspendSfRserverClient

The following example shows a sample code to activate or suspend a real server in the server farm:

```
/**
 * Title:       ActivateSuspendSfRserverClient
 * Description: Sample WS client code using java to activate or suspend a serverfarm real
configured in ACE/CSS/CSM device
 *
 * Copyright:   Copyright(C) 2010 Cisco Systems. All Rights Reserved.
 * Company:     Cisco Systems
 * @version:    1.0
 */

package com.cisco.anm.client;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.List;
```

```
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

import com.cisco.anm.DeviceID;
import com.cisco.anm.DeviceIDArray;
import com.cisco.anm.DeviceType;
import com.cisco.anm.OperationManager;
import com.cisco.anm.Serverfarm;
import com.cisco.anm.ServerfarmArray;
import com.cisco.anm.SessionToken;
import com.cisco.anm.SfRserver;
import com.cisco.anm.SfRserverArray;
import com.cisco.anm.SuspendState;


/**
 *
 *
 *  Sample WS client code using java to activate or suspend a serverfarm real configured
in ACE/CSS/CSM device.
 *
 *  Prerequiste:
 *  To compile the file, include the stubs generated from the wsdl file. It is already
packed in a jar file - dcm-ws-client-<version>.jar
 *
 *
 */
public class ActivateSuspendSfRserverClient {
    static final String NAMESPACE = "http://anm.cisco.com";
    static String URL_TEMPLATE = "%s://%s:%s/anm/%s"; //
<protocol>://<IPAddress>:<Port>/anm/<service name>
    static final String WS_NAME = "OperationManager";

    public static BindingProvider getPort(String ws, String protocol, String host, String
port ) throws IOException {

        String url = String.format(URL_TEMPLATE, ""+ protocol,port, host, ws);
        String ns = NAMESPACE;

        BindingProvider bp;

        Service svc;
        try {
            QName qname = new QName(ns, ws + "Service");
            Class<Service> clazz = (Class<Service>) Class.forName("com.cisco.anm." + ws +
"Service");
            Constructor<Service> ctor = (Constructor<Service>) clazz.getConstructor(new
Class[]{URL.class, QName.class});
            svc =  ctor.newInstance(new Object[]{new URL(url+"?wsdl"), qname});
            Method m = svc.getClass().getMethod("get"+ws+"Port", null);
            bp = (BindingProvider)m.invoke(svc, null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return bp;
    }

    public static void main(String args[]){

    // args[0] = protocol , http or https
    // args[1] = IP Address or hostname of ANM server
    // args[2] = username,
```

```
    // args[3] = password.
    // args[4] = (activate|suspend)
    // args[5] = sfName/rsName/rsPort/weight/rsIP
    // args[6] = (css|csm|vc)
    // args[7] = <vc name> if args[6] is VC
    // args[8] = <device IP Address>
    // args[9] = <chassis IP address>
    // args[10] = <slot>

     String usage = "For appliance vc, \n" +
                        "Usage: java com.cisco.anm.client.ActivateSuspendSfRserverClient
(http|https) <ANM host name> <anm username> <anm password>
(activate|suspend|graceful|suspend_clear_conn) <sfName/rsName/rsPort> vc <vc name>
<appliance ip address>\n" +
                        "Example: java com.cisco.anm.client.ActivateSuspendSfRserverClient
http 10.77.241.54 admin admin activate sjSF/sjRS/8080 vc Admin 10.77.241.46\n\n" +

                        "For module vc,\n" +
                        "Usage: java com.cisco.anm.client.ActivateSuspendSfRserverClient
(http|https) <ANM host name> <anm username> <anm password>
(activate|suspend|graceful|suspend_clear_conn) <sfName/rsName/rsPort> vc <vc name>
<chassis ip address> <module slot>\n" +
                        "Example: java com.cisco.anm.client.ActivateSuspendSfRserverClient
http 10.77.241.54 admin admin suspend sjSF/sjRS/8080 vc Admin 10.77.241.2 5\n\n" +

                        "For css,\n" +
                        "Usage: java com.cisco.anm.client.ActivateSuspendSfRserverClient
(http|https) <ANM host name> <anm username> <anm password>
(activate|suspend|graceful|suspend_clear_conn) <sfName/rsName/rsPort/weight> css <css ip
address>\n" +
                        "Example: java com.cisco.anm.client.ActivateSuspendSfRserverClient
https 10.77.241.54 admin admin suspend sjSF/sjRS/8080/5 css 10.77.241.100\n\n" +

                        "For csm,\n" +
                        "Usage: java com.cisco.anm.client.ActivateSuspendSfRserverClient
(http|https) <ANM host name> <anm username> <anm password>
(activate|suspend|graceful|suspend_clear_conn) <sfName/rsName/rsPort/rsIP> csm <chassis ip
address> <module slot>\n" +
                        "Example: java com.cisco.anm.client.ActivateSuspendSfRserverClient
https 10.77.241.54 admin admin activate sjSF/1.1.1.10/8080/1.1.1.10 csm 10.77.241.2 4\n"
                        ;
        if( args.length > 10 || args.length < 8 ) {
        System.out.println("Invalid arguments.");
        System.out.println(usage);
        System.exit(0);
    }

    String proto=args[0], port="", host=args[1], user=args[2], pass=args[3] ;
    String activatSuspend = args[4] , sfRsName = args[5], deviceType = args[6];

    if( proto != null && proto.equals("https")){
        port = "8443";
    }else if( proto.equals("http")){
        port = "8080";
    }else{
        System.out.println("Invalid arguments.");
        System.out.println(usage);
        System.exit(0);
    }
    String ipAddr="", chassisIP="", slot="", vcName="" ;
    DeviceID devId = new DeviceID();
    if( deviceType != null && deviceType.equals("css") ){ //css
        devId.setDeviceType(DeviceType.CSS);
        devId.setIpAddr(args[7]);
```

```
}else if( deviceType.equals("csm")){ // csm
    devId.setDeviceType(DeviceType.CSM);
    devId.setChassisIPAddr(args[7]);
    devId.setSlot(args[8]);
}else if( deviceType.equals("vc")) {
    if( args.length == 9 ){
        //appliance
        devId.setDeviceType(DeviceType.VIRTUAL_CONTEXT);
        devId.setIpAddr(args[8]);
        devId.setName(args[7]);
    }else if (args.length == 10) {
        //module
        devId.setDeviceType(DeviceType.VIRTUAL_CONTEXT);
        devId.setName(args[7]);
        devId.setChassisIPAddr(args[8]);
        devId.setSlot(args[9]);
    }else{
     System.out.println("Invalid arguments.");
     System.out.println(usage);
     System.exit(0);
    }
}

/*
 * Create SfRserver from the user's input.
