



Java API Reference Guide for Cisco License Manager

Release 2.0
November 2007

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-13001-01

~~THE SPECIFICATION AND INFORMATION REGARDING THE PRODUCT IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND. EXPRESS OR IMPLIED. USER MUST TAKE FULL RESPONSIBILITY FOR THE APPLICATION OF ANY PRODUCTS.~~

~~THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.~~

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

o Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, ce marks of Cisco Systems, Inc.; and Access Registrar, Aironet, ASIST, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Ci Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Empowering the In erChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expert rd, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, Packet, PIX, Post-Ro X, ScriptShare, SlideCast, SMARTnet, StrataView Plus, TeleRouter, The Fastest Way to Increase Your Internet Quotient, and TransPath ystems, Inc. and/or its affiliates in the United States and certain other countries.

entioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partner y other company. (0502R)

Any Internet Protocol (IP) addresses used in this document are not intended to be actual addresses. Any examples, command display output, and figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses in illustrative content is unintentional and coincidental.

Java API Reference Guide for Cisco License Manager
© 2007 Cisco Systems, Inc. All rights reserved.



CONTENTS

License and Warranty Information	ix
Supplemental License Agreement	ix
Cisco Limited 90-Day Software Warranty Terms	x
Preface	xiii
Audience	xiii
Purpose	xiii
Organization	xiii
Conventions	xiv
Product Documentation	xv
Obtaining Documentation, Obtaining Support, and Security Guidelines	xvi

CHAPTER 1

Getting Started	1-1
Overview	1-1
Installation	1-5
API Prerequisites	1-5
Installing the Cisco License Manager SDK	1-6
Using the SDK	1-7

CHAPTER 2

Access Control Functions	2-1
addUserToDeviceAccessList	2-1
addUserToGroupAccessList	2-2
addUserToPAKAccessList	2-3
getAssociatedActions	2-4
getUserAccessListFromDevice	2-4
getUserAccessListFromGroup	2-5
getUserAccessListFromPAK	2-6
removeAccessListFromDevice	2-6
removeAccessListFromGroup	2-7
removeUserFromDeviceAccessList	2-8
removeUserFromGroupAccessList	2-8

[removeUserFromPAKAccessList](#) 2-9

CHAPTER 3

AuditTrail Management Functions 3-1

[AuditTrailRecords](#) 3-1

[readAuditTrailRecords](#) 3-2

CHAPTER 4

Device Group Management Functions 4-1

[addDevicesToGroup](#) 4-1

[createDeviceGroup](#) 4-2

[deleteDeviceGroup](#) 4-3

[listAllGroups](#) 4-3

[removeDevicesFromGroup](#) 4-4

[renameDeviceGroup](#) 4-5

CHAPTER 5

Device Management Functions 5-1

[asyncDiscoverDevices](#) 5-1

[asyncPollDeviceLicenseInfo](#) 5-4

[createDevicesByIPAddr](#) 5-5

[createDevicesByUDI](#) 5-7

[deleteAsyncOperationJobRecords](#) 5-8

[deleteDevices](#) 5-9

[getAsyncOperationJobRecords](#) 5-10

[getAsyncOperationStatus](#) 5-11

[killAsyncOperation](#) 5-12

[listAllAsyncOperationJobRecords](#) 5-12

[listAllDevicesInGroup](#) 5-13

[listAllGroupsByDevice](#) 5-14

[listDeviceIdsByFilter](#) 5-14

[listRunningAsyncOperationJobRecords](#) 5-15

[readDevices](#) 5-16

[reCreateDevices](#) 5-17

[setAsyncOperationJobComment](#) 5-19

[writeDevices](#) 5-19

CHAPTER 6

Folder Management Functions 6-1

[addPAKsToFolder](#) 6-1

createFolder	6-2
deleteFolder	6-3
listAllFolders	6-3
removePAKsFromFolder	6-4
renameFolder	6-5

CHAPTER 7**License Inventory Management Functions 7-1**

asyncAnnotateLicenses	7-1
asyncDeployLicenses	7-3
asyncObtainLicense	7-4
getLicensesOnDevice	7-6
getRehostInfo	7-7
initRehostLicense	7-7
listAllLicensesInPAK	7-8
obtainLicenseForRehost	7-9
readLicenses	7-10
rehostLicense	7-12
reObtainLicense	7-12
resendLicense	7-13
revokeLicenseForRehost	7-14
writeLicenses	7-15

CHAPTER 8**License Line Management Functions 8-1**

asyncAnnotateLicenseLines	8-1
asyncDeployLicenseLines	8-3
getLicenseLinesByLicense	8-4
getLicenseLinesOnDevice	8-5
listExpiredLicenseLines	8-6
readLicenseLines	8-7
writeLicenseLines	8-8

CHAPTER 9**License Manager Class Constructors 9-1**

LicenseManager (Default)	9-1
LicenseManager (Nondefault)	9-1

CHAPTER 10

Notification Handling Functions 10-1

- deregisterNotificationListener 10-1
- disableNotificationByEmail 10-2
- enableNotificationByEmail 10-2
- registerNotificationListener 10-3

CHAPTER 11

PAK Management Functions 11-1

- asyncDownloadPAKInfo 11-1
- createPAKs 11-3
- deletePAKs 11-4
- listAllFoldersByPAK 11-6
- listAllPAKsInFolder 11-6
- listPAKContainFeatures 11-7
- readPAKs 11-8
- writePAKs 11-9

CHAPTER 12

Policy Management Functions 12-1

- asyncExecutePolicy 12-1
- createPolicy 12-3
- deletePolicy 12-4
- enumerateDeviceFilterAttribute 12-4
- enumerateSKUFilterAttribute 12-5
- listAllPolicies 12-6
- listFilteredDevices 12-6
- listFilteredSKUs 12-7
- readPolicy 12-7
- writePolicy 12-8

CHAPTER 13

Report Management Functions 13-1

- generateReport 13-1
- readReport 13-2

CHAPTER 14

Scheduler Management Functions 14-1

- listScheduler 14-1
- registerScheduler 14-2
- unregisterScheduler 14-2

CHAPTER 15	Troubleshooting Utility Functions	15-1
	checkCiscoPortalConnection	15-1
	checkDeviceConnection	15-1
CHAPTER 16	User Login and Logout Functions	16-1
	checkUserEULAStatus	16-1
	login	16-2
	logout	16-3
CHAPTER 17	User Management Functions	17-1
	createUser	17-1
	deleteUser	17-2
	listAllUsers	17-3
	readUserInfo	17-3
	writeUserInfo	17-4
APPENDIX A	Sample Client Program in Java	A-1
APPENDIX B	Data Object Definitions	B-1
APPENDIX C	Access Control	15
	User Control	15
	Device Control	17
	PAK Control	17
	Rules of Control	17



License and Warranty Information

Supplemental License Agreement

SUPPLEMENTAL LICENSE AGREEMENT FOR CISCO SYSTEMS LICENSE MANAGER SOFTWARE DEVELOPMENT KIT ("SDK")

IMPORTANT—READ CAREFULLY: This Supplemental License Agreement (“SLA”) contains additional limitations on the license to the Software provided to Customer under the Software License Agreement between Customer and Cisco. Capitalized terms used in this SLA and not otherwise defined herein shall have the meanings assigned to them in the Software License Agreement. To the extent that there is a conflict among any of these terms and conditions applicable to the Software, the terms and conditions in this SLA shall take precedence.

By installing, downloading, accessing or otherwise using the Software, Customer agrees to be bound by the terms of this SLA. If Customer does not agree to the terms of this SLA, Customer may not install, download, or otherwise use the Software. When used below, the term “server” refers to central processor unit.

1. ADDITIONAL LICENSE RESTRICTIONS.

- **Installation and Use.** Customer may only install and use the SDK on a single development station or node for internal development of products which will utilize Cisco Systems License Manager software ("CLM Software").
- Use of the SDK on multiple stations or nodes requires the purchase of an additional SDK for each of those units. Installation and use of CLM Software APIs that may be made available to Customer at Cisco's sole discretion are governed under separate terms and conditions.
- **Reproduction and Distribution.** Customer may not reproduce nor distribute SDK software.

2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

Please refer to the Cisco Systems, Inc. Software License Agreement.

Cisco Limited 90-Day Software Warranty Terms

There are special terms applicable to your software warranty and various services that you can use during the warranty period. Your formal Warranty Statement, including the warranties and license agreements applicable to Cisco software, is available on Cisco.com. Follow these steps to access and download the *Cisco Information Packet* and your warranty and license agreements from Cisco.com.

1. Launch your browser, and go to this URL:

http://www.cisco.com/univercd/cc/td/doc/es_inpk/cetrans.htm

The Warranties and License Agreements page appears.

2. To read the *Cisco Information Packet*, follow these steps:

- a. Click the **Information Packet Number** field, and make sure that the part number 78-5235-03B0 is highlighted.
- b. Select the language in which you would like to read the document.
- c. Click **Go**.

The Cisco Limited Warranty and Software License page from the Information Packet appears.

- d. Read the document online, or click the **PDF** icon to download and print the document in Adobe Portable Document Format (PDF).



Note You must have Adobe Acrobat Reader to view and print PDF files. You can download the reader from Adobe's website: <http://www.adobe.com>

3. To read translated and localized warranty information about your product, follow these steps:

- a. Enter this part number in the Warranty Document Number field:
78-13712-01C0
- b. Select the language in which you would like to read the document.
- c. Click **Go**.

The Cisco warranty page appears.

- d. Read the document online, or click the **PDF** icon to download and print the document in Adobe Portable Document Format (PDF).

You can also contact the Cisco service and support website for assistance:

http://www.cisco.com/public/Support_root.shtml.

Duration of Software Warranty

Ninety (90) Days

To Receive a Return Materials Authorization (RMA) Number

Contact the company from whom you purchased the product. If you purchased the product directly from Cisco, contact your Cisco Sales and Service Representative.

Complete the information below, and keep it for reference.

Company product purchased from	
Company telephone number	

Product model number	
Product serial number	
Maintenance contract number	



Preface

This preface explains the audience, purpose, and organization of the *Java API Reference Guide for Cisco License Manager*. It also defines the conventions that are used to present instructions and information.

Audience

The *Java API Reference Guide for Cisco License Manager* is intended for system administrators who install and configure internetworking software and who are familiar with Cisco IOS software or for software developers who plan to create scripts to run Cisco License Manager functions.

Purpose

The *Java API Reference Guide for Cisco License Manager* provides detailed information about the Java application programming interface (API) functions used with Cisco License Manager Release 2.0. For additional information on related documentation, see the [“Product Documentation” section on page xv](#).

Organization

The *Java API Reference Guide for Cisco License Manager* is organized as follows:

- [Chapter 1, “Getting Started,”](#) provides overview and installation information needed to use the Java API with Cisco License Manager Release 2.0.
- [Chapter 2, “Access Control Functions,”](#) provides information about the access control functions.
- [Chapter 3, “AuditTrail Management Functions,”](#) provides information about the audit trail management functions.
- [Chapter 4, “Device Group Management Functions,”](#) provides information about the device group management functions.
- [Chapter 5, “Device Management Functions,”](#) provides information about the device management functions.
- [Chapter 6, “Folder Management Functions,”](#) provides information about the folder management functions.

- [Chapter 7, “License Inventory Management Functions,”](#) provides information about the license inventory management functions.
- [Chapter 8, “License Line Management Functions,”](#) provides information about the license line management functions.
- [Chapter 9, “License Manager Class Constructors,”](#) provides information about the Cisco License Manager class constructors:
- [Chapter 10, “Notification Handling Functions,”](#) provides information about the notification handling functions.
- [Chapter 11, “PAK Management Functions,”](#) provides information about the product authorization key (PAK) management functions.
- [Chapter 12, “Policy Management Functions,”](#) provides information about the policy management functions.
- [Chapter 13, “Report Management Functions,”](#) provides information about the report management functions.
- [Chapter 14, “Scheduler Management Functions,”](#) provides information about the scheduler management functions.
- [Chapter 15, “Troubleshooting Utility Functions,”](#) provides information about the troubleshooting utility functions.
- [Chapter 16, “User Login and Logout Functions,”](#) provides information about the user login and logout functions.
- [Chapter 17, “User Management Functions,”](#) provides information about the user management functions.
- [Appendix A, “Sample Client Program in Java,”](#) provides a sample client program in Java.
- [Appendix B, “Data Object Definitions,”](#) provides definitions of data objects that are used by the LicenseManager class.
- [Appendix C, “Access Control,”](#) provides information about the levels of access control in Cisco License Manager, as well as the rules of control.

Conventions

This document uses the conventions listed in [Table 1](#).

Table 1 **Font Conventions**

Item	Convention
Commands and keywords	boldface font
Variables for which you supply values	<i>italic</i> font
Displayed session and system information	screen font
Information you enter	boldface screen font
Menu items and button names	boldface font

**Note**

Means *reader take note*. Notes contain helpful suggestions or references to material not contained in the publication.

**Caution**

Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

**Tip**

Means the following information will help you solve a problem.

Product Documentation

**Note**

We sometimes update the printed and electronic documentation after original publication. Therefore, you should also review the documentation on Cisco.com for any updates.

The following table lists the Cisco License Manager Release 2.0 documentation.

Table 2 **Product Documentation**

Document	Location
<i>Home page for Cisco License Manager documentation (all releases)</i>	On Cisco.com at http://www.cisco.com/en/US/products/ps7138/tsd_products_support_series_home.html
<i>Release Notes for Cisco License Manager Release 2.0</i>	On Cisco.com at http://www.cisco.com/en/US/products/ps7138/prod_release_note09186a00808f84de.html
<i>Finding Documentation for Cisco License Manager Release 2.0</i>	<ul style="list-style-type: none"> • Adobe Acrobat PDF on SDK CD • On Cisco.com at http://www.cisco.com/en/US/products/ps7138/products_documentation_roadmap09186a00808f7c00.html (in HTML and PDF format)
<i>User Guide for Cisco License Manager Release 2.0</i>	On Cisco.com at http://www.cisco.com/application/pdf/en/us/guest/products/ps8295/c1626/ccmigration_09186a00808c04d8.pdf
<i>Java API Reference Guide for Cisco License Manager Release 2.0</i>	<ul style="list-style-type: none"> • Adobe Acrobat PDF on SDK CD • On Cisco.com at http://www.cisco.com/en/US/products/ps7138/products_programming_reference_guide_book09186a00808e2dce.html (in HTML and PDF format)

Table 2 **Product Documentation (continued)**

Document	Location
<i>Perl API Reference Guide for Cisco License Manager Release 2.0</i>	<ul style="list-style-type: none">• Adobe Acrobat PDF on SDK CD• On Cisco.com at http://www.cisco.com/en/US/products/ps7138/products_programming_reference_guide_book09186a00808e2f0e.html (in HTML and PDF format)
Online Help	In the user interface, click Help to display help in a separate window.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

<http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html>



CHAPTER 1

Getting Started

This chapter provides overview and installation information needed to use the Java application programming interface (API) with Cisco License Manager Release 2.0 and contains the following sections:

- [Overview, page 1-1](#)
- [Installation, page 1-5](#)
- [Using the SDK, page 1-7](#)

Overview

Cisco License Manager provides an object-oriented Java API for client programs to invoke functions implemented in the Cisco License Manager back-end server.

The API is a set of Java class libraries in a single Java archive (JAR) file. The top-level class License Manager contains all functions that can be invoked by client programs. This is the same class library set used by the Cisco License Manager graphical user interface (GUI). This guide provides the external Java API as seen by its client programs and includes the following functions:

- [addDevicesToGroup, page 4-1](#)
- [addPAKsToFolder, page 6-1](#)
- [addUserToDeviceAccessList, page 2-1](#)
- [addUserToGroupAccessList, page 2-2](#)
- [addUserToPAKAccessList, page 2-3](#)
- [asyncAnnotateLicenseLines, page 8-1](#)
- [asyncAnnotateLicenses, page 7-1](#)
- [asyncDeployLicenseLines, page 8-3](#)
- [asyncDeployLicenses, page 7-3](#)
- [asyncDiscoverDevices, page 5-1](#)

- [asyncDownloadPAKInfo](#), page 11-1
- [asyncExecutePolicy](#), page 12-1
- [asyncObtainLicense](#), page 7-4
- [asyncPollDeviceLicenseInfo](#), page 5-4
- [AuditTrailRecords](#), page 3-1
- [checkCiscoPortalConnection](#), page 15-1
- [checkDeviceConnection](#), page 15-1
- [checkUserEULAStatus](#), page 16-1
- [createDeviceGroup](#), page 4-2
- [createDevicesByIPAddr](#), page 5-5
- [createDevicesByUDI](#), page 5-7
- [createFolder](#), page 6-2
- [createPAKs](#), page 11-3
- [createPolicy](#), page 12-3
- [createUser](#), page 17-1
- [deleteAsyncOperationJobRecords](#), page 5-8
- [deleteDeviceGroup](#), page 4-3
- [deleteDevices](#), page 5-9
- [deleteFolder](#), page 6-3
- [deletePAKs](#), page 11-4
- [deletePolicy](#), page 12-4
- [deleteUser](#), page 17-2
- [deregisterNotificationListener](#), page 10-1
- [disableNotificationByEmail](#), page 10-2
- [enableNotificationByEmail](#), page 10-2
- [enumerateDeviceFilterAttribute](#), page 12-4

- [enumerateSKUFilterAttribute](#), page 12-5
- [generateReport](#), page 13-1
- [getAssociatedActions](#), page 2-4
- [getAsyncOperationJobRecords](#), page 5-10
- [getAsyncOperationStatus](#), page 5-11
- [getLicenseLinesByLicense](#), page 8-4
- [getLicenseLinesOnDevice](#), page 8-5
- [getLicensesOnDevice](#), page 7-6
- [getRehostInfo](#), page 7-7
- [getUserAccessListFromDevice](#), page 2-4
- [getUserAccessListFromGroup](#), page 2-5
- [getUserAccessListFromPAK](#), page 2-6
- [initRehostLicense](#), page 7-7
- [killAsyncOperation](#), page 5-12
- [LicenseManager \(Default\)](#), page 9-1
- [LicenseManager \(Nondefault\)](#), page 9-1
- [listAllAsyncOperationJobRecords](#), page 5-12
- [listAllDevicesInGroup](#), page 5-13
- [listAllFolders](#), page 6-3
- [listAllFoldersByPAK](#), page 11-6
- [listAllGroups](#), page 4-3
- [listAllGroupsByDevice](#), page 5-14
- [listAllLicensesInPAK](#), page 7-8
- [listAllPAKsInFolder](#), page 11-6
- [listAllPolicies](#), page 12-6
- [listAllUsers](#), page 17-3

- [listDeviceIdsByFilter](#), page 5-14
- [listExpiredLicenseLines](#), page 8-6
- [listFilteredDevices](#), page 12-6
- [listFilteredSKUs](#), page 12-7
- [listPAKContainFeatures](#), page 11-7
- [listRunningAsyncOperationJobRecords](#), page 5-15
- [listScheduler](#), page 14-1
- [login](#), page 16-2
- [logout](#), page 16-3
- [obtainLicenseForRehost](#), page 7-9
- [readAuditTrailRecords](#), page 3-2
- [readDevices](#), page 5-16
- [readLicenseLines](#), page 8-7
- [readLicenses](#), page 7-10
- [readPAKs](#), page 11-8
- [readPolicy](#), page 12-7
- [readReport](#), page 13-2
- [readUserInfo](#), page 17-3
- [reCreateDevices](#), page 5-17
- [registerNotificationListener](#), page 10-3
- [registerScheduler](#), page 14-2
- [rehostLicense](#), page 7-12
- [removeAccessListFromDevice](#), page 2-6
- [removeAccessListFromGroup](#), page 2-7
- [removeDevicesFromGroup](#), page 4-4
- [removePAKsFromFolder](#), page 6-4

- [removeUserFromDeviceAccessList](#), page 2-8
- [removeUserFromGroupAccessList](#), page 2-8
- [removeUserFromPAKAccessList](#), page 2-9
- [renameDeviceGroup](#), page 4-5
- [renameFolder](#), page 6-5
- [reObtainLicense](#), page 7-12
- [resendLicense](#), page 7-13
- [revokeLicenseForRehost](#), page 7-14
- [setAsyncOperationJobComment](#), page 5-19
- [unregisterScheduler](#), page 14-2
- [writeDevices](#), page 5-19
- [writeLicenseLines](#), page 8-8
- [writeLicenses](#), page 7-15
- [writePAKs](#), page 11-9
- [writePolicy](#), page 12-8
- [writeUserInfo](#), page 17-4

Installation

The Java API communicates with the Cisco License Manager Server using Java.

This section provides the following installation information:

- [API Prerequisites](#), page 1-5
- [Installing the Cisco License Manager SDK](#), page 1-6

API Prerequisites

The API requires Java Development Kit (JDK) Version 1.5.

The following table provides details of the recommended software and hardware requirements for the Cisco License Manager server. These recommended requirements support the maximum data capacity (30,000 Cisco devices in the network). Also use these requirements if you are installing the server and client on the same host.

Table 1-1 Server Recommended Software and Hardware Requirements

Requirement Type	Recommended Requirement
Processor	Intel Pentium 4 3.2 GHz or equivalent CPU-based machine Sun UltraSPARC IIIi or equivalent CPU-based machine
Software	Operating system <ul style="list-style-type: none"> Windows Server 2003 R2 Standard Edition Windows XP Professional Solaris 10 (SPARC system only)
RAM	2 GB
Disk	10 GB

The following table provides details of the recommended software and hardware requirements for the Cisco License Manager client workstation.

Table 1-2 Client Workstation Recommended Software and Hardware Requirements

Requirement Type	Recommended Requirement
Processor	Intel Pentium 4 3.2 GHz or equivalent CPU-based machine Sun UltraSPARC IIIi or equivalent CPU-based machine
Software	Operating system <ul style="list-style-type: none"> Windows Server 2003 R2 Standard Edition Windows XP Professional Solaris 10 (SPARC system only)
RAM	1 GB
Disk	10 GB
Network connectivity	Server must be reachable from the client software.
Java requirements	Version 1.5

Installing the Cisco License Manager SDK

Cisco recommends that you install the Cisco License Manager Software Developer Kit (SDK) on the same machine on which the server resides.

To install the Cisco License Manager SDK, complete the following 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

conf, a folder containing the file ClmErrorMessages.properties.

Using the SDK

Complete the following steps to begin using the SDK:

Step 1 Add clm-sdk.jar to the Java classpath when compiling and running the client program.

```
javac -classpath clm-sdk.jar MyProgram.java
```

Step 2 If the Cisco License Manager License Manager server is SSL enabled:

Run the following command to import the Cisco License Manager certificate file, clm_ssl_rmi.cert, to the trust store on your client system:

```
%JAVA_HOME%\bin\keytool -import -keystore <keystore_file_path> -alias  
ciscolicensemanagerca -file <clm_ssl_rmi_cer_file_path>
```

where <keystore_file_path> is the trust store of your client system, for example, C:\jdk1.5\jre\lib\security\cacerts, and <clm_ssl_rmi_cer_file_path> is the fullpath of the clm_ssl_rmi.cert file, for example, C:\CLM2.0_SDK\clm_ssl_rmi.cert.

Step 3 Start the Cisco License Manager server. For details, refer to the [User Guide for Cisco License Manager](#).

Step 4 Start the client program.



CHAPTER 2

Access Control Functions

This chapter provides information about the following access control functions:

- [addUserToDeviceAccessList](#), page 2-1
- [addUserToGroupAccessList](#), page 2-2
- [addUserToPAKAccessList](#), page 2-3
- [getAssociatedActions](#), page 2-4
- [getUserAccessListFromDevice](#), page 2-4
- [getUserAccessListFromGroup](#), page 2-5
- [getUserAccessListFromPAK](#), page 2-6
- [removeAccessListFromDevice](#), page 2-6
- [removeAccessListFromGroup](#), page 2-7
- [removeUserFromDeviceAccessList](#), page 2-8
- [removeUserFromGroupAccessList](#), page 2-8
- [removeUserFromPAKAccessList](#), page 2-9

addUserToDeviceAccessList

Synopsis

```
DeviceStatus addUserToDeviceAccessList(UserToken token, String dev, String[] usr_ids)  
throws RemoteException;
```

Description

This function provides user access control and associates a device with a list of user IDs.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev	String, mandatory	—	The device ID.
user_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	Array of user IDs to be added to a device access list.

Return

This function returns a DeviceStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an operation error occurs, the information is contained in the returned DeviceStatus object. To inspect the status of an individual element, you must traverse the DeviceStatusItem array within DeviceStatus. Each DeviceStatusItem contains the Device object, error code, and error message.

addUserToGroupAccessList

Synopsis

```
GroupStatus AddUserToDeviceAccessList(UserToken token, String group, String[] usr_ids)
throws RemoteException;
```

Description

This function provides user access control. It adds user IDs to the access list of a given group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Parameter	Type	Value	Description
group	String, mandatory	—	The group ID.
user_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of user IDs to be added to a device access list.

Return

This function returns a GroupStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

This function returns the GroupStatus object that contains an error code and error message when operation error occurs. Each GroupStatusItem contains individual Group object and error code and error message.

addUserToPAKAccessList

Synopsis

```
PAKStatus AddUserToPAKAccessList(UserToken token, String pak, String[] usr_ids) throws RemoteException;
```

Description

This function adds an array of user IDs into the access list of a given product authorization key (PAK) name. This function supports user access control.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak	String, mandatory	—	The PAK ID.
User_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of user IDs to be added to the PAK access list.

Return

This function returns a PAKStatus object. If the operation is successful, the error code will be SUCCESS. PAKStatus contains an array of PAKStatusItem. Each PAKStatusItem contains the PAK Object, error code, and error message.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The PAKStatus contains overall operation status. When a system-level error occurs and the operation cannot be completed, the PAKStatus error code returns none SUCCESS and an error message if the cause of the failure is known. For each individual ID, the PAKStatus contains PAKStatusItem[]. Each PAKStatusItem contains a PAK object, error code, and error message that shows the individual status.

getAssociatedActions

Synopsis

```
String[] getAssociatedActions (UserToken token, String username) throws RemoteException;
```

Description

This function returns a list of strings representing the actions that can be performed.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
username	String, mandatory	—	The username.

Return

This function returns an array of String.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

getUserAccessListFromDevice

Synopsis

```
String[] getUserAccessListFromDevice (UserToken token, String dev_id) throws RemoteException;
```

Description

This function returns the user access list of a given device ID as an array of string. Each string is a user ID.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	—	The device ID.

Return

This function returns an array of string.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

getUserAccessListFromGroup

Synopsis

```
String[] getUserAccessListFromGroup (UserToken token, String group) throws RemoteException;
```

Description

This function returns the user access list of a given group as an array of string. Each string is a user ID.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	—	The group name.

Return

This function returns an array of string.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

getUserAccessListFromPAK

Synopsis

```
String[]getUserAccessListFromPAK (UserToken token, String pak_id) throws RemoteException;
```

Description

This function returns the user access list of a given PAK as an array of string. Each string is a user ID.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_id	String, mandatory	—	The PAK ID.

Return

This function returns an array of string.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

removeAccessListFromDevice

Synopsis

```
public Status removeAccessListFromDevice(UserToken token, String dev_id) throws  
RemoteException;
```

Description

This function removes an access list from the specified device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	—	The device ID of the access list to be removed.

Return

This function returns a Status object. If an error occurs, the Status object contains an error code with none ClmErrors.SUCCESS and an error message regarding the error.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

removeAccessListFromGroup

Synopsis

```
public Status removeAccessListFromGroup(UserToken token, String group_id) throws
RemoteException;
```

Description

This function removes an access list from the specified group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group_id	String, mandatory	—	The group ID of the access list to be removed.

Return

The function returns a Status object. If an error occurs, the Status object contains the error code none ClmErrors.SUCCESS and an error message regarding the error.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

removeUserFromDeviceAccessList

Synopsis

```
DeviceStatus RemoveUserFromDeviceAccessList(UserToken token, String dev, String[] usr_ids)
throws RemoteException;
```

Description

This function provides user access control. It removes a list of user IDs from the access list of a given device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev	String, mandatory	—	Device ID.
user_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of user IDs to be removed from a device access list.

Return

This function returns a DeviceStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an operation error occurs, information about the error is contained in the returned DeviceStatus object. To inspect the individual element status, you must traverse the DeviceStatusItem array within DeviceStatus. Each DeviceStatusItem contains the Device object, error code, and error message.

removeUserFromGroupAccessList

Synopsis

```
GroupStatus removeUserFromDeviceAccessList(UserToken token, String group, String[]
usr_ids) throws RemoteException;
```

Description

This function provides user access control. It removes a list of user IDs from the access list of a given group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	—	The group ID.
user_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of user IDs to be removed from a group access list.

Return

This function returns a GroupStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

This function returns a GroupStatus object that contains none SUCCESS error code and error message when an operation error occurs and also an array of GroupStatusItems. Each GroupStatusItem contains an individual GROUP object, error code, and error message.

removeUserFromPAKAccessList

Synopsis

```
PAKStatus removeUserFromPAKAccessList(UserToken token, String pak, String[] usr_ids)
throws RemoteException);
```

Description

This function removes an array of user IDs from the access list of a given PAK ID.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Parameter	Type	Value	Description
pak	String, mandatory	—	The PAK ID.
Usr_ids	Array of string, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of user IDs to be removed from PAK access list.

Return

This function returns a PAKStatus object. The error code will be SUCCESS if the operation is successful. PAKStatus contains an array of PAKStatusItem. Each PAKStatusItem contains a PAK Object, error code, and error message.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The PAKStatus contains overall operation status. When a system-level error occurs and the operation cannot be completed, the PAKStatus error code returns none SUCCESS and is accompanied by an error message (if the cause of failure is known). For each individual ID, PAKStatus contains PAKStatusItem[]. Each PAKStatusItem contains a PAK object, error code, and error message that shows the individual status.



CHAPTER 3

AuditTrail Management Functions

This chapter provides information about the following audit trail management functions:

- [AuditTrailRecords](#), page 3-1
- [readAuditTrailRecords](#), page 3-2

AuditTrailRecords

Synopsis

```
Status purgeAuditTrailRecords(UserToken token, int numToKeep) throws RemoteException;
```

Description

This function purges the audit trail logs that contain records of user activities. It removes the older records that exceed the numToKeep parameter.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
numToKeep	Integer, mandatory	Valid integer value	The number (quantity) of the log records to keep.

Return

This function returns a Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, the Status object returns the none SUCCESS error code and error message.

readAuditTrailRecords

Synopsis

```
AuditTrail[] readAuditTrailRecords(UserToken token, int numRecords) throws
RemoteException;
```

Description

This function retrieves the audit trail logs that contain records of user activities. If numRecords is set to 0, it returns all records in the system.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
numRecords	Integer, mandatory	Valid integer value	The number (quantity) of the log records to keep.

Return

This function returns an array of AuditTrail objects.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null. An array of length zero indicates that no record was found.



CHAPTER 4

Device Group Management Functions

This chapter provides information about the following device group management functions:

- [addDevicesToGroup](#), page 4-1
- [createDeviceGroup](#), page 4-2
- [deleteDeviceGroup](#), page 4-3
- [listAllGroups](#), page 4-3
- [removeDevicesFromGroup](#), page 4-4
- [renameDeviceGroup](#), page 4-5

addDevicesToGroup

Synopsis

```
IDStatus addDevicesToGroup(UserToken token, String[] dev_ids, String group) throws  
RemoteException;
```

Description

This function adds devices to the specified device group. You can add devices to more than one group.



Note

If you want to keep a device in a single group, you should remove the device from any other group by calling the `removeDevicesFromGroup()` function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Parameter	Type	Value	Description
dev_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs to be added to the group.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the group.

Return

This function returns an IDStatus object. If the operation is successful, CImErrors.SUCCESS is returned in the error code and the IDStatus object contains IDStatusItem[]. Each IDStatusItem contains the ID of the device that is being added to the group. If the device is added successfully to the group, the error code in the IDStatusItem returns CImErrors.SUCCESS.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the IDStatus object with the error code and error message.

createDeviceGroup

Synopsis

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

Description

This function creates a new device group with the given name.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the group.

Return

This function returns a Status object. If the operation is successful, CImErrors.SUCCESS is returned in the error code.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns the `IDStatus` object with the error code and error message.

deleteDeviceGroup

Synopsis

```
Status deleteDeviceGroup(UserToken token, String group) throws RemoteException;
```

Description

This function deletes a device group from the system. Devices under this group are not deleted, but they become ungrouped.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the device group.

Return

This function returns a `Status` object. If the operation is successful, `ClmErrors.SUCCESS` is returned in the error code.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns the `IDStatus` object with the error code and error message.

listAllGroups

Synopsis

```
String[] listAllGroups(UserToken token) throws RemoteException;
```

Description

This function returns a list of all device groups in the system.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns an array representing the name of the device group.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

removeDevicesFromGroup

Synopsis

```
IDStatus removeDevicesFromGroup(UserToken token, String[] dev_ids, String group) throws RemoteException;
```

Description

This function removes devices from the specified device group. The devices become ungrouped but remain in the device inventory. To delete the devices from the inventory, use the deleteDevices() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs to be removed to the group.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the device group.

Return

This function returns an IDStatus object. If the operation is successful, ClmErrors.SUCCESