Cisco License Manager Software Developer's Kit Cookbook

Chapter 1: Overview

- What is the Cisco License Manager Software Developer's Kit (SDK)?
- What can Cisco License Manager SDK do for me?
- What do I need to use Cisco License Manager SDK?
- What is in this cookbook?

What Is Cisco License Manager SDK?

Cisco License Manager SDK is a set of Java class libraries that can be linked with your Java application to invoke licensing functions provided by the Cisco License Manager server. These licensing functions include obtaining licenses from Cisco, deploying licenses to devices, and managing inventory of licenses and devices.

What Can Cisco License Manager SDK Do for Me?

Cisco License Manager SDK provides the same set of class libraries used by the Cisco License Manager GUI client program. By using the Cisco License Manager SDK in your Java application, you can integrate full functionality of Cisco License Manager into your own program.

What Do I Need to Use Cisco License Manager SDK?

Before you can start using Cisco License Manager SDK, make sure that:

- You have the Cisco License Manager server installed and running on your computer.
- You have the Cisco License Manager Java SDK installed on your computer.
- You have Java Version 1.6 or later for your Java development environment.

What Is in This Cookbook?

This cookbook covers the most common scenarios for using Cisco License Manager SDK and provides instructions on how to achieve them. You can read each chapter sequentially or independently of other chapters. If you read them independently, we recommend that you read the first three chapters before moving on to other chapters.

Chapter 2: Installation and Setup

- Install Cisco License Manager SDK on my computer
- How to set up my working environment

Installation

To install the Cisco License Manager SDK, complete these steps:

Step 1. Copy clm_java_sdk.zip from the CD to a local drive.

Step 2. Unzip the file.

The folder “CiscoLicenseManager_JAVA_SDK” is created on the local drive.

It contains these files:

- clm-sdk.jar
- clm_ssl_rmi.cert
- javadoc.zip
- conf, a folder containing the file ClmErrorMessages.properties

Setup

To begin using the SDK, complete these steps:

Step 1. Add clm-sdk.jar to the Java classpath when compiling and running the client program. You must use Java Version 6.0 to run Cisco License Manager Release 3.0 SDK.

```
javac –classpath <CLM_SDK_HOME>/clm-sdk.jar:<CLM_SDK_HOME>/conf MyProgram.java
```

`<CLM_SDK_HOME> : the directory that holds the CLM SDK files.`

Step 2. If the Cisco License Manager server has Secure Sockets Layer (SSL) enabled, run the following command to import the Cisco License Manager certificate file, clm_ssl_rmi.cer, to the trust store on your client system:

```
<JAVA_HOME>\bin\keytool –import –keystore <keystore_file_path> -alias
ciscolicnesemanagerca –file <clm_ssl_rmi_cer_file_path>
```

```
      <keystore_file_path> : trust store of your client system
      <clm_ssl_rmi_cer_file_path> : the full path of the clm_ssl_rmi.cer file
       for example,
      C:\jdk1.6\jre\bin\keytool –import –keystore
C:\jdk1.6\jre\lib\security\cacerts –alias ciscolicensemanagerca –file
c:\clm3.0_SDK\clm_ssl_rmi.cer
```

Step 3. Start the Cisco License Manager server. For details, see the *User Guide for Cisco License Manager*.

Step 4. Start the client program.

Chapter 3: Getting Started

- My first Cisco License Manager client program, “HelloCLMServer”
- Connect and log in to the Cisco License Manager server
- Compile my client program
- Run my client program

Solution

In this chapter, you create your first client program by using the Cisco License Manager SDK. Your first program, HelloCLMServer, connects and logs you in to the Cisco License Manager server. To achieve this, you complete the following processes in your programs:

Instantiate the LicenseManager Object

First, you create an instance of the LicenseManager class:

```
static LicenseManager lmInstance = new LicenseManager();
```

You usually declare the instance as static so that you can use it in other places of your program.

Log In to Cisco License Manager Server

Assume that you have installed Cisco License Manager server on the computer “johndoe-server,” and you configured the Cisco License Manager server to use the default port 1099. Assume that you want to log in as “admin” user with password “cisco,” and you do not want the connection to time out. This is your code snippet:

```
// Create an instance of signed EULA agreement.
EulaInfo eula = new EulaInfo(Calendar.getInstance().getTime(), true, true);
// Log in to CLM server
UserToken token = lmInstance.login("admin", "cisco",
"johndoe-server", 1099, 0, eula);
```

You can use the returned token as an input parameter in the subsequent calls to functions defined in the LicenseManager class.

Log Out from Cisco License Manager Server

You should always log out from the Cisco License Manager server at the end of your program so that the resource for the connection can be released:

```
lmInstance.logout(token);
```

Sample Program

Here is the complete program of HelloCLMServer.java:

```
import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;
import java.rmi.RemoteException;
import java.util.Calendar;


public class HelloCLMServer {
```

```
static protected LicenseManager lmInstance;

public void sayHello() {

   // Create an instance of LicenseManager class.
   lmInstance = new LicenseManager();

   // Create an instance of signed EULA agreement.
   EulaInfo eula =
             new EulaInfo(Calendar.getInstance().getTime(), true, true);

   try {
      System.out.println("Say Hello to CLM Server...");

      // Log in to CLM server
      UserToken token = lmInstance.login("admin",
          "cisco", "johndoe-server", 1099, 0, eula);
      if (token != null) {
         System.out.println("Login successful!
      } else {
         System.out.println("Login failed.");
      }

      // Always log out from CLM server at the end
      if (token != null) {
         lmInstance.logout(token);
      }
   } catch (RemoteException e) {
      e.printStackTrace();
   }

   // main of the program.
   public static void main(String[] args) {
      HelloCLMServer hello = new HelloCLMServer();
      hello.sayHello();

}
```

Compile the Program

To compile the program, run this command line:

```
javac -cp <CLM_SDK_HOME>/clmsdk.jar:<CLM_SDK_HOME>/conf
        HelloCLMServer.java
```

where <CLM_SDK_HOME> is the directory that holds the Cisco License Manager SDK files.

Run the Program

To run the program, make sure that the Cisco License Manager server is running on computer “johndoe-server,” and then run this command line:

```
java -cp <CLM_SDK_HOME>/clmsdk.jar:<CLM_SDK_HOME>/conf
        HelloCLMServer
```

See Also

- For a more detailed description of the functions used in this program, see Chapter 16 of the *Java API Reference Guide for Cisco License Manager*.
- If you need to see logs for troubleshooting purposes, you can find them in <CLM_HOME>/log/clm.logfile, where <CLM_HOME> is the directory that holds the Cisco License Manager server installation.

Chapter 4: Device Discovery

- Discover licensable devices in my network
- What protocol should I choose?
- How do I organize discovered devices in groups?
- Schedule a periodic device discovery

Solution

In this chapter, you learn how to discover devices by using the network IP address and a network mask and also how to schedule periodic device discovery to add new devices.

Define and Instantiate a Callback Object

Cisco License Manager discovery runs asynchronously, which means that when this function is called, it runs on the server side and immediately returns a request ID. Therefore, before calling the discover function, you must define a callback function for the Cisco License Manager server to call when discover is complete.

To define your callback function, create your own IDStatusListener class and implement the onStatus function to handle the incoming data.

```
public class MyIDStatusListener implements IDStatusListener {
   public void onStatus(String requestId, IDStatus status) {
      //overwrite this area for your data handling
   }
}


MyIDStatusListener _listener=new MyIDStatusListener();
```

Log In to the Cisco License Manager Server

Log in to the Cisco License Manager server.

```
// Create an instance of signed EULA agreement.
EulaInfo eula = new EulaInfo( true, true);


//log in to CLM server
UserToken token = lmInstance.login(username, password,
 serverHost, 1099, 0, eula);
```

Construct a Discovery Request

Using the handle of Cisco License Manager, you can use the asyncDiscoveryDevices API, and provide these parameters:

- User token that you acquired during log in
- Network IP address you want to discover
- Network mask for the range of the network
- Group name that you want the discovered devices to be placed in
- Device access username and password pairs

- Transport methods you want to use to discover the devices
- Policy IDs that you want run after discovery is completed
- Callback function to be called by the Cisco License Manager server

```
String req_id=lmInstance.asyncDiscoveryDevices(token, “172.1.120.0”, “255.255.255.0”,
null, devAuthInfo, transportMethods,, policyIds, _listener);
```

The API is defined as follows:

```
public String asyncDiscoverDevices(
UserToken token, String subnet,
        String subnetMask, String group,
DiscoveryAuthInfo devAuthInfo,
        Device.TransportMethod[] transportMethods,
String[] policyIds,
        IDStatusListener listener)throws RemoteException
```

- Subnet and Subnet Mask

  For the subnet and subnet mask, you can enter either a network address or an individual IP address. Cisco License Manager converts it to a network address based on the subnet mask and calculates all possible IP addresses in this subnet.

- Grouping

  You can organize your discovered devices in groups. There is a Default group already created for you. If you want to put a device in the Default group, just put null; otherwise, make sure the group is already created. Cisco License Manager rejects this call if there is no group found in Cisco License Manager Inventory.

  To create a group, call the following API before calling this function:

  ```
  public Status createDeviceGroup(UserToken token, String group)
           throws RemoteException


    Status _st=lmInstance.createDeviceGroup(token, “MyGroupName”);
  ```

- Providing Discovery Authentication Information

  You must provide the username and password pairs using the DiscoveryAuthInfo object. The class is defined as follows:

  ```
  public class DiscoveryAuthInfo implements Serializable {
      private UserPwd[] authpairs;

      public DiscoveryAuthInfo(UserPwd[] auths){
          authpairs = auths;
      }

      public UserPwd[] getUserPwdPairs(){
          return authpairs;
      }
  }
  ```

```
public class UserPwd implements Serializable {
    private String username = "";
    private String passwd = "";
    public UserPwd(String name, String pwd);
}
```

- To discover all devices in the subnet, you might want to enter all possible username and password pairs.
- The enable password also uses the same UserPwd class by assigning only the enable password as the password.
- Because Cisco License Manager uses all possible combinations of the username/password pairs, it might take a long time to complete if you enter too many username/password pairs.

- Choosing Protocols for Discovery

  Cisco License Manager supports four different kinds of transport methods or protocols for discovery: HTTP, HTTPS, Telnet, and Secure Shell (SSH) Protocol. They are defined in the Device class as public enum values.

  ```
  public enum TransportMethod {HTTP, HTTPS, TELNET, SSH};
  ```

  1. You can specify any combination of the transport methods that you want to use for discovery.

  2. If all of your network devices are license agent enabled, you can use only HTTP. Otherwise, you can use HTTP and Telnet.

  3. Using all four transport methods results in longer discovery time.

  4. If you have a network with a variety of devices using all sorts of protocols for access and different username/password/enable password, you might want to discover them in different discovery calls to reduce discovery time.

- Running a Policy

  You can also run a policy after discovery by providing the policy IDs in the call. The policy should be created so that you can pass the policy ID. To learn how to create a policy, see Chapter 12: Policy Licensing.

Schedule a Discovery

You can schedule a device discovery by registering a discovery task into the Cisco License Manager Scheduler. The parameters of the scheduler are similar to those for regular device discovery plus the frequency of the discovery. Scheduled discovery allows you to enter more than one set of network addresses and masks.

```
public Status registerScheduler(UserToken token,
     ClmTask taskName,
          Schedule schedule )
     throws RemoteException;
```

Where

- taskName is an enum value and should be defined as ClmTask.DISCOVER_DEVICES
- schedule object should contain the starting date, frequency of the discovery, discoverySettings, and policyIds

The Schedule class contains DiscoverySetting, which is designed to contain all the necessary parameters for running discovery. The attributes in the class definition are the same as the parameters used in asyncDiscoverDevices.

```
public class DiscoverySetting implements Serializable {
       private TransportMethod[] transmethods;
       private String devgroup;
       private DiscoveryAuthInfo discoveryAuth;
       private String netIpaddr;
       private String netMask;
       private String[] policyIds;
}
```

See the preceding bullet “Construct a Discovery Request” for the parameter details.

```
lmInstance.registerScheduler(m_token, ClmTask.DISCOVER_DEVICES, dsched);
```

Log Out from the Cisco License Manager Server

You should always log out from the Cisco License Manager server at the end of your program so that the resource for the connection can be released:

```
lm_instance.logout(token);
```

Sample Program

Device Discovery Sample Code

Following is the complete Java sample program to discover devices in a subnet:

```
import java.rmi.RemoteException;
import java.util.*;
import org.apache.log4j.Logger;
import com.cisco.nm.clm.common.*;
import com.cisco.nm.clm.sdk.LicenseManager;


public class DiscoveryTest {
static final protected String m_server_host = "localhost";
static final protected String m_username = "admin";
   static final protected String m_password = "cisco";
   static final protected String m_subnet = "172.221.111.0";
static final protected String m_subnet_mask = "255.255.255.0";
   static protected String[] m_device_ids = null;
   static protected Device[] m_device_objs= null;
   static protected UserToken m_token = null;
   static protected Date m_today;
   static protected int m_counter = 0;
   static protected DiscoveryTest m_thread = null;
   static protected boolean m_IDStatusListenerInvoked = false;
```

```
   static Logger m_logger = Logger.getLogger(DiscoveryTest.class);
   final static String USER_NAME = "admin";
IDStatus _st=null;

//Define IDStatusListener class and instantiate it
//this listener object will be passed to the server

   public class MyIDStatusListener implements IDStatusListener {
      public void onStatus(String request_id, IDStatus status) {
m_logger.debug("MyIDStatusListener for " + request_id +
" invoked");
_st=status;
         System.out.println("received notification.........");
         m_IDStatusListenerInvoked = true
      }
   }
   protected static LicenseManager m_lm_instance =new LicenseManager(88888);
   private static EulaInfo _eula=new EulaInfo(true, true);
   public static void main(String[] args) {
      Calendar cldr = Calendar.getInstance();
      EulaInfo eula = new EulaInfo(cldr.getTime(), true, true);
      try {
         //login function as get a token back
         m_token = m_lm_instance.login(m_username, m_password,
m_server_host, 1099, 0, eula);
         DiscoveryTest _dt=new DiscoveryTest();
         _dt.testAsyncDiscoverDevices();
         m_lm_instance.logout(m_token);
         System.out.println("return from discovery and exit");
         System.exit(0);
      } catch (RemoteException e) {
         e.printStackTrace();
      }
   }
   public void testAsyncDiscoverDevices() throws RemoteException {
      m_IDStatusListenerInvoked=false;
             //Instantiate user-defined listener
             MyIDStatusListener my_listener = new MyIDStatusListener();
```

```
            //Define username password pairs for DiscoverAuthInfo
            UserPwd[] ups= {new UserPwd("lab", "lab"), new UserPwd("cisco",
            "cisco")};
      DiscoveryAuthInfo  dev_auth_info=new DiscoveryAuthInfo(ups);
      //Define what transport methods are used in this discovery
      Device.TransportMethod[] transports={TransportMethod.HTTP,
TransportMethod.TELNET};
      /*
            Calls to the asyncDiscoverDevices API will return right away with a
            unique request ID. You can query the CLM server with the request ID for
            the status of the task using the getAsyncOperationStatus API.
            The while loop checks whether the listener callback has been called by
            CLM server and sets m_IDStatusListenerInvoked to true to break out of the
            loop.
            */
            String req_id = m_lm_instance.asyncDiscoverDevices(m_token, m_subnet,
            m_subnet_mask, null, dev_auth_info, transports, null, my_listener);
      m_logger.debug("testAsyncDiscoverDevices starts, jobid=" + req_id);
      while(!m_IDStatusListenerInvoked){
           try {
            Thread.sleep(2000);
                        ProgressStatus
                        _ps=m_lm_instance.getAsyncOperationStatus(m_token, req_id);
                if(_ps !=null){
                              System.out.println("num dev found=" +
                              _ps.getNumDevFound());
}
           } catch (InterruptedException e) {
            e.printStackTrace();
          } catch(Exception ex){
            ex.printStackTrace();
           }
      }
      System.out.println("Got notification and out of the loop");
   }
}
```

Schedule Discovery Task Sample Code

Following is the complete Java program to schedule a discovery task:

```
import java.rmi.RemoteException;
import java.util.Calendar;
import java.util.Date;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import com.cisco.nm.clm.common.*;
import com.cisco.nm.clm.sdk.LicenseManager;

public class SchedulerDiscovery {

   static final protected String m_server_host = "localhost";
   static final protected String m_username = "admin";
   static final protected String m_password = "cisco";
   static final protected String m_subnet = "172.222.205.0";
   static final protected String m_subnet_mask = "255.255.255.0";
   static protected String[] m_device_ids = null;
   static protected Device[] m_device_objs= null;
   static protected UserToken m_token = null;
   static protected Date m_today;
   static protected int m_counter = 0;
   static protected DiscoveryTest m_thread = null;

   final static String USER_NAME = "admin";
   IDStatus _st=null;

   static Logger m_logger =
      Logger.getLogger(DiscoveryTest.class);

   static protected LicenseManager m_lm_instance = null;

   public static void main(String[] args) {

      if (m_lm_instance == null) {
         m_lm_instance = new LicenseManager();
```

```
    }
    m_today = Calendar.getInstance().getTime();
    Calendar cldr = Calendar.getInstance();
    EulaInfo eula = new EulaInfo(cldr.getTime(), true, true);
    try {
                m_token = m_lm_instance.login(m_username, m_password,
                m_server_host, 1099, 0, eula);
       SchedulerDiscovery _st=new SchedulerDiscovery();

       _st.registerDiscoveryScheduler();
       _st.unRegisterDiscoveryScheduler();

       m_lm_instance.logout(m_token);

    } catch (RemoteException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
    }
}

/*
* This function will schedule a discovery task
*/
public  void registerDiscoveryScheduler() throws RemoteException {

    m_logger.debug("testRegisterScheduler starts");

    ClmTask taskName taskName=ClmTask.DISCOVER_DEVICES;

    DiscoverySchedule dsched=null;
    try {
       //Instantiate a Schedule for discovery
       dsched = new DiscoverySchedule();

       /*
       * Preparing parameters for DiscoverySetting, first setup
       * username, password pairs and passed in DiscoveryAuthInfo.
       * then define network address and network mask.
```

```
        * You can define multiple DiscoverySetting in one schedule.
        */
                    UserPwd[] ups={new UserPwd("lab","lab"), new UserPwd("cisco",
                    "lab")};
        DiscoveryAuthInfo auth_info=new DiscoveryAuthInfo(ups);
        DiscoverySetting[] infos={new DiscoverySetting("172.222.205.0",
        "255.255.255.0", null, null, auth_info, null)};
        dsched.setDiscoverySettings(infos);

        //set the execution frequency to everyday
        dsched.setFrequency(1);

        //setting start time and date of the discovery task
        Calendar start=Calendar.getInstance();
        start.set(2009, 12, 20);
        dsched.setStartingDate(start);
     } catch (ClmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
     }

     //register discovery schedule
     _st = m_lm_instance.registerScheduler(m_token, taskName, dsched);

     m_logger.debug("testRegisterScheduler ends");
}

     /*
* This function will remove a scheduled discovery task
*/

public  void unRegisterDiscoveryScheduler() throws RemoteException {

     m_logger.debug("testUnRegisterScheduler starts");
     m_logger.debug("testUnRegisterScheduler CHECK_EXPIRING_LICENSES");

     //define task to be removed from the schedule
     ClmTask taskName=ClmTask.DISCOVER_DEVICES;
```

```
        //calling unregister scheduler
        m_logger.debug("testUnRegisterScheduler DISCOVER_DEVICES");
        _st = m_lm_instance.unregisterScheduler(m_token, taskName) ;

        m_logger.debug("testUnRegisterScheduler ends");
    }

}
```

See Also

- For more detailed descriptions of functions used in this program, see Chapter 5 of the *Java API Reference Guide for Cisco License Manager*.

Chapter 5: Adding Devices to Inventory

- How do I add new devices without going through discovery?
- Add new device by IP address
- Add new device by UDI

Solution

In this chapter, you learn how to create a device with a unique device identifier (UDI) or with an IP address.

Create Device by UDI

This API is mainly used in managing devices in an isolated network. See the “Two Stage Deployment” section of the following white paper for more information:

http://www.cisco.com/en/US/prod/collateral/netmgtsw/ps5734/ps7138/white_paper_C11-562757.html

In a two-stage environment, you first create a device with its UDI and obtain a license with the UDI in a network that can access Cisco.com. In the second stage, you bring the server to a private network where the device resides, fill in device connection information, and poll or discover the device and deploy licenses obtained in the first stage. This chapter only covers the creation part. Obtaining licenses, polling licenses, and discovering devices are covered by other chapters.

Assume you already instantiated the LicenseManager object (licMgr), logged in to the server, and obtained a valid token. This example creates a device with a UDI “C3900-SPE100/K9:FHH13010044” in a group called “myGroup”. The UDI is a unique ID used to identify a device and has the format of <ProductID>:<SerialNumber>. In this example, C3900-SPE100/K9 is the product ID, and FHH13010044 is the serial number.

- Create the group if it does not exist already.

```
licMgr.createDeviceGroup(token, group);
```

- Create a device with a given UDI.

```
String[] udis = { “C3900-SPE100/K9:FHH13010044” }
DeviceStatus status =
   licMgr.createDevicesByUDI(token, udis, group);
```

- Check the top-level return value to see whether the operation is successful.

```
if (status.getErrorCode() != ClmErrors.SUCCESS) {
   System.out.println(“Operation failed, msg: “ +
   status.getErrorMessage());
   return;
}

DeviceStatusItem[] items =
   status.getDeviceStatusItems();
```

- Check the return value for the individual item. In this example, there is only one item.

```
for (int i = 0; i < udis.length; i++) {
   if (items[i].getErrorCode()!=ClmErrors.SUCCESS) {
      System.out.println(“Operation failed for UDI “
      udis[i] + “, msg: “ + items[i].getErrorMessage());
```

```
    }
    else {
        Device dev = items[i].getDevice();
        // continue process device
}
```

Create Device by IP Address

Assume you already instantiated the LicenseManager object (licMgr), logged in to the server, and obtained a valid token.

This example creates a device with an IP address and login information.

- Set the device login credentials. This includes the username, password, and the enable password, if applicable. You can set multiple pairs, and the API tries all possibilities until it is successful.

```
UserPwd[] userpwds = new UserPwd[2];
userpwds[0] = new UserPwd(“usr1”, “pwd1”);
userpwds[1] = new UserPwd(“usr2”, “pwd2”);
DiscoveryAuthInfo authinfo = new
    DiscoveryAuthInfo(userpwds);
```

- Set the transport method. The transport method is used to connect to a device. You can specify multiple transport methods to try. The TransportMethodInfo object contains a transport method and a port number. If a port is set to -1, it uses the default port for the specified protocol.

```
TransportMethodInfo[] transinfo = new
    TransportMethodInfo[2];
transinfo[0] = new TransportMethodInfo(
    Device.TransportMethod.HTTP, -1);
transinfo[1] = new TransportMethodInfo(
    Device.TransportMethod.TELNET, -1);
```

- Create a device with IP address “10.10.1.1”.

```
String ipaddr = “10.10.1.1”;
DeviceStatus status =
    licMgr.createDevicebyIPAddr(token, ipaddr, group, authinfo, transinfo);
```

- Check the top-level return value to see whether the operation is successful.

```
if (status.getErrorCode() != ClmErrors.SUCCESS) {
    System.out.println(“Operation failed, msg: “ +
    status.getErrorMessage());
    return;
}

DeviceStatusItem[] items =
    status.getDeviceStatusItems();
```

- Check the return value for an individual item. There should be only one entry:

```
if (items[0].getErrorCode()!=ClmErrors.SUCCESS) {
   System.out.println(“Operation failed for ip “
      ipaddr + “, msg: “ + items[0].getErrorMessage());
}
else {
   Device dev = items[0].getDevice();
   // continue process device
}
```

Sample Program

Create Device by UDI Sample Code

```
import com.cisco.nm.clm.common.*;
import com.cisco.nm.clm.sdk.*;
import java.util.*;

public class ClmTest {
   public static LicenseManager licMgr = new LicenseManager();

   public static void main(String[] args) {
      try {
         EulaInfo eula = new EulaInfo(null, true, true);
         UserToken token = licMgr.login("admin", "cisco", “localhost", 1099, 0, eula);

         if (token == null) {
            System.err.println("Login failed.");
            System.exit(-1);
         }

         // Create a Device with UDI, and put it in a group
         // called “MyGroup”.
         String group = "MyGroup";

      // Create group if it doesn’t exist
      licMgr.createDeviceGroup(token, group);

      String[] udis = { "UDI1" };
```

```
            DeviceStatus status = licMgr.createDevicesByUDI(token,
            udis, group);

            // check top-level error status
            if (status.getErrorCode() != ClmErrors.SUCCESS) {
               System.out.println("Operation failed, msg: " +
               satus.getErrorMessage());
           return;
            }

            DeviceStatusItem[] items=status.getDeviceStatusItems();

      // check individual error status
      for (int i = 0; i < udis.length; i++) {
        if (items[i].getErrorCode()!=ClmErrors.SUCCESS) {
                System.out.println("Operation failed for UDI " +
                   udis[i] + ", msg: " +
                   items[i].getErrorMessage());
            } else {
                Device dev = items[i].getDevice();
         // continue process device
        }
       }
   }
   catch (Exception e) {}
   }
}
```

Create Device by IP Address Sample Code

```
import com.cisco.nm.clm.common.*;
import com.cisco.nm.clm.sdk.*;
import java.util.*;

public class ClmTest {
   public static LicenseManager licMgr = new LicenseManager();

   public static void main(String[] args) {
      try {
```

```
EulaInfo eula = new EulaInfo(null, true, true);
UserToken token = licMgr.login("admin", "cisco",
   "localhost", 1099, 0, eula);

if (token == null) {
   System.err.println("Login failed.");
   System.exit(-1);
}

// Create a Device with UDI, and put it in Default
// group.
String group = "Default";

// Set device login username and password pairs.
// You can set multiple pairs.
UserPwd[] userpwds = new UserPwd[2];
userpwds[0] = new UserPwd("usr1", "pwd1");
userpwds[1] = new UserPwd("usr2", "pwd2");
DiscoveryAuthInfo authinfo = new
   DiscoveryAuthInfo(userpwds);

// IP address of the device
String ipaddr = "10.10.1.1";

// Transport method used to connect to device
// You can specify multiple transport methods to try.
// TransportMeotdInfo contains transport method and
// port number. If port is set to -1, it uses the
// default port for the specified protocol.
TransportMethodInfo[] transinfo = new
   TransportMethodInfo[2];
transinfo[0] = new
   TransportMethodInfo(Device.TransportMethod.HTTP, -1);
transinfo[1] = new
  TransportMethodInfo(Device.TransportMethod.TELNET,-1);

DeviceStatus status = licMgr.createDeviceByIPAddr(token,
```

```
            ipaddr, group, authinfo, transinfo);

          // check top-level error status
          if (status.getErrorCode() != ClmErrors.SUCCESS) {
             System.out.println("Operation failed, msg: " +
                status.getErrorMessage());
             return;
          }

          DeviceStatusItem[] items=status.getDeviceStatusItems();

          // check individual error status. There should only be
          // one entry
          if (items[0].getErrorCode()!=ClmErrors.SUCCESS) {
        System.out.println("Operation failed for ip: " +
                ipaddr + ", msg: " + items[0].getErrorMessage());
          }
      else {
         Device dev = items[0].getDevice();
             // continue process device
      }
   }
   catch (Exception e) {}
   }
}
```

See Also

- For detailed information on polling license information from a device, see Chapter 6.
- For detailed information on device discovery, see Chapter 4.

Chapter 6: Polling Licensing Information from a Device

- How do I get licensing information from a device?
- What licensable features are in a device?
- What licenses have been deployed for each feature?

Solution

In this chapter, the program CLMPollILicenseApp is developed to show how to poll licenses on a device and then read the license information from the Device object. To achieve this, your program will do the following:

- Log in to the Cisco License Manager server
- Poll license information
- Read license information
- Log out from the server

Poll Licenses

First, the program polls licenses on a device with UDI CISCO2821:FTX1201A29D.

The asynchronous call `asyncPollDeviceLicenseInfo` requests the server to perform a license poll. When the request is complete, the function onStatus() in the passed IDStatusListener object is called.

```
String jobid = lm_instance.asyncPollDeviceLicenseInfo(
   token, null,
   dev_ids, new IDStatusListener() {

      public void onStatus(String arg0, IDStatus arg1) {
         // TODO Auto-generated method stub
         if (arg1.getErrorCode() == ClmErrors.SUCCESS)
            System.out.println("Poll license success");
         else
            System.out.println("Poll license error: code="
               + arg1.getErrorCode());

         done = true;
      }
```

The done flag causes the program to continue with the read licenses calls. After the poll license call is completed, the device object in the server has the up-to-date information.

Read Licenses

The device object is retrieved with the device identifier or UDI. The readDevices() call gets the device status, which has the device object in the DeviceStatusItem[].

```
DeviceStatus deviceStatus = lm_instance.readDevices(token, dev_ids);
```

The device object retrieved with the device identifier or UDI has the FeatureInfo objects.

```
FeatureInfo[] infos = device.getFeatureInfo();
```

Each FeatureInfo has a feature name and the list of license lines for that feature. You retrieve the license line by using the readLicenseLine() API.

Sample Program

This sample program shows you how to read devices and run poll licenses from the devices.

```
import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;

import java.rmi.RemoteException;
import java.util.Calendar;
import java.util.LinkedList;

public class CLMPollLicenseApp {

    static protected LicenseManager lm_instance;

    static UserToken token = null;

    static String id_2821 = "CISCO2821:FTX1201A29D";

    static boolean done = false;
    // log in to CLM server
    public void initServer() {

        // Create an instance of the LicenseManager class.
        lm_instance = new LicenseManager();

        // Create an instance of a signed EULA agreement.
        EulaInfo eula = new EulaInfo(
Calendar.getInstance().getTime(), true,
                true);

        try {
            // Log in to CLM server
            token = lm_instance.login("admin",
"cisco", "localhost", 1099, 0,
                    eula);
```

```
            if (token != null) {
                System.out.println("Login successful!");
            } else {
                System.out.println("Login failed.");
            }
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    // log out from CLM server
    public void finiServer() {
        if (token != null && lm_instance != null) {
            try {
                lm_instance.logout(token);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }

    // get Device object to get license information
    public void readLicense() {

        Device device = null;

        try {
            String[] dev_ids = new String[1];
            dev_ids[0] = id_2821;


        // read device object
DeviceStatus deviceStatus =

lm_instance.readDevices(token,
dev_ids);
```

```
            if (deviceStatus != null
               && deviceStatus.getErrorCode() == ClmErrors.SUCCESS) {
DeviceStatusItem[] items = deviceStatus.getDeviceStatusItems();
                if (items != null && items.length == 1) {
                    if (items[0].getErrorCode() == ClmErrors.SUCCESS) {
                        // get to device object
                        device = items[0].getDevice();
                    }
                } else {
                    System.out.println("Error reading device");
                }
            } else {
                System.out.println("Error reading device");
            }

        } catch (RemoteException e) {
            System.out.println(e.toString());

            e.printStackTrace();
        }

        System.out.println();

        // print out licensable features for device
        if (device != null) {
            String[] features = device.getLicensableFeatures();
            if (features != null) {
                for (int i = 0; i < features.length; i++) {
                    System.out.println("Licensable Feature:" + features[i]);
                }
            }
        }
        System.out.println();

        // print out licenses for each feature
        if (device != null) {
            FeatureInfo[] infos = device.getFeatureInfo();
```

```
            if (infos != null) {
                for (int i = 0; i < infos.length; i++) {
                    System.out.println("Feature:" + infos[i].getFeatureName());

                    // each feature each zero or more license lines
LinkedList<LicenseLineInfo> llInfos =   infos[i]
.getLicLineInfoList();

                    String licLineId = null;
                    LicenseLineInfo llInfo = null;

                    for (int j = 0; j < llInfos.size(); j++) {
                        llInfo = (LicenseLineInfo) lInfos.get(j);

                        licLineId = llInfo.getLicLineId();
                            System.out.println("License Line ID:" + licLineId);
                        // use lm_instance.readLicenseLines to retrieve license
                        // line
                    }
                }
            }
        }

    // poll license for the device id_2821
    // create IDStatusListener to receive the callback
    public void pollLicense() {

        done = false;
        try {
            String[] dev_ids = new String[1];
            dev_ids[0] = id_2821;

     String jobid =         lm_instance.asyncPollDeviceLicenseInfo(
token, null,
                    dev_ids, new IDStatusListener() {

                public void onStatus(String arg0, IDStatus arg1) {
```

```
                // TODO Auto-generated method stub
                if (arg1.getErrorCode() == ClmErrors.SUCCESS)
                    System.out.println("Poll license success");
                else
                    System.out.println("Poll license error: code="
                            + arg1.getErrorCode());

                done = true;
            }

        });

    } catch (RemoteException e) {
        System.out.println(e.toString());

        e.printStackTrace();
    }
}

// main section of the program.
public static void main(String[] args) {
    CLMPollLicenseApp app = new CLMPollLicenseApp();
    app.initServer();
    app.pollLicense();
    // wait for poll license completion
    while (!done) {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    app.readLicense();
    app.finiServer();

    System.out.println("App ends");
```

```
    }

}
```

The sample output:

```
Login successful!
Poll license success

Licensable Feature:gatekeeper

Feature:VMIVR-VM-MBX
License Line ID:CISCO2821:FTX1201A29D-9tN6+axrYL8mnjYtiTBZXPdQNZA=
Feature:gatekeeper
License Line ID:CISCO2821:FTX1201A29D-TN/msf5Rw7Uy+0SVdQFQ/mhQB1Q=
License Line ID:CISCO2821:FTX1201A29D-zbRWqVW40+aYvNFTgkgDi8Tu3Cg=
License Line ID:CISCO2821:FTX1201A29D-PzkuWjJ3IfdvO3eCfDd+c3BYmRg=
App ends
```

Compile the Program

To compile the program, run this command line:

```
javac -cp <CLM_SDK_HOME>/clmsdk.jar:<CLM_SDK_HOME>/conf
   CLMPollLicenseApp.java
```

where <CLM_SDK_HOME> is the directory that holds the Cisco License Manager SDK files.

Run the Program

To run the program, make sure the Cisco License Manager server is running on computer “localhost,” and then run this command line:

```
java -cp <CLM_SDK_HOME>/clmsdk.jar:<CLM_SDK_HOME>/conf
   CLMPollLicenseApp
```

See Also

- For login functions to the Cisco License Manager server, see Chapter 3.
- For detailed descriptions of functions used in this program, see Chapter 16 of the *Java API Reference Guide for Cisco License Manager*.

Chapter 7: Adding a PAK to Inventory

- How do I create a new PAK?
- Download PAK information from Cisco.com
- What information is in a PAK?

Solution

In this chapter, you learn how to create a new PAK and download PAK information from Cisco.com.

Create a New PAK

First, make sure you have a valid PAK ID. In this example, a PAK with ID “FXPAKOADFC4” is created in the Default folder:

```
String[] pakIds = {“FXPAKOADFC4”};
PAKStatus pakSt = licMgr.createPAKs(token, pakIds, “Default”);
```

Create Callback

Because download PAK is an asynchronous call, the Cisco License Manager server immediately returns a request ID. Your client program must set up a callback function for the Cisco License Manager server to call upon completion of the download operation. You must implement the onStatus function in your callback function:

```
public class MyIDStatusListener implements IDStatusListener {
   public void onStatus(String requestId, IDStatus status) {
      idSt = status;
      idStatusListenerInvoked = true;
   }
}
```

Download PAK Information from Cisco.com

Your user profile should have the Cisco.com username and password set correctly to download the PAK from Cisco.com.

```
String[] intenalIds = new String[1];
// get PAK internal ID, this ID is used as a key to identify PAK
internalIds[0] = pakSt.getPAKStatusItems()[0].getPAK().getPakID()
// get Request ID from asynchronous call
String reqId = licMgr.asyncDownloadPAKInfo(token, null, internalIds, listener);
```

Sample Program

```
import java.rmi.RemoteException;
import com.cisco.nm.clm.common.*;
import com.cisco.nm.clm.sdk.LicenseManager;

public class DownloadPAKDemo {
   public static LicenseManager lic_mgr = new LicenseManager()
```

```
public class MyIDStatusListener implements IDStatusListener {
   private boolean idStatusListenerInvoked = false;
   private IDStatus idSt = null;

   public IDStatus getIDStatus() {
      return idSt;
   }

   public boolean isIDStatusListenerInvoked() {
      return idStatusListenerInvoked;
   }

   public void onStatus(String request_id, IDStatus status) {
      System.out.println("MyIDStatusListener for " +
         requestId + " invoked");
      idSt = status;
      idStatusListenerInvoked = true;
   }
}

public static void main(String[] args) {
   try {
      EulaInfo eula = new EulaInfo(null, true, true);
      UserToken token = licMgr.login(“admin”, “cisco”,
         "localhost”, 1099, 0, eula);

      DownloadPAKDemo demo =new DownloadPAKDemo();
      MyIDStatusListener listener =demo.new MyIDStatusListener();

      // Create PAK in Default folder
      String[] pakIds = {"FXPAK0ADFC4"};
      PAKStatus pakSt = licMgr.createPAKs(token, pakIds,
       “Default");

      if (pakSt.getErrorCode()!=ClmErrors.SUCCESS ||
pakSt.getPAKStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
         System.out.println("CreatePAK failed.");
```

```
            System.exit(0);
        }

        // Get PAK internal ID. This ID is used as a key to
        // identify PAK
        String[] intPakIds =
            {pakSt.getPAKStatusItems()[0].getPAK().getPakID()};

        // Download PAK from Cisco.com
        String reqId = licMgr.asyncDownloadPAKInfo(token,
            null, intPakIds, listener);

        // Wait until job is done
        while(!my_listener.isIDStatusListenerInvoked()){
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // Retrieve job status
        IDStatus idSt = listener.getIDStatus();
        if (idSt==null ||
            idSt.getErrorCode()!=ClmErrors.SUCCES ||
idSt.getIDStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {

            System.out.println("failed to downloadPAK...");
            if (idSt!=null)
                            System.out.println(idSt.getErrorMessage());
        } else {
            System.out.println("downloadPAK successfully
              completes.");
        }
        licMgr.logout(token);
        System.out.println("return from downloadPAK demo");
        System.exit(0);
```

```
      } catch (RemoteException e) {
         e.printStackTrace();
         System.exit(-1);
      }
   }
}
```

See Also

- For getting started information, see Chapters 1, 2, and 3.

Chapter 8: Obtaining Licenses

- What are SKU and quantity in a PAK?

  SKU stands for Stock Keeping Unit; software on Cisco.com is a “SKU.” A SKU maps to one or more license features.

- How do I obtain licenses for a device?

  You must retrieve SKU information from Cisco License Manager Inventory, construct a LicenseRequest object by using the SKU and the device, and call the asyncObtainLicense() API to obtain a license for a device.

Solution

In this chapter, you create a client program using the Cisco License Manager SDK to obtain licenses from the Cisco License Registration Server. The program, called ObtainLicenseDemo.java, connects and logs you in to the Cisco License Manager server, constructs LicenseRequest, and calls the asyncObtainLicense() API to obtain new licenses.

Write a Listener to Implement the IDStatusListener Callback Interface

The asyncObtainLicense() function runs asynchronously, which means that when this function is called, it runs on the server side and immediately returns a request ID. Therefore, before calling the asyncObtainLicense() function, you must set up a callback function for the Cisco License Manager server to call upon when asyncObtainLicense() completes.

```
public class MyIDStatusListener implements  IDStatusListener {
   IDStatus idSt = null;
   boolean idStatusListenerInvoked = false;
   public void onStatus(String request_id, IDStatus status) {
      m_logger.debug("MyIDStatusListener for " + request_id + " invoked");idSt=status;
   idStatusListenerInvoked = true;
   }

   public IDStatus getIDStatus() {
      return idSt;
   }
   public boolean isIDStatusListenerInvoked() {
      return idStatusListenerInvoked;
   }
}
```

Create an Instance of the LicenseManager Class

```
static LicenseManager lm_instance = new LicenseManager();
```

Log In to the Cisco License Manager Server

```
// Create an instance of signed EULA agreement.
Calendar cldr = Calendar.getInstance();
```

```
EulaInfo eula = new EulaInfo(cldr.getTime(), true, true);

// Log in to CLM server
UserToken token = lm_instance.login(m_username, m_password,
   m_server_host, 1099, 0, eula);
```

Read PAK from Cisco License Manager Server and Construct LicenseRequest

```
String[] int_pak_ids = {"D8D6B26BA7AD08E7071460F687E07E26"};
//this must be the internal ID of PAK, not displayName.
PAKStatus pakSt = lm_instance.readPAKs(token, int_pak_ids);
if (pakSt.getErrorCode()!=ClmErrors.SUCCESS ||
   pakSt.getPAKStatusItems()[0].getErrorCode()!=
      ClmErrors.SUCCESS) {
      System.out.println("readPAK failed.");
      System.exit(0);
}

PAK pak = pakSt.getPAKStatusItems()[0].getPAK();
SKU[] skuArr = pak.getSkuList();
LicenseRequest licReq = new LicenseRequest();
licReq.setDeviceID("C3800-MPE:FXNTEWQQ");
licReq.setSKUSelection(skuArr);
licReq.setPAKType(pak.getPakType());
LicenseRequest[] licReqs = {licReq};
```

Send License Upgrade Request to License Registration Server

This program creates “my_listener” so that the client can be notified when the job created by asyncObtainLicense() completes.

```
MyIDStatusListener my_listener = new MyIDStatusListener();
String req_id = lm_instance.asyncObtainLicense (token, null,
   LicReqs, false, my_listener);
//wait until job is done
while(!my_listener.isIDStatusListenerInvoked()){
   try {
      Thread.sleep(5000);
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}
```

```
//retrieve job status
IDStatus idSt = my_listener.getIDStatus();
if(idSt==null||dSt.getErrorCode()!=ClmErrors.SUCCESS||
   idSt.getIDStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
   System.out.println("failed to obtain licenses, error....");
   if (idSt!=null)
      System.out.println(idSt.toString());
} else {
      System.out.println("obtainLicenses successfully completes.");
}
```

Log Out from the Cisco License Manager Server

You should always log out from the Cisco License Manager server at the end of your program so that the resource for the connection can be released.

```
lm_instance.logout(token);
```

Sample Program

```
import java.rmi.RemoteException;
import java.util.Calendar;
import com.cisco.nm.clm.common.ClmErrors;
import com.cisco.nm.clm.common.EulaInfo;
import com.cisco.nm.clm.common.IDStatus;
import com.cisco.nm.clm.common.IDStatusListener;
import com.cisco.nm.clm.common.LicenseRequest;
import com.cisco.nm.clm.common.PAK;
import com.cisco.nm.clm.common.PAKStatus;
import com.cisco.nm.clm.common.SKU;
import com.cisco.nm.clm.common.UserToken;
import com.cisco.nm.clm.sdk.LicenseManager;

public class ObtainLicenseDemo {
static final protected String m_server_host = "localhost";
   static final protected String m_username = "admin";
   static final protected String m_password = "cisco";
   final static String USER_NAME = "admin";

   public class MyIDStatusListener implements IDStatusListener {
      private boolean idStatusListenerInvoked = false;
```

```
      private IDStatus idSt = null;

      public IDStatus getIDStatus() {
         return idSt;
      }

      public boolean isIDStatusListenerInvoked() {
         return idStatusListenerInvoked;
      }

      public void onStatus(String request_id,
IDStatus status) {
         System.out.println("MyIDStatusListener for " + request_id + " invoked");
         idSt = status;
         idStatusListenerInvoked = true;
      }
   }

   public static void main(String[] args) {
      LicenseManager lm_instance = new icenseManager(88888);
      Calendar cldr = Calendar.getInstance();
      EulaInfo eula = new EulaInfo(cldr.getTime(), true, true);
      try {
         UserToken token = lm_instance.login(m_username, m_password, m_server_host,
1099, 0, eula);
         ObtainLicenseDemo demo = new ObtainLicenseDemo();
         MyIDStatusListener my_listener = demo.new MyIDStatusListener();

   //this must be the internal ID of PAK, not displayName.
         String[] int_pak_ids = {"D8D6B26BA7AD08E7071460F687E07E26"};
PAKStatus pakSt = lm_instance.readPAKs(token, int_pak_ids);
if (pakSt.getErrorCode()!=ClmErrors.SUCCESS
||pakSt.getPAKStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
            System.out.println("readPAK failed.");
            System.exit(0);
         }
         PAK pak =  pakSt.getPAKStatusItems()[0].getPAK();
```

```
         SKU[] skuArr = pak.getSkuList();
         LicenseRequest licReq = new LicenseRequest();
         licReq.setDeviceID("C3800-MPE:FXNTEWQQ");
         licReq.setSKUSelection(skuArr);
         licReq.setPAKType(pak.getPakType());
         LicenseRequest[] licReqs = {licReq};
         String requestId =            lm_instance.asyncObtainLicense(token, null,
licReqs, false, my_listener);
         //wait until job is done
         while( !my_listener.isIDStatusListenerInvoked()){
            try {
               Thread.sleep(5000);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
         //retrieve job status
         IDStatus idSt = my_listener.getIDStatus();
         if(idSt==null || idSt.getErrorCode()!=ClmErrors.SUCCESS ||
idSt.getIDStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
            System.out.println("failed to obtain licenses, error....");
if (idSt!=null)
System.out.println(idSt.toString());
         } else {
         System.out.println("obtainLicenses successfully completes.");
         }
         lm_instance.logout(token);
         System.out.println("return from obtain licenses demo");
         System.exit(0);
      } catch (RemoteException e) {
         e.printStackTrace();
         System.exit(-1);
      }
   }
 }
```

See Also

- For login functions to the Cisco License Manager server, see Chapter 3.

- For detailed descriptions of functions used in this program, see Chapter 16 of the *Java API Reference Guide for Cisco License Manager*.

Chapter 9: Deploying Licenses

- Where are my undeployed licenses?

  When a license is marked as undeployed in Cisco License Manager Inventory, the license is still kept in Cisco License Manager but is cleared from the device. The license might still be valid, binding with that device.

- How do I deploy licenses to devices?

  You can use the Cisco License Manager asyncDeployLicenses() API to deploy the license to the device if it is not read-only.

Solution

In this chapter, you create a client program using the Cisco License Manager SDK to deploy licenses to the device. The program, called DeployLicenseDemo.java, connects and logs you in to the Cisco License Manager server, and calls the asyncDeployLicenses() API to deploy licenses to the device.

Write a Listener to Implement the IDStatusListener Callback Interface

The asyncDeployLicenses() function runs asynchronously, which means that when this function is called, it runs on the server side and immediately returns a request ID. Therefore, before calling the asyncDeployLicenses() function, you must set up a callback function for the Cisco License Manager server to call upon when asyncDeployLicenses() completes.

```
public class MyIDStatusListener implements  IDStatusListener {
   IDStatus idSt = null;
   boolean idStatusListenerInvoked = false;
   public void onStatus(String request_id, IDStatus status) {
      m_logger.debug("MyIDStatusListener for " + request_id + " invoked");
      idSt=status;
      IdStatusListenerInvoked = true;
   }

   public IDStatus getIDStatus() {
      return idSt;
   }
   public boolean isIDStatusListenerInvoked() {
      return idStatusListenerInvoked;
   }
}
```

Create an Instance of the LicenseManager Class

```
static LicenseManager lm_instance = new LicenseManager();
```

Log In to the Cisco License Manager Server

```
// Create an instance of signed EULA agreement.
EulaInfo eula = new EulaInfo(Calendar.getInstance().getTime(), true, true);
// Log in to CLM server
UserToken token = lm_instance.login("admin",          "cisco", "johndoe-server", 1099,
0, eula);
```

Deploy Licenses to a Device

```
String[] lic_ids = { "B8218233A15832420EFBAA1A8E912BA4-::-L-SL-39-UC-K9=-::-C3900-
SPE150/K9:FHH123000J4"};
String requestId = lm_instance.asyncDeployLicenses(token, null, lic_ids, my_listener);
//wait until job is done
while(!my_listener.isIDStatusListenerInvoked()){
   try {
      Thread.sleep(5000);
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}

//retrieve job status
IDStatus idSt = my_listener.getIDStatus();
if(idSt==null || idSt.getErrorCode()!=ClmErrors.SUCCESS||
idSt.getIDStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
   System.out.println("failed to obtain licenses, error....");
if (idSt!=null)
System.out.println(idSt.toString());
} else {
   System.out.println("obtainLicenses successfully completes.");
}
```

Log Out from the Cisco License Manager Server

You should always log out from Cisco License Manager server at the end of your program so that the resource for the connection can be released.

```
lm_instance.logout(token);
```

Sample Program

```
import java.rmi.RemoteException;
import java.util.Calendar;
import com.cisco.nm.clm.common.ClmErrors;
```

```
import com.cisco.nm.clm.common.EulaInfo;
import com.cisco.nm.clm.common.IDStatus;
import com.cisco.nm.clm.common.IDStatusListener;
import com.cisco.nm.clm.common.UserToken;
import com.cisco.nm.clm.sdk.LicenseManager;

public class DeployLicensesDemo {
   static final protected String m_server_host = "localhost";
   static final protected String m_username = "admin";
   static final protected String m_password = "cisco";
   final static String USER_NAME = "admin";

   public class MyIDStatusListener implements IDStatusListener {
      private boolean idStatusListenerInvoked = false;
      private IDStatus idSt = null;

      public IDStatus getIDStatus() {
         return idSt;
      }

      public boolean isIDStatusListenerInvoked() {
         return idStatusListenerInvoked;
      }

      public void onStatus(String request_id, IDStatus status)
{
         System.out.println("MyIDStatusListener for " + request_id + " invoked");
         idSt = status;
         idStatusListenerInvoked = true;
      }
   }

   public static void main(String[] args) {
      LicenseManager lm_instance = new  LicenseManager(88888);
      Calendar cldr = Calendar.getInstance();
      EulaInfo eula = new EulaInfo(cldr.getTime(), true, true);
      try {
```

```
         UserToken token = lm_instance.login(m_username, m_password, m_server_host,
1099, 0, eula);
         DeployLicensesDemo demo =new DeployLicensesDemo();
         MyIDStatusListener my_listener = demo.new MyIDStatusListener();

         String[] lic_ids = {"B8218233A15832420EFBAA1A8E912BA4-::-L-SL-39-UC-K9=-::-
C3900-SPE150/K9:FHH123000J4"};
         String requestId = lm_instance.asyncDeployLicenses(token, null, lic_ids,
my_listener);
         //wait until job is done
                                    while( !my_listener.isIDStatusListenerInvoked()){
            try {
               Thread.sleep(5000);
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
         }

         //retrieve job status
         IDStatus idSt = my_listener.getIDStatus();
         if(idSt==null || idSt.getErrorCode()!=ClmErrors.SUCCESS||
idSt.getIDStatusItems()[0].getErrorCode()!=ClmErrors.SUCCESS) {
            System.out.println("failed to deploy licenses, error....");
            if (idSt!=null)

System.out.println(idSt.toString());
         } else {
            System.out.println("deployLicense successfully completes.");
         }
         lm_instance.logout(token);
         System.out.println("return from deploy licenses demo");
         System.exit(0);
      } catch (RemoteException e) {
         e.printStackTrace();
         System.exit(-1);
      }
   }
```

```
}
```

See Also

- For login functions to the Cisco License Manager server, see Chapter 3.
- For detailed descriptions of functions used in this program, see Chapter 16 of the *Java API Reference Guide for Cisco License Manager*.

Chapter 10: Listening to Device Notifications

- How do I register a listener?
- What do I do with the incoming notifications from devices?

Solution

In this chapter, the program CLMRegisterNotificationApp is developed to show how to listen for device notifications. To achieve this, your program will do the following:

- Log in to the Cisco License Manager server
- Register for notifications
- Log out from the Cisco License Manager server

Register for Notifications

Use the call `registerNotificationListener` to register a listener for all notifications. Passed in the listener is a NotificationListener object. NotificationListener.onNotification() is invoked when a notification event arrives. The Notification object contains the detailed information.

```
boolean rc = m_instance.registerNotificationListener(token,
                    new NotificationListener() {
              public void onNotification(Notification arg0) {
             …
              }

         });

                public void onNotification(Notification arg0) {
                    // TODO Auto-generated method stub

                    Notification notification = arg0;
Notification.Operation op =             notification.getOperation();
                    String devId = null;
                    String licLineId = null;

System.out.println("Notification, type is: " +    op.toString());

                    if (op.equals(Notification.Operation.EXPIRED)) {
```

```
                devId = notification.getDeviceId();
                licLineId = notification.getLiclineId();

                if (devId != null)
                    System.out.println("Device id=" + devId);
                System.out.println("Expired notification: ");

                if (licLineId != null)
              System.out.println("License line id=" +            licLineId);

              } else if     (op.equals(Notification.Operation.INSTALL)) {
                devId = notification.getDeviceId();
                licLineId = notification.getLiclineId();

                if (devId != null)
                    System.out.println("Device id=" + devId);
                System.out.println("Deployed or installed notification: ");

                if (licLineId != null)
                    System.out.println("License line id=" + licLineId);
            }
  }
```

The example NotificationListener.onNotification() looks for the license expired and license install notifications, and prints the device UDI and the license line ID associated with the notifications.

Sample Program

```
import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;

import java.rmi.RemoteException;
import java.util.Calendar;

public class CLMDeviceNotificationApp {

    static protected LicenseManager lm_instance;

    static UserToken token = null;
```

```
static boolean done = false;

public void initServer() {

    // Create an instance of LicenseManager class.
    lm_instance = new LicenseManager();

    // Create an instance of signed EULA agreement.
    EulaInfo eula = new EulaInfo(
        Calendar.getInstance().getTime(), true,
            true);

    try {
        // Log in to CLM server
        token = lm_instance.login("admin", "cisco", "localhost", 1099, 0,
                eula);

        if (token != null) {
            System.out.println("Login successful!");
        } else {
            System.out.println("Login failed.");
        }
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}

public void finiServer() {
    if (token != null && lm_instance != null) {
        try {
            lm_instance.logout(token);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

```
    public void registerNotification() {

        try {
            boolean rc = lm_instance.registerNotificationListener(
                token,
                    new NotificationListener() {

                public void onNotification(Notification arg0) {
                    // TODO Auto-generated method stub

                    Notification notif = arg0;
       Notification.Operation op =           notif.getOperation();
                    String devId = null;
                    String licLineId = null;

                    System.out.println(
"Notification, type is: " + op.toString());

                    if (op.equals(Notification.Operation.EXPIRED)) {

                        devId = notif.getDeviceId();
                        licLineId = notif.getLiclineId();

                        if (devId != null)
                            System.out.println("Device id=" + devId);
                        System.out.println("Expired notification: ");

                        if (licLineId != null)
                            System.out.println("License line id=" + licLineId);

                    } else if (op.equals(Notification.Operation.INSTALL)) {
                        devId = notif.getDeviceId();
                        licLineId = notif.getLiclineId();

                        if (devId != null)
```

```
                            System.out.println("Device id=" + devId);
                        System.out.println("Deployed or installed notification: ");

                        if (licLineId != null)
                            System.out.println("License line id=" + licLineId);
                    }
                }

            });

        } catch (RemoteException e) {
            System.out.println(e.toString());

            e.printStackTrace();
        }
    }

    // main of the program.
    public static void main(String[] args) {
        CLMDeviceNotificationApp app = new CLMDeviceNotificationApp();
        app.initServer();
        app.registerNotification();
    // wait forever until user terminates
        while (!done) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        app.finiServer();

        System.out.println("App ends");
    }
}
```

See Also

- For login functions to the Cisco License Manager server, see Chapter 3.
- For detailed descriptions of functions used in this program, see Chapter 16 of the *Java API Reference Guide for Cisco License Manager*.

Chapter 11: Transferring Licenses

- How do I transfer (rehost) licenses from one device to another?
- What are permission tickets and rehost tickets?
- What licenses can be transferred?

Solution

In this chapter, you learn how to transfer licenses from one device to another by using the Cisco License Manager Java API. To transfer a license from a source device, you need a permission ticket from Cisco.com and a rehost ticket from the source device. A permission ticket is a certificate from the Cisco licensing back-end server to start a license transfer process. A rehost ticket is a certificate generated by the router for the Cisco licensing back-end server to confirm the revoke operation.

License transfer involves four major steps:

1. Getting the permission ticket from Cisco.com for the source device
2. Retrieving the rehost ticket from the source device
3. Sending the rehost ticket to Cisco.com to obtain licenses for the destination device
4. Deploying the newly obtained licenses to the destination device

Currently, you can transfer only permanent licenses.

Cisco License Manager provides APIs for transferring licenses by using a step-by-step approach as well as an atomic approach by using a single API. For the step-by-step approach, use the following four APIs in sequence:

- `public Status initRehostLicense(UserToken token, RehostRequest rehostReq)`
- `public Status revokeLicenseForRehost(UserToken token, RehostRequest rehostReq)`
- `public Status obtainLicenseForRehost(UserToken token, RehostRequest rehostReq)`
- `public String asyncDeployLicenseLines(UserToken token, String jobGroup, String[] licLineIDs, IDStatusListener listener)`

Cisco License Manager also provides a single API that encapsulates all the license transfer steps into one atomic operation:

- `public Status rehostLicense(UserToken token, RehostRequest rehostReq)`

In this chapter we will focus on transferring the licenses using the single API.

Create an Instance of the LicenseManager Class

```
static LicenseManager lmInstance = new LicenseManager(1099);
```

Log In to the Cisco License Manager Server

```
// Create an instance of signed EULA agreement.
EulaInfo eula = new EulaInfo( true, true);
//log in to CLM server
            UserToken token = lmInstance.login(username,   password, serverHost, 1099,
0, eula);
```

Retrieve Source and Destination Device IDs

If the source and destination devices already exist in Cisco License Manager Inventory, use listAllDevices or listAllDevicesInGroup to get the source and destination device ID.

```
//specify the paging information
Pagination pageInfo = new Pagination(0,-1);
//Get a list of all the devices
DevicePagingInfo devPagingInfo = lmInstance.listAllDevices(token,pageInfo);
//Retrieve the device IDs
Device[] devs = devPagingInfo.getDevices();
            String srcDeviceID = devs[0].getDeviceID();
            String destDeviceID = devs[1].getDeviceID();
```

Retrieve Rehostable SKUs for the Source Device

Get the list of SKUs and features on the source device that can be transferred to another device from the Cisco back-end licensing server.

```
RehostableSKUStatus rehostSKUStatus = lmInstance.getRehostableSKUsByDevice
(token,srcDeviceID);
if(rehostSKUStatus.getErrorCode() == 0 &&
    rehostSKUStatus.getRehostableSKUs() != null &&
    rehostSKUStatus.getRehostableSKUs()[0] != null{
       //select the first SKU to transfer
   RehostableSKU[] skus =
{rehostSKUStatus.getRehostableSKUs()[0]};


}
```

Construct a Rehost Request

Construct the rehost request specifying the source device ID, the destination device ID, and the SKUs that you want to transfer from the source device to the destination device.

```
//Construct the RehostRequest, containing the source and destination device IDs and
the SKU selection
            RehostRequest rr = new RehostRequest();
                  //set the destination device ID
   rr.setDestDeviceID(destDeviceID);
//set the source device ID
rr.setSourceDeviceID(srcDeviceID);
//set the SKUs to be transferred
rr.setSKUSelection(skus);
```

Rehost a License

Transfer the license. After the successful license transfer, the license is deployed to the destination device.

```
 Status rehostStatus = lmInstance.rehostLicense(token, rr);
 if(rehostStatus != null && rehostStatus.getErrorCode() == ClmErrors.SUCCESS){
        System.out.println("rehost successful");
                                }else{
   System.out.println(“rehost failed with message: “+rehostStatus.getErrorMessage());
}
```

Log Out from the Cisco License Manager Server

You should always log out from Cisco License Manager server at the end of your program so that the resource for the connection can be released.

```
    lmInstance.logout(token);
```

Sample Program

Here is the complete Java program TransferLicenseDemo.java:

```
import java.rmi.RemoteException;


import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;


public class TransferLicenseDemo {
   static final protected String serverHost = "localhost";
   static final protected String username = "admin";
   static final protected String password = "cisco";
   static protected LicenseManager lmInstance;


   /**
   * this function transfers the licenses from one device to
   * another device
   */
   public void transferLicenses(){
   LicenseManager lmInstance =new LicenseManager(1099);
   EulaInfo eula = new EulaInfo( true, true);
   try {
         UserToken token = lmInstance.login(username, password, serverHost, 1099, 0,
eula);
             //specify the paging information
         Pagination pageinfo = new Pagination(0,-1);
```

```
          DevicePagingInfo pageInfo;
          String srcDeviceID ;
          String destDeviceID ;

          //retrieve the source and destination device ID
         pageInfo = lmInstance.listAllDevices(token, pageinfo);
         Status status = pageInfo.getStatus();

         if(status.getErrorCode() == 0){
       Device[] devs = pageInfo.getDevices();
       srcDeviceID = devs[0].getDeviceID();
       destDeviceID = devs[1].getDeviceID();
       //get the SKUs that can be transferred
       RehostableSKUStatus rehostSKUStatus= lmInstance.getRehostableSKUsByDevice(token,
srcDeviceID);

       if(rehostSKUStatus.getErrorCode() == 0 &&
rehostSKUStatus.getRehostableSKUs() != null &&
rehostSKUStatus.getRehostableSKUs()[0] != null){
          //transfer only the first SKU
          RehostableSKU[] skus = {rehostSKUStatus.getRehostableSKUs()[0]};
          //Construct the RehostRequest,
                   // containing the source and destination device
                   // IDs and the SKU selection
                   RehostRequest rr = new RehostRequest();
          rr.setDestDeviceID(destDeviceID);
          rr.setSourceDeviceID(srcDeviceID);
          rr.setSKUSelection(skus);
          //rehost license
          Status rehostStatus = lmInstance.rehostLicense(token, rr);

          //check the rehost status
          if(rehostStatus != null && rehostStatus.getErrorCode() == ClmErrors.SUCCESS){
                 System.out.println("rehost successful");
          }
                   else{
             System.out.println("rehost failed with message:
"+rehostStatus.getErrorMessage());
```

```
        }
        }else{
      System.out.println("unable to retrieve SKUs, error message:
"+rehostSKUStatus.getErrorMessage());
       }
   }else{
      System.out.println("Error retrieving source/destination device IDs");
   }
   //log out
   lmInstance.logout(token);
   System.exit(0);
    }catch(RemoteException re){
   re.printStackTrace();
   System.exit(-1);
     }
   }

   public static void main(String[] args) {

   TransferLicenseDemo demo =new TransferLicenseDemo();
   demo.transferLicenses();
   }
  }
```

See Also

- For login functions to the Cisco License Manager server, see Chapter 3.
- For detailed descriptions of functions used in this program, see Chapter 7 of the *Java API Reference Guide for Cisco License Manager.*

Chapter 12: Policy Licensing

- What is rule-based licensing using a policy?
- How do I create a policy?
- How do I filter devices?
- How do I execute a policy?

Solution

In this chapter, you learn about rule-based licensing using a policy. You write a client program to create a policy by using the Cisco License Manager SDK. You also learn how to filter devices with the policy you created as well as how to run the policy by using the Cisco License Manager SDK.

The Cisco License Manager SDK supports rule-based policies. Policies are used to get licenses and deploy licenses to devices based on user-specified criteria (or rules). Policies provide an automated process that scans your network and adds features to your specified devices. A policy contains two rules:

- SKU rule that is used to select SKUs of interest
- Device rule that is used to select devices of interest

You can specify the following criteria to select SKUs of interest:

- PAK
- SKU
- Feature name

You can specify the following criteria to select devices of interest:

- Device group
- Device model
- Device IP address range

To create a policy, you will do the following in your program:

Create a Policy

First, you create a policy (for example, “policy_1”):

```
PolicyStatus status = lmInstance.createPolicy(token,
   policy_1", null, null);
```

Set the SKU Rule

To set the SKU rule, you get the policy object you just created:

```
Policy policy = status.getPolicy();
```

Next, create a SKUFilter object with the desired feature name (for example, “ipservices”) and set the SKUFilter:

```
SKUFilter skuFilter = new SKUFilter(“ipservices”);
policy.setSKUFilter(skuFilter);
```

You can find out all the available features in the Cisco License Manager database using the enumerateSKUFilterAttribute API:

```
String[] features = lmInstance.enumerateSKUFilterAttribute(
   token, Policy.SKUAttribute.FEATURE_NAME);
```

After creating the SKUFilter object, you set SKUIdentifier, which contains the desired PAK and SKU information:

```
SKUIdentifier[] identifiers = lmInstance.listFilteredSKUs(
   token, skuFilter);
policy.setSKUsInUse(identifiers);
```

Set a Device Rule

You can set all, some, or none of the device filter criteria. You can do the following in your program to set a device rule:

```
DeviceFilter deviceFilter = new DeviceFilter();
deviceFilter.setModel(“3845”);
deviceFilter.setGroup(“Default”);
deviceFilter.setIPLowerLimit(“10.1.2.10”);
deviceFilter.setIPUpperLimit(“10.1.2.20”);
policy.setDeviceFilter(deviceFilter);
```

Write the Policy

After setting up the SKU and device rules, you write the policy to the Cisco License Manager database in your program:

```
Status status = lmInstance.writePolicy(token, policy);
```

To execute a policy, you use the asyncExecutePolicy API in your program. The asyncExecutePolicy API is an asynchronous function, and you must provide a listener to find the status of this operation. The callback function of this listener is called when the policy execution is completed. The following is what you will do:

```
String jobId = lmInstance.asyncExecutePolicy(token,
   null, policy.getID(), listener);
```

Sample Program

Here is the sample client program for creating, filtering, and executing a policy:

```
import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;
import java.rmi.RemoteException;
import java.util.Calendar;


public class CLMPolicyTest {


   static protected LicenseManager lmInstance;
   static protected UserToken token;
```

```
static protected Policy policy;

public CLMPolicyTest() {
}



/**
 * Initialize and log in to LicenseManager
 */
public void init() {

   lmInstance = new LicenseManager();

   EulaInfo eula = new EulaInfo(
      Calendar.getInstance().getTime(), true, true);

   try {
      token = lmInstance.login("admin", cisco",
                     "localhost", 1099, 0, eula);
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}



/**
 * Log out from LicenseManager
 */
public void close() {
   try {
      lmInstance.logout(token);
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}
```

```
/**
 * Create a policy
 */
public void createPolicy() {
   try {
      PolicyStatus status = lmInstance.createPolicy(token,
                        "policy_1", null, null);

      if (status == null || status.getErrorCode()
                         != ClmErrors.SUCCESS) {
         System.out.println("Create policy failed!");
      } else {
         System.out.println("Create policy succeeded!");

         policy = status.getPolicy();
      }
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
 * Set SKU rule
 */
public void setSkuRule() {
   try {
      SKUFilter skuFilter = new SKUFilter("gatekeeper");
      policy.setSKUFilter(skuFilter);

      SKUIdentifier[] allIdentifiers =
         lmInstance.listFilteredSKUs(token, skuFilter);

      SKUIdentifier[] identifierToUse = new SKUIdentifier[1];
      identifierToUse[0] = allIdentifiers[0];
      policy.setSKUsInUse(identifierToUse);
   } catch (RemoteException e) {
```

```
      e.printStackTrace();
   }
}


/**
 * Set Device rule
 */
public void setDeviceRule() {
   DeviceFilter deviceFilter = new DeviceFilter();
   deviceFilter.setModel("2821");
   deviceFilter.setIPLowerLimit("172.19.205.70");
   deviceFilter.setIPUpperLimit("172.19.205.80");
   policy.setDeviceFilter(deviceFilter);
}


/**
 * Write the policy
 */
public void writePolicy() {
   try {
      Status status = lmInstance.writePolicy(token, policy);

      if (status == null || status.getErrorCode()
                        != ClmErrors.SUCCESS) {
         System.out.println("Write policy failed!");
      } else {
         System.out.println("Write policy succeeded!");
      }
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
```

```
 * Run a policy
 */
public void executePolicy() {
   try {
      MyIDStatusListener listener = new MyIDStatusListener();
      String requestId = lmInstance.asyncExecutePolicy(
                        token, null, policy.getID(), listener);

      System.out.println("Execute policy started,
                        the request ID is "+requestId);
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
 * @param args
 */
public static void main(String[] args) {
   CLMPolicyTest testApp = new CLMPolicyTest();
   testApp.init();

   testApp.createPolicy();

   testApp.setSkuRule();
   testApp.setDeviceRule();
   testApp.writePolicy();

   testApp.executePolicy();

   testApp.close();
}


    public class MyIDStatusListener implements
       IDStatusListener {
```

```
            public void onStatus(String requestId, IDStatus status) {
                        System.out.println("MyIDStatusListener for
                           request "+requestId+" callback invoked");
                System.out.println("Policy executed,
                           returned error code "+status.getErrorCode());
        }
                    }

}
```

See Also

- For detailed descriptions of functions used in this program, see Chapter 12 of the Java API Reference Guide for Cisco License Manager.
- For login functions to the Cisco License Manager server, see Chapter 3.

Chapter 13: Generating Reports

- What reports are supported?
- How do I generate a report?
- How do I read a report?
- How do I filter a report?

Solution

In this chapter, you learn what types of reports Cisco License Manager provides. You write a client program to generate and read reports by using the Cisco License Manager SDK. Furthermore, you learn how to filter a large report to find the desired information by using the Cisco License Manager SDK.

Cisco License Manager supports these types of reports:

- Audit Trail Report
- Device Summary Report
- Hardware End of Life and End of Sale Report
- License Discrepancy Report
- License Expiry Report
- Newly Discovered Device Report
- PSIRT Summary Report
- PSIRT Details Report
- Redeployable License Report
- RMA Discrepancy Report
- Software End of Life and End of Sale Report
- Undeployed License Report

Generate a Report

You need to generate a report before reading and viewing the report. To generate a report, you specify which type of report you are interested in by specifying the ReportSubject. This is what you do in your program:

```
Status status = lmInstance.generateReport(token,
   ReportSubject.DEVICE_SUMMARY, null);
```

Read a Report

Reports are generated on the Cisco License Manager server in HTML format. You use the readReport API to find the URL of the generated report. You can then display the report in a browser or use it in other applications. To read a report, this is what you do in your program:

```
ReportStatus status = lmInstance.readReport(token,
   ReportSubject.DEVICE_SUMMARY,
   OutputFormat.HTML);


String reportURL = status.getContent();
```

Report Filters

You can filter a report to trim down its content if the report is too large. You can use these criteria to find matching devices for a report filter:

- Device model
- Device group
- Device IP address range
- Device license capability
- Software version on device
- Type of license on device
- Numbers of days to license expiration
- PAK name
- Feature name

To find the possible values for the device model or the device group, use the enumerateDeviceFilterAttribute API as follows in your code:

```
String[] values = lmInstance.enumerateDeviceFilterAttribute(
   token, Policy.DeviceAttribute.MODEL);


String[] values = lmInstance.enumerateDeviceFilterAttribute(
   token, Policy.DeviceAttribute.GROUP);
```

Possible values for device license capability are: Annotate, Deploy, Rehost, and Resend.

Possible values for the type of license on the device are: Evaluation, Extension, Permanent, Evaluation Subscription, Extension Subscription, Paid Subscription, Trial, and Unknown.

To find the possible values for PAK name, use the listAllPAKs API as follows:

```
PAKPagingInfo pagingInfo = lmInstance.listAllPAKs(token,
   new Pagination(0, -1));


PAK[] paks = pagingInfo.getPAKs();


ArrayList<String> values = new ArrayList<String>();


for (PAK pak : paks) {
   values.add(pak.getPakDisplayName());
}
```

To find the possible values for feature name, use the enumerateSKUFilterAttribute API as follows:

```
String[] values = lmInstance.enumerateSKUFilterAttribute(
    token, Policy.SKUAttribute.FEATURE_NAME);
```

Generate a Filtered Report

The following code example filters the Device Summary Report to show all the devices in group “group_1”:

```
DeviceFilter filter = new DeviceFilter();

filter.setGroup(“group_1”);

Status status = lmInstance.generateReport(token,
   ReportSubject.DEVICE_SUMMARY, filter);
```

Sample Program

Here is the sample client program for generating, filtering, and reading a report:

```
import com.cisco.nm.clm.sdk.*;
import com.cisco.nm.clm.common.*;
import
   com.cisco.nm.clm.common.LicenseManagerInterface.OutputFormat;
import
   com.cisco.nm.clm.common.LicenseManagerInterface.ReportSubject;
import java.rmi.RemoteException;
import java.util.Calendar;

public class CLMReportTest {

   static protected LicenseManager lmInstance;
   static protected UserToken token;


   public CLMReportTest() {
   }


   /**
    * Initialize and log in to LicenseManager
    */
   public void init() {

      lmInstance = new LicenseManager();

      EulaInfo eula = new EulaInfo(
```

```
      Calendar.getInstance().getTime(), true, true);

   try {
      token = lmInstance.login("admin", "cisco",
                     "localhost", 1099, 0, eula);
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
 * Log out from LicenseManager
 */
public void close() {
   try {
      lmInstance.logout(token);
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
 * Generate a report
 */
public void generateReport() {
   try {
      ReportSubject subject = ReportSubject.DEVICE_SUMMARY;
      DeviceFilter filter = null;
      Status status = lmInstance.generateReport(token,
                     subject, filter);

      if (status == null || status.getErrorCode()
                      != ClmErrors.SUCCESS) {
         System.out.println("Generate report failed!");
      } else {
```

```
            System.out.println("Generate report succeeded!");
         }
      } catch (RemoteException e) {
         e.printStackTrace();
      }
   }

   /**
    * Generate a filtered report
    */
   public void generateFilteredReport() {
      try {
         ReportSubject subject = ReportSubject.DEVICE_SUMMARY;

         DeviceFilter filter = new DeviceFilter();
         filter.setGroup("Group_1");
         filter.setDaysToExpire(30);

         String[] values =
            lmInstance.enumerateDeviceFilterAttribute(
                              token, Policy.DeviceAttribute.MODEL);
         filter.setModel(values[0]);

         Status status = lmInstance.generateReport(token,
            subject, filter);

         if (status == null || status.getErrorCode()
                           != ClmErrors.SUCCESS) {
            System.out.println("Generate report failed!");
         } else {
            System.out.println("Generate report succeeded!");
         }
      } catch (RemoteException e) {
         e.printStackTrace();
      }
   }
```

```
/**
 * Read a report
 */
public void readReport() {
   try {
      ReportSubject subject = ReportSubject.DEVICE_SUMMARY;
      OutputFormat format = OutputFormat.HTML;
      ReportStatus status = lmInstance.readReport(token,
                        subject, format);

      if (status == null || status.getErrorCode()
                         != ClmErrors.SUCCESS) {
         System.out.println("Read report failed!");
      } else {
         System.out.println("Read report succeeded!");

         String url = status.getContent();
         System.out.println("Report URL is – "+url);
      }
   } catch (RemoteException e) {
      e.printStackTrace();
   }
}


/**
 * @param args
 */
public static void main(String[] args) {
                  CLMReportTest testApp = new CLMReportTest();
   testApp.init();

   testApp.generateReport();
   testApp.readReport();

   testApp.generateFilteredReport();
```

```
            testApp.readReport();

            testApp.close();
        }

    }
```

See Also

- For detailed descriptions of functions used in this program, see Chapter 13 of the Java API Reference Guide for Cisco License Manager.
- For detailed descriptions of each type of report, see Chapter 6 of the *User Guide for Cisco License Manager*.
- For login functions to Cisco License Manger server, see Chapter 3.