 */
SfRserver sfRs = new SfRserver();
String[] rs = sfRsName.split("/");
if( rs.length == 3 ){
    sfRs.setServerfarmName(rs[0]);
    sfRs.setRealserverName(rs[1]);
    sfRs.setPort(Integer.parseInt(rs[2]));
}else if( rs.length == 4 ){
    sfRs.setServerfarmName(rs[0]);
    sfRs.setRealserverName(rs[1]);
    sfRs.setPort(Integer.parseInt(rs[2]));
    sfRs.setWeight(Integer.parseInt(rs[3]));
}else if( rs.length == 5 ){
    sfRs.setServerfarmName(rs[0]);
    sfRs.setRealserverName(rs[1]);
    sfRs.setPort(Integer.parseInt(rs[2]));
    sfRs.setWeight(Integer.parseInt(rs[3]));
    sfRs.setIpAddr(rs[4]);
}else{
    System.out.println("Invalid arguments.");
    System.out.println(usage);
    System.exit(0);
}
OperationManager oprMgr = null;
SessionToken st = null;
try{
    oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

    /*
     * Invoke login() API to get session token.
     */
    st = oprMgr.login(user, pass);

    if(activatSuspend.equals("activate")){
     System.out.println("Start: Activate realserver");
     /*
      * Invoke the activate api to activate the rserver in the given deviceId.
      */
```

```
        oprMgr.activateServerfarmRserver(st, devId, sfRs, "Activate Rserver from Java WS
Client.");

        System.out.println("Completed: Activate realserver");
        }else if(activatSuspend.equals("suspend")){

        System.out.println("Start: Suspend realserver");
        /*
         * Invoke the activate api to activate the rserver in the given deviceId.
         */
        oprMgr.suspendServerfarmRserver(st, devId, sfRs, SuspendState.SUSPEND, "Suspend
Rserver from JAVA WS Client.");

        System.out.println("Completed: Suspend realserver");
        }else if (activatSuspend.equals("graceful")){
        System.out.println("Start: Graceful suspend realserver");
        /*
         * Invoke the activate api to activate the rserver in the given deviceId.
         */
        oprMgr.suspendServerfarmRserver(st, devId, sfRs, SuspendState.GRACEFUL, "Graceful
Suspend Rserver from JAVA WS Client.");

        System.out.println("Completed: Graceful suspend realserver");
        }else if (activatSuspend.equals("suspend_clear_conn")){
        System.out.println("Start: Suspend with clear connection  realserver");
        /*
         * Invoke the activate api to activate the rserver in the given deviceId.
         */
        oprMgr.suspendServerfarmRserver(st, devId, sfRs,
SuspendState.SUSPEND_CLEAR_CONNECTIONS, "Suspend with Clear Connection Rserver from JAVA
WS Client.");

        System.out.println("Completed: Suspend with clear connection  realserver");
        }else{
            System.out.println("Invalid arguments.");
        System.out.println(usage);
        System.exit(0);
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }finally{
        try{
            /*
             * Close the ws session using logout() API.
             */
            oprMgr.logout(st);
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }



    }
}
```

Note As a prerequisite to compile the sample client code, you should have the client java file generated from the ANM WSDL files in the classpath. The ANM_web_services_API_client.jar file is located at the following location: http://developer.cisco.com/web/anm/docs.

Use Case 3: Virtual Server WS Client

The following example shows a sample code that works with the ANM APIs that are related to virtual servers:

```
/**
 * Title:       Virtual Server WS Client code
 * Description: Sample WS client code using java to work with Virtual server related APIs
in ANM
 *
 * Copyright:   Copyright(C) 2011 Cisco Systems. All Rights Reserved.
 * Company:     Cisco Systems
 * @version:    1.0
 */

package com.cisco.anm.client;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

import com.cisco.anm.DeviceID;
import com.cisco.anm.DeviceIDArray;
import com.cisco.anm.DeviceType;
import com.cisco.anm.OperationManager;
import com.cisco.anm.Serverfarm;
import com.cisco.anm.ServerfarmArray;
import com.cisco.anm.SessionToken;
import com.cisco.anm.SfRserver;
import com.cisco.anm.SfRserverArray;
import com.cisco.anm.SuspendState;
import com.cisco.anm.Vserver;
import com.cisco.anm.VserverArray;


/**
 *
 *
 *  Sample WS client code using java to work with Virtual server related APIs in ANM
 *
 *  Prerequiste:
 *  To compile the file, include the stubs generated from the wsdl file. It is already
packed in a jar file - dcm-ws-client-<version>.jar
 *
 *
 */
public class VserverWSClient {
    static final String NAMESPACE = "http://anm.cisco.com";
    static String URL_TEMPLATE = "%s://%s:%s/anm/%s"; //
<protocol>://<IPAddress>:<Port>/anm/<service name>
    static final String WS_NAME = "OperationManager";

    public static BindingProvider getPort(String ws, String protocol, String host, String
port ) throws IOException {

        String url = String.format(URL_TEMPLATE, ""+ protocol,port, host, ws);
        String ns = NAMESPACE;
```

```
        BindingProvider bp;

        Service svc;
        try {
            QName qname = new QName(ns, ws + "Service");
            Class<Service> clazz = (Class<Service>) Class.forName("com.cisco.anm." + ws +
"Service");
            Constructor<Service> ctor = (Constructor<Service>) clazz.getConstructor(new
Class[]{URL.class, QName.class});
            svc =  ctor.newInstance(new Object[]{new URL(url+"?wsdl"), qname});
            Method m = svc.getClass().getMethod("get"+ws+"Port", null);
            bp = (BindingProvider)m.invoke(svc, null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return bp;
    }

    public static void main(String args[]){

    // args[0] = protocol , http or https
    // args[1] = IP Address or hostname of ANM server
    // args[2] = username,
    // args[3] = password.
    // args[4] = (activate|suspend|listAllVirtualServer)
    // args[5] = virtualServerName/(policymap)
    // args[6] = (css|csm|vc)
    // args[7] = <vc name> if args[6] is VC
    // args[8] = <device IP Address>
    // args[9] = <chassis IP address>
    // args[10] = <slot>

     String usage = "For ACE4710 Appliance Virtual Context, \n" +
                        "Usage: java com.cisco.anm.client.VserverWSClient (http|https) <ANM
host name> <anm username> <anm password> (activate|suspend) <(virtualServerName/policymap)
| virtualServerName> vc <vc name> <appliance ip address>\n" +
                        "Example: java com.cisco.anm.client.VserverWSClient http
10.77.241.54 admin admin activate sjVS/global vc Admin 10.77.241.46\n\n" +

                        "For ACE Module Virtual Context,\n" +
                        "Usage: java com.cisco.anm.client.VserverWSClient (http|https) <ANM
host name> <anm username> <anm password> (activate|suspend) <(virtualServerName/policymap)
| virtualServerName> vc <vc name> <chassis ip address> <module slot>\n" +
                        "Example: java com.cisco.anm.client.VserverWSClient http
10.77.241.54 admin admin suspend sjVS/global vc Admin 10.77.241.2 5\n\n" +

                        "For CSS,\n" +
                        "Usage: java com.cisco.anm.client.VserverWSClient (http|https) <ANM
host name> <anm username> <anm password> (activate|suspend) <(virtualServerName/policymap)
| virtualServerName> css <css ip address>\n" +
                        "Example: java com.cisco.anm.client.VserverWSClient https
10.77.241.54 admin admin suspend sjVS/global css 10.77.241.100\n\n" +

                        "For CSM,\n" +
                        "Usage: java com.cisco.anm.client.VserverWSClient (http|https) <ANM
host name> <anm username> <anm password> (activate|suspend) <(virtualServerName/policymap)
| virtualServerName> csm <chassis ip address> <module slot>\n" +
                        "Example: java com.cisco.anm.client.VserverWSClient https
10.77.241.54 admin admin activate sjVS/global csm 10.77.241.2 4\n\n" +

                        "To list all Virtualservers in ANM ,\n" +
                        "Usage: java com.cisco.anm.client.VserverWSClient (http|https) <ANM
host name> <anm username> <anm password> listAllVirtualServer\n" +
```

```
                        "Example: java com.cisco.anm.client.VserverWSClient https
10.77.241.54 admin admin listAllVirtualServer\n\n";

     if( args.length > 10 || args.length < 8 ) {
           if( args.length != 5 ) {
           System.out.println("Invalid arguments.");
           System.out.println(usage);
           System.exit(0);
           }
    }

       if( args.length != 5 ) {
           String proto=args[0], port="", host=args[1], user=args[2], pass=args[3] ;
           String activatSuspend = args[4] , vsName = args[5], deviceType = args[6];

           if( proto != null && proto.equals("https")){
               port = "8443";
           }else if( proto.equals("http")){
               port = "8080";
           }else{
               System.out.println("Invalid arguments.");
               System.out.println(usage);
               System.exit(0);
           }
           DeviceID devId = new DeviceID();
           if( deviceType != null && deviceType.equals("css") ){ //css
               devId.setDeviceType(DeviceType.CSS);
               devId.setIpAddr(args[7]);
           }else if( deviceType.equals("csm")){ // csm
               devId.setDeviceType(DeviceType.CSM);
               devId.setChassisIPAddr(args[7]);
               devId.setSlot(args[8]);
           }else if( deviceType.equals("vc")) {
               if( args.length == 9 ){
                   //appliance
                   devId.setDeviceType(DeviceType.VIRTUAL_CONTEXT);
                   devId.setIpAddr(args[8]);
                   devId.setName(args[7]);
               }else if (args.length == 10) {
                   //module
                   devId.setDeviceType(DeviceType.VIRTUAL_CONTEXT);
                   devId.setName(args[7]);
                   devId.setChassisIPAddr(args[8]);
                   devId.setSlot(args[9]);
               }else{
                   System.out.println("Invalid arguments.");
                   System.out.println(usage);
                   System.exit(0);
               }
           }

           /*
            * Create Vserver from the user's input.
            */
           Vserver virtualServer = new Vserver();
           String[] vs = vsName.split("/");
           if( vs.length == 1 ){
               virtualServer.setVirtualserverName(vs[0]);
           }else if( vs.length == 2 ){
               virtualServer.setVirtualserverName(vs[0]);
               virtualServer.setPolicyMapName(vs[1]);
           }else{
               System.out.println("Invalid arguments.");
               System.out.println(usage);
```

```
                System.exit(0);
            }
            OperationManager oprMgr = null;
            SessionToken st = null;
            try{
                oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

                /*
                 * Invoke login() API to get session token.
                 */
                st = oprMgr.login(user, pass);

                if(activatSuspend.equals("activate")){
                    System.out.println("Start: Activate Virtualserver");
                    /*
                     * Invoke the activate api to activate the vserver in the given
deviceId.
                     */
                    oprMgr.activateVirtualServer(st, devId, virtualServer, "Activate
Vserver from Java WS Client.");
                    System.out.println("Completed: Activate Virtualserver");
                }else if(activatSuspend.equals("suspend")){

                    System.out.println("Start: Suspend Virtualserver");
                    /*
                     * Invoke the suspend api to suspend the vserver in the given deviceId.
                     */
                    oprMgr.suspendVirtualServer(st, devId, virtualServer, "Suspend Vserver
from JAVA WS Client.");
                    System.out.println("Completed: Suspend Virtualserver");
                }

                else{
                    System.out.println("Invalid arguments.");
                    System.out.println(usage);
                    System.exit(0);
                }
                oprMgr.listVirtualServers(st, devId, true);
            }catch(Exception ex){
                ex.printStackTrace();
            }finally{
                try{
                    /*
                     * Close the ws session using logout() API.
                     */
                    oprMgr.logout(st);
                }catch(Exception ex){
                    ex.printStackTrace();
                }
            }

        }
        else if(args.length == 5){
            String proto=args[0], port="", host=args[1], user=args[2], pass=args[3];
            OperationManager oprMgr = null;
            SessionToken st = null;
            if( proto != null && proto.equals("https")){
                port = "8443";
            }else if( proto.equals("http")){
                port = "8080";
            }else{
                System.out.println("Invalid arguments.");
                System.out.println(usage);
                System.exit(0);
```

```
            }
            try{
                oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

                /*
                 * Invoke login() API to get session token.
                 */
                st = oprMgr.login(user, pass);

                DeviceID devId = new DeviceID();
                DeviceIDArray deviceIdArr =  oprMgr.listDeviceIds(st,
DeviceType.ACE_BLADE);
                List<DeviceID> listDeviceId = deviceIdArr.getItem();
                System.out.println("ACE Module");
                System.out.println("Device Type\t | Virtual Context Name\t | Module Slot\t
| Chassis IP address\t | Virtualserver Name\t");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
                for( DeviceID dev : listDeviceId ){
                    /*
                 * Invoke listVirtualContext() to get all the virtual contexts for each ACE
module
                 * imported in ANM.
                 */
                   DeviceIDArray vcIds = oprMgr.listVirtualContexts(st,dev);
                   for( DeviceID vcId : vcIds.getItem() ){
                       /*
                        * Invoke listVirtualservers() API to get all the Virtual Servers
(recognized by ANM) configured for each VC
                        *
                        */
                      VserverArray vsArr = oprMgr.listVirtualServers(st, vcId, false);
                      for(Vserver vs: vsArr.getItem() ){
                          System.out.println(vcId.getDeviceType() + "\t | " +
vcId.getName() + "\t\t | "
                                  + vcId.getSlot() + "\t\t | " + vcId.getChassisIPAddr()
+ "\t\t | "
                                  + vs.getVirtualserverName());
                      }

                   }
                }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");

            System.out.println("\n\n\n");
            /*
             * Use the session token obtained from the login() API call and get all
ACE4710 Appliances
             * imported in ANM.
             */
                deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.ACE_4710);
                listDeviceId = deviceIdArr.getItem();
                System.out.println("ACE4710 Appliance");
                System.out.println("Device Type\t | Virtual Context Name\t | Appliance IP
address\t | Virtualserver Name\t ");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
```

```
                for( DeviceID dev : listDeviceId ){
                    /*
                     * Invoke listVirtualContext() to get all the virtual contexts for each ACE
module
                     * imported in ANM.
                     */
                       DeviceIDArray vcIds = oprMgr.listVirtualContexts(st,dev);
                       for( DeviceID vcId : vcIds.getItem() ){
                           /*
                            * Invoke listVirtualServers() API to get all the Virtual Servers
(recognized by ANM) configured for each VC
                            *
                            */
                              VserverArray vsArr = oprMgr.listVirtualServers(st, vcId, false);
                              for(Vserver vs: vsArr.getItem() ){
                                  System.out.println(vcId.getDeviceType() + "\t | " +
vcId.getName() + "\t\t | "
                                          + vcId.getIpAddr() + "\t\t | " +
vs.getVirtualserverName() + "\t | "
                                         );
                              }
                       }
                }

System.out.println("----------------------------------------------------------------------
----------------------------------------------------------------------------------------
--------------------");

            System.out.println("\n\n\n");
            /*
             * Use the session token obtained from the login() API call and get all CSM
modules
             * imported in ANM.
             */
                deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.CSM);
                listDeviceId = deviceIdArr.getItem();
                System.out.println("CSM");
                System.out.println("Device Type\t | Module Slot\t | Chassis IP address\t |
Virtualserver Name\t ");

System.out.println("----------------------------------------------------------------------
----------------------------------------------------------------------------------------
-----------------");
                for( DeviceID dev : listDeviceId ){
                    /*
                     * Invoke listVirtualServers() API to get all the Virtual
Servers(recognized by ANM) configured for each CSM
                     *
                     */
                       VserverArray vsArr = oprMgr.listVirtualServers(st, dev, false);
                           for(Vserver vs: vsArr.getItem() ){
                               System.out.println(dev.getDeviceType() + "\t | " +
dev.getSlot() + "\t\t | " + dev.getChassisIPAddr() + "\t\t | "
                                                  + vs.getVirtualserverName() + "\t | "
                                        );
                           }

                }

System.out.println("----------------------------------------------------------------------
----------------------------------------------------------------------------------------
--------------------");

            System.out.println("\n\n\n");
```

```
            /*
             * Use the session token obtained from the login() API call and get all CSS
devices.
             * imported in ANM.
             */
                deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.CSS);
                listDeviceId = deviceIdArr.getItem();
                System.out.println("CSS");
                System.out.println("Device Type\t | Device IP address\t | Virtualserver
Name\t ");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
                for( DeviceID dev : listDeviceId ){
                    /*
                     * Invoke listVirtualServers() API to get all the Virtual Servers
configured for each CSS device.
                     *
                     */
                        VserverArray vsArr = oprMgr.listVirtualServers(st, dev, false);
                            for(Vserver vs: vsArr.getItem() ){
                                System.out.println(dev.getDeviceType() + "\t | "
                                        + dev.getIpAddr() + "\t\t | " +
vs.getVirtualserverName() + "\t | "
                                        );
                            }

                }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");

                System.out.println("\n\n\n");
            }
                catch(Exception ex){
                    ex.printStackTrace();
                }finally{
                    try{
                        /*
                         * Close the ws session using logout() API.
                         */
                        oprMgr.logout(st);
                    }catch(Exception ex){
                        ex.printStackTrace();
                    }
                }
            }
            else {
                System.out.println("Invalid arguments.");
                System.out.println(usage);
                System.exit(0);
            }
        }
    }
```

Use Case 4: GSSWSClient

The following example shows a sample code to activate, suspend, and list the answers and DNS rules configured on a GSS:

```
/**
 * Title:      GSSWSClient
 * Description: Sample WS client code using java to activate/suspend/list the
answer/dnsrule configured in GSS device
 *
 * Copyright:   Copyright(C) 2011 Cisco Systems. All Rights Reserved.
 * Company:     Cisco Systems
 * @version:    1.0
 */

package com.cisco.anm.client;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

import com.cisco.anm.AnswerArray;
import com.cisco.anm.DeviceID;
import com.cisco.anm.DeviceIDArray;
import com.cisco.anm.DeviceType;
import com.cisco.anm.DnsRuleArray;
import com.cisco.anm.OperationManager;
import com.cisco.anm.Serverfarm;
import com.cisco.anm.ServerfarmArray;
import com.cisco.anm.SessionToken;
import com.cisco.anm.SfRserver;
import com.cisco.anm.Answer;
import com.cisco.anm.DnsRule;
import com.cisco.anm.SfRserverArray;
import com.cisco.anm.SuspendState;


/**
 *
 *
 *  Sample WS client code using java to activate/suspend/list the answer/dnsrule
configured in GSS device
 *
 *  Prerequiste:
 *  To compile the file, include the stubs generated from the wsdl file. It is already
packed in a jar file - dcm-ws-client-<version>.jar
 *
 *
 */
public class GSSWSClient {
    static final String NAMESPACE = "http://anm.cisco.com";
    static String URL_TEMPLATE = "%s://%s:%s/anm/%s"; //
<protocol>://<IPAddress>:<Port>/anm/<service name>
    static final String WS_NAME = "OperationManager";

    public static BindingProvider getPort(String ws, String protocol, String host, String
port ) throws IOException {
```

```
        String url = String.format(URL_TEMPLATE, ""+ protocol,port, host, ws);
        String ns = NAMESPACE;

        BindingProvider bp;

        Service svc;
        try {
            QName qname = new QName(ns, ws + "Service");
            Class<Service> clazz = (Class<Service>) Class.forName("com.cisco.anm." + ws +
"Service");
            Constructor<Service> ctor = (Constructor<Service>) clazz.getConstructor(new
Class[]{URL.class, QName.class});
            svc =  ctor.newInstance(new Object[]{new URL(url+"?wsdl"), qname});
            Method m = svc.getClass().getMethod("get"+ws+"Port", null);
            bp = (BindingProvider)m.invoke(svc, null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return bp;
    }

    public static void main(String args[]){

        // args[0] = protocol , http or https
        // args[1] = IP Address or hostname of ANM server
        // args[2] = username,
        // args[3] = password.
        // args[4] = (activate|suspend|list)
        // args[5] = (answer|dns)
        // args[6] = <device IP Address>
        // args[7] = <answerName/ansIpAdress> | <ruleName/sourceAddress>

        String usage = "For Answer activate/suspend operation , \n" +
        "Usage: java com.cisco.anm.client.GSSWSClient (http|https) <ANM host name> <anm
username> <anm password> (activate|suspend)  (answer) <GSS ip address>
<answerName/ansIpAdress> \n" +
        "Example: java com.cisco.anm.client.GSSWSClient http 10.77.241.54 admin admin
activate answer 10.77.241.46 VIP-CSCO-9-99/11.0.9.99 \n\n" +

        "To list all Answers, \n" +
        "Usage: java com.cisco.anm.client.GSSWSClient (http|https) <ANM host name> <anm
username> <anm password> (list)  (answer) \n" +
        "Example: java com.cisco.anm.client.GSSWSClient http 10.77.241.54 admin admin list
answer \n\n" +

        "For DNS rule activate/suspend operation , \n" +
        "Usage: java com.cisco.anm.client.GSSWSClient (http|https) <ANM host name> <anm
username> <anm password> (activate|suspend)  (dns) <GSS ip address> <ruleName> \n" +
        "Example: java com.cisco.anm.client.GSSWSClient http 10.77.241.54 admin admin
activate dns 10.77.241.46 RULE-CSCO-9-96 \n\n" +

        "To list all DNS rules, \n" +
        "Usage: java com.cisco.anm.client.GSSWSClient (http|https) <ANM host name> <anm
username> <anm password> (list) <dns> <GSS name> \n" +
        "Example: java com.cisco.anm.client.GSSWSClient http 10.77.241.54 admin admin list
dns \n\n"
        ;
        if(args.length < 8 )  {
            if( args.length != 6 ){
                System.out.println("Invalid arguments.");
                System.out.println(usage);
                System.exit(0);
            }
```

```
        }

        String proto=args[0], port="", host=args[1], user=args[2], pass=args[3] ;
        String opeartion = args[4] , type = args[5];

        if( proto != null && proto.equals("https")){
            port = "8443";
        }else if( proto.equals("http")){
            port = "8080";
        }else{
            System.out.println("Invalid arguments.");
            System.out.println(usage);
            System.exit(0);
        }

        OperationManager oprMgr = null;
        SessionToken st = null;
        String operData = null;
        String[] data = null;
        DeviceID GSS = null;
        if(args.length==8)
        {
            GSS = new DeviceID();
            GSS.setDeviceType(DeviceType.GSS);
            GSS.setSlot("0");
            GSS.setIpAddr(args[6]);
            operData = args[7];
            data = operData.split("/");
        }

        if(type.equalsIgnoreCase("Answer"))
        {
            Answer answer = new Answer();
            if(!opeartion.equalsIgnoreCase("list")){
                if( data.length == 2 ){
                    answer.setAnswerName(data[0]);
                    answer.setIpAddr(data[1]);
                    answer.setType("vip");
                }
                else{
                    System.out.println("Invalid arguments.");
                    System.out.println(usage);
                    System.exit(0);
                }
            }
            try{
                oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

                /*
                 * Invoke login() API to get session token.
                 */
                st = oprMgr.login(user, pass);

                if(opeartion.equalsIgnoreCase("activate")){
                    System.out.println("Start: Activate Answer");
                    /*
                     * Invoke the activate api to activate the answer in the given
deviceId.
                     */
                    oprMgr.activateAnswer(st, GSS, answer, "Activate Answer from Java WS
Client.");

                    System.out.println("Completed: Activate answer");
                }else if(opeartion.equalsIgnoreCase("suspend")){
```

```
                    System.out.println("Start: Suspend answer");
                    /*
                     * Invoke the activate api to suspend the answer in the given deviceId.
                     */
                    oprMgr.suspendAnswer(st, GSS, answer, "Suspend Answer from Java WS
Client.");

                    System.out.println("Completed: Suspend answer");
                }else if (opeartion.equalsIgnoreCase("list")){
                    System.out.println("Start: List Answers");
                    DeviceIDArray deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.GSS);
                    List listDeviceId = deviceIdArr.getItem();
                    for( DeviceID gss: deviceIdArr.getItem() ){
                        /*
                         * Invoke the list api to get the  list of answers in the given
deviceId.
                         */
                        AnswerArray answerArray = oprMgr.listAnswers(st, gss);
                        int list_size = 0;
                        if (answerArray.getItem()!= null &&
!answerArray.getItem().isEmpty())
                            list_size = answerArray.getItem().size();

                        System.out.println("GSS Name\t | Answer Name\t | Answer IP
Address\t | Config State \t | PGSSM Oper State \t | Answer Group \t | Location \t");

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-----------------");
                        for( Answer ans: answerArray.getItem() ){
                            System.out.println(gss.getName() + "\t | " +
ans.getAnswerName() + "\t | " + ans.getIpAddr() + "\t\t | "
                                    + ans.getConfigState() + "\t\t | " + ans.getOperState()
+ "\t\t | " + ans.getAnswerGroups() + "\t\t | "
                                    + ans.getLocation() +  "\t\t ");
                        }
                        if( answerArray.getItem().size() == 0) {
                            System.out.println("No answers is configured in the given GSS
device" );
                        }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");

                    }
                    if( deviceIdArr.getItem().size() == 0) {
                        System.out.println("No GSS device is imported in this ANM server"
);
                    }

System.out.println("-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
--------------------");
                    System.out.println("Completed: List Answers");
                }
            }
            catch(Exception ex){
                ex.printStackTrace();
            }finally{
                try{
                    /*
                     * Close the ws session using logout() API.
```

```
                     */
                    oprMgr.logout(st);
                }catch(Exception ex){
                    ex.printStackTrace();
                }
            }

        }
        else if(type.equalsIgnoreCase("dns")){
            DnsRule dnsRule = new DnsRule();
            if(!opeartion.equalsIgnoreCase("list")){
                if( data.length == 1 ){
                    dnsRule.setDnsRuleName(data[0]);
                    //dnsRule.setSourceAddress(data[1]);
                }else{
                    System.out.println("Invalid arguments.");
                    System.out.println(usage);
                    System.exit(0);
                }
            }
            try{
                oprMgr = (OperationManager) getPort(WS_NAME, proto, port, host);

                /*
                 * Invoke login() API to get session token.
                 */
                st = oprMgr.login(user, pass);

                if(opeartion.equalsIgnoreCase("activate")){
                    System.out.println("Start: Activate dns rule");
                    /*
                     * Invoke the activate api to activate the dns rule in the given
deviceId.
                     */
                    oprMgr.activateDnsRule(st, GSS, dnsRule, "Activate DNS rule from Java
WS Client.");

                    System.out.println("Completed: Activate dns rule");
                }else if(opeartion.equalsIgnoreCase("suspend")){

                    System.out.println("Start: Suspend dns rule");
                    /*
                     * Invoke the activate api to suspend the dns rule in the given
deviceId.
                     */
                    oprMgr.suspendDnsRule(st, GSS, dnsRule, "Suspend DNS rule from Java WS
Client.");

                    System.out.println("Completed: Suspend dns rule");
                }else if (opeartion.equalsIgnoreCase("list")){
                    DeviceIDArray deviceIdArr =  oprMgr.listDeviceIds(st, DeviceType.GSS);
                    List listDeviceId = deviceIdArr.getItem();
                    System.out.println("Start: List DNS rules");
                    for( DeviceID gss: deviceIdArr.getItem() ){
                        /*
                         * Invoke the list api to get the  list of dns rule from the given
deviceId.
                         */
                        DnsRuleArray dnsruleArray = oprMgr.listDnsRules(st, gss);
                        int list_size = 0;
                        if (dnsruleArray.getItem()!= null &&
!dnsruleArray.getItem().isEmpty())
                            System.out.println("GSS name\t | DNS name\t | Source Address \t
| Domains \t | Config state \t | Answer Group \t | Owner \t");
```

```
System.out.println("------------------------------------------------------------------------
------------------------------------------------------------------------------------------
-----------------");
                        for( DnsRule dns: dnsruleArray.getItem() ){
                            System.out.println(gss.getName() + "\t | " +
dns.getDnsRuleName() + "\t | " + dns.getSourceAddress() + "\t\t | "
                                    + dns.getDomains() + "\t\t | " + dns.getDnsConfigState()
+ "\t\t | "
                                    + dns.getAnswerGroups() + "\t\t | " + dns.getOwner() +
"\t\t ");
                        }
                        if( dnsruleArray.getItem().size() == 0) {
                            System.out.println("No dns rule is configured in the given GSS
device" );
                        }

System.out.println("------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------");
                    }
                    if( deviceIdArr.getItem().size() == 0) {
                        System.out.println("No GSS device is imported in this ANM server"
);
                    }

System.out.println("------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------");
                    System.out.println("Completed: List DNS rule");
                }

            }
            catch(Exception ex){
                ex.printStackTrace();
            }finally{
                try{
                    /*
                     * Close the ws session using logout() API.
                     */
                    oprMgr.logout(st);
                }catch(Exception ex){
                    ex.printStackTrace();
                }
            }
        }
        else
        {
            System.out.println("Invalid arguments.");
            System.out.println(usage);
            System.exit(0);

        }

    }
}
```

Use Case 5: ApplTemplateCreateWSClient

The following example code shows how to obtain application template instances and definitions lists, and how to create and delete a template instance.

```
/**
 * Title:       ApplTemplateCreateWSClient
 * Description: Sample WS client code using java. It lists the Application Template
Instances, List of template definition
 *             Create and delete template options *
 * Copyright:   Copyright(C) 2011 Cisco Systems. All Rights Reserved.
 * Company:     Cisco Systems
 * @author:     kamachan, Sep 2011
 * @version:    1.0
 */
package com.cisco.anm.client;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;

import com.cisco.anm.ApplicationTemplateManager;
import com.cisco.anm.DeviceID;
import com.cisco.anm.DeviceIDArray;
import com.cisco.anm.DeviceType;
import com.cisco.anm.OperationManager;
import com.cisco.anm.Session;
import com.cisco.anm.SessionToken;
import com.cisco.anm.TagName;
import com.cisco.anm.TemplateDefinition;
import com.cisco.anm.TemplateDefinitionArray;
import com.cisco.anm.TemplateInput;
import com.cisco.anm.TemplateInputArray;
import com.cisco.anm.TemplateInstance;
import com.cisco.anm.TemplateInstanceArray;
import com.cisco.anm.Vserver;

/**
 * @author kamachan
 *
 *         Description: Sample WS client code using java. It lists the
 *         Application Template Instances, List of template definition Create
 *         and delete template options * Prerequisite: To compile the file,
 *         include the stubs generated from the wsdl file. It is already packed
 *         in a jar (????) file - dcm-ws-client-<version>.jar ????
 *
 *
 */
public class ApplTemplateWSClient {
    static final String NAMESPACE = "http://anm.cisco.com";
```

```
    static String URL_TEMPLATE = "%s://%s:%s/anm/%s"; //
<protocol>://<IPAddress>:<Port>/anm/<service
    // name>
    static final String WS_NAME = "OperationManager";

    public static void main(String args[]) {

        // validate the args
        // args[0] = protocol , http or https
        // args[1] = IP Address or hostname of ANM server
        // args[2] = username,
        // args[3] = password.

        String usage = "Usage: java com.cisco.anm.client.ApplTemplateCreateWSClient
(http|https) <ANM host name> <anm username> <anm password>  \n"
                + "Example: java com.cisco.anm.client.ApplTemplateCreateWSClient http
10.77.241.134 admin admin \n\n";

        if (args.length != 4) {
            System.out.println("Invalid arguments.");
            System.out.println(usage);
            System.exit(0);
        }

        ApplicationTemplateManager applMgr = null;
        SessionToken st = null;
        OperationManager oprMgr = null;
        Session session = null;
        int len = args.length;

        /*
         * ***********USER INPUTS NEEDS TO BE GIVEN HERE******************
         * Inputs should be given based on the xml variable elements
         */

        // Input for the Layer 3 LB template instance creation
        String templateName = "Layer 3 LB";
        // Template Name
        String name = "layer3lb-5";
        // Virtual server ip address. IPv4/IPv6 address supported Ip Address
        // with prefix length can be given.
        String vip = "142.12.3.3/128";
        // Real server ip address. Can be given as comma seperated. IPv4/IPv6
        // address supported
        String ipAddr = "144.12.3.3,144.12.3.144,44.12.3.4";
        // Sticky can be enable by providing this value.
        String sticky = "false";
        // all vlan can be given or the configured vlan can be given
        String vlans = "ALL_VLAN";
        // Enable soure nat
        String autoNat = "false";

        // User need to give the contex name here
        String contextName = "";
        // Need to give the ip address for the Appliance or chassis ip address
        // for the ACE module
        String ipAddress = "";
        // Slot should be "0" for the Appliance and the corresponding slot
        // number should be given for the ACE module
        String slot = "";

        /*
         * To Delete the device instance need to provide the instance id
         */
```

```
        int instanceId = -1;

        // ********* USER INPUT ends here************************

        String proto = args[0], port = "", host = args[1], user = args[2], pass = args[3];

        if (proto != null && proto.equals("https")) {
            port = "8443";
        } else if (proto.equals("http")) {
            port = "8080";
        } else {
            System.out
                    .println("Invalid arguments. Please provide the http/https protocol in
the argument");
            System.out.println(usage);
            System.exit(0);
        }

        DeviceID devId = new DeviceID();

        try {

            /*
             * Formulate the mapping based on the input need to given for the
             * specific application template input. Need to check the App
             * Template xml before putting the key value for other App
             * Templates.
             */

            // This map contains the Application as the key. Example
            // "Layer 3 LB" is key here.
            Map<String, Map<String, String>> inputData = new HashMap<String, Map<String,
String>>();
            // Based on the variable name user need to formulate the map for
            // other App templates.
            Map<String, String> userInput = new HashMap<String, String>();
            userInput.put("name", name);
            userInput.put("vip", vip);
            userInput.put("ipAddr", ipAddr);
            userInput.put("sticky", sticky);
            userInput.put("vlans", vlans);
            userInput.put("autoNat", autoNat);
            inputData.put(templateName, userInput);

            session = (Session) getPort("Session", proto, port, host);
            oprMgr = (OperationManager) getPort("OperationManager", proto,
                    port, host);
            applMgr = (ApplicationTemplateManager) getPort(
                    "ApplicationTemplateManager", proto, port, host);

            // Invoke login() API to get session token.
            st = session.login(user, pass);

            /*
             * Use the session token obtained from the previous call and get all
             * ACE modules imported in ANM.
             */
            DeviceIDArray deviceIdArr = oprMgr.listDeviceIds(st,
                    DeviceType.VIRTUAL_CONTEXT);
            List<DeviceID> listDeviceId = deviceIdArr.getItem();

            if (listDeviceId == null || listDeviceId.size() == 0) {
                System.out
```

```
                        .println(" ACE module or Ace appliance needs to be imported to
porvision the template instance ");
            } else {
                if (contextName.equals("") && ipAddress.equals("")) {
                    devId = listDeviceId.get(0);
                } else {
                    devId.setDeviceType(DeviceType.VIRTUAL_CONTEXT);
                    devId.setName(contextName);
                    if (slot.equalsIgnoreCase("") || slot.equalsIgnoreCase("0"))
                        devId.setIpAddr(ipAddress);
                    else {
                        devId.setChassisIPAddr(ipAddress);
                        devId.setSlot(slot);
                    }
                }

                TemplateInstanceArray templateInstanceArray = applMgr
                        .listTemplateInstances(st);

                // applMgr.
                List<TemplateInstance> instanceList = templateInstanceArray
                        .getItem();
                Map<Integer, TemplateInstance> instanceMap = new HashMap<Integer,
TemplateInstance>();

                for (TemplateInstance tempInstance : templateInstanceArray
                        .getItem()) {
                    instanceMap.put(tempInstance.getInstanceId(), tempInstance);
                }
                Set keySet = (Set) instanceMap.keySet();
                List<Integer> list = new ArrayList<Integer>(keySet);
                Collections.sort(list);
                System.out.println("List of created Template Instances");
                System.out

.println("-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------");
                System.out
                        .println("InstanceID  | Template name\t | Status\t | Type\t |
Template type\t\t |Device\t\t | Last Updated Time\t");
                System.out

.println("-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------");
                for (Integer keynum : list) {

                    TemplateInstance tempInstance = instanceMap.get(keynum);
                    System.out.println(tempInstance.getInstanceId() + " | "
                            + tempInstance.getTemplateName() + "\t\t | "
                            + tempInstance.getStatus() + "\t\t | "
                            + tempInstance.getType() + "\t\t | "
                            + tempInstance.getApplicationType() + "\t   | "
                            + tempInstance.getDeviceId() + "\t\t | "
                            + tempInstance.getLastUpdatedTime() + "\t\t  ");
                }
                System.out

.println("-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------");

                TemplateDefinitionArray definationArray = applMgr
```

```
                .listTemplateDefinitions(st);
            List<TemplateDefinition> definationList = definationArray
                .getItem();
            Map<Integer, TemplateDefinition> definationMap = new HashMap<Integer,
TemplateDefinition>();
            for (TemplateDefinition defination : definationArray.getItem()) {
                definationMap.put(new Integer(defination.getId()),
                        defination);
            }

            keySet = (Set) definationMap.keySet();
            list = new ArrayList<Integer>(keySet);
            Collections.sort(list);

            System.out.println("List of Available Template Definations");
            System.out

.println("-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------");
            System.out
                .println("TemplateID | Template\t\t | Template version | Template
Name\t | Template Description\t ");
            System.out

.println("-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------");
            for (Integer id : list) {
                TemplateDefinition tempDefn = definationMap.get(id);
                System.out.println(tempDefn.getId() + "|"
                        + tempDefn.getApplication() + "\t\t | "
                        + tempDefn.getAppVersion() + "| "
                        + tempDefn.getName() + "\t | "
                        + tempDefn.getDescription() + "\t  ");
            }
            System.out

.println("-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------");
            System.out
                .println("Template Definations metadata for the App Template :"
                        + templateName);
            System.out

.println("-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-------");
            System.out
                .println("Tag Name\t |Input Name\t | InputType\t | Child Element
size\t ");
            for (int j = 0; j < definationList.size(); j++) {

                TemplateInputArray tia = applMgr
                        .getTemplateDefinitionMetadata(st, definationList
                                .get(j));
                if (definationList.get(j).getApplication()
                        .equalsIgnoreCase(templateName)) {

                    System.out.println(definationList.get(j)
                            .getApplication()
                            + "\t\t | ");
                    for (TemplateInput tempInput : tia.getItem()) {
```

```
                                System.out.println(tempInput.getTagName() + "\t | "
                                        + tempInput.getName() + "\t | "
                                        + tempInput.getType() + "\t | "
                                        + tempInput.getChildElements().size()
                                        + "\t  ");
                                if (tempInput.getChildElements().size() >= 1) {
                                    for (TemplateInput temp : tempInput
                                            .getChildElements()) {
                                        System.out.println(temp.getTagName()
                                                + "\t | " + temp.getName()
                                                + "\t | " + temp.getType()
                                                + "\t | "
                                                + temp.getChildElements().size()
                                                + "\t  ");
                                        if (temp.getChildElements().size() >= 1) {
                                            for (TemplateInput input : temp
                                                    .getChildElements()) {
                                                System.out.println(input
                                                        .getTagName()
                                                        + "\t | "
                                                        + input.getName()
                                                        + "\t | "
                                                        + input.getType()
                                                        + "\t | "
                                                        + input.getChildElements()
                                                                .size() + "\t  ");
                                            }
                                        }
                                    }
                                }
                            }

                            if (setData(tia, definationList.get(j), inputData)) {
                                System.out

.println("----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-------");
                                System.out
                                        .println("Start creating instance for the templateName:
"
                                                + templateName);
                                int instId = applMgr.createTemplateInstance(st,
                                        devId, definationList.get(j), tia);
                                System.out
                                        .println("Template instance created successfully."
                                                + templateName
                                                + " instance Id is :" + instId);

                            }

                        }
                    }
                    System.out

.println("----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
-------");
                    if (devId != null && instanceId != -1) {
                        System.out
                                .println("Start deleting the configuration related to the
template instance id :"
                                        + instanceId);
                        applMgr.deleteTemplateInstance(st, devId, instanceId);
```

```
                    System.out
                            .println("Deleted the configuration related to the instance id
:"
                                    + instanceId);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                // Close the ws session using logout() API.
                oprMgr.logout(st);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }

    private static boolean setData(TemplateInputArray tia,
            TemplateDefinition td, Map<String, Map<String, String>> data) {

        Map<String, String> dt = data.get(td.getApplication());
        boolean retval = true;
        if (dt == null) {
            retval = false;
            return retval;
        }
        System.out

.println("-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-------");
        System.out.println("Application Template: " + td.getApplication()
                + "\n");

        for (Map.Entry<String, String> entry : dt.entrySet()) {
            System.out.println("TemplateInput element = " + entry.getKey()
                    + ", Value = " + entry.getValue());
        }

        for (TemplateInput ti : tia.getItem()) {
            setValue(ti, dt);
        }

        return retval;
    }

    private static void setValue(TemplateInput ti, Map<String, String> dt) {

        if (ti.getTagName().equals(TagName.VARIABLE)) {
            ti.setUserData(dt.get(ti.getName()));
            if (ti.getChildElements().size() > 0) {
                for (int i = 0; i < ti.getChildElements().size(); i++) {
                    TemplateInput inp = (TemplateInput) ti.getChildElements()
                            .get(i);
                    setValue(inp, dt);
                }
            }
        } else if (ti.getTagName().equals(TagName.VARIABLE)) {
            ti.setUserData(dt.get(ti.getName()));
        } else if (ti.getTagName().equals(TagName.ARRAY)) {
            for (TemplateInput e : ti.getChildElements()) {
                String[] val = dt.get(e.getName()).split(",");
                for (String v : val) {
```

```
                    e.getUserDatas().add(v);
                }
            }
        } else if (ti.getChildElements().size() > 0) {
            for (int i = 0; i < ti.getChildElements().size(); i++) {
                TemplateInput inp = (TemplateInput) ti.getChildElements()
                        .get(i);
                setValue(inp, dt);
            }
        }
    }

    public static BindingProvider getPort(String ws, String protocol,
            String host, String port) throws IOException {

        String url = String.format(URL_TEMPLATE, "" + protocol, port, host, ws);
        String ns = NAMESPACE;
        BindingProvider bp;
        Service svc;
        try {
            QName qname = new QName(ns, ws + "Service");
            Class<Service> clazz = (Class<Service>) Class
                    .forName("com.cisco.anm." + ws + "Service");
            Constructor<Service> ctor = (Constructor<Service>) clazz
                    .getConstructor(new Class[] { URL.class, QName.class });
            svc = ctor
                    .newInstance(new Object[] { new URL(url + "?wsdl"), qname });
            Method m = svc.getClass().getMethod("get" + ws + "Port", null);
            bp = (BindingProvider) m.invoke(svc, null);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return bp;
    }
}
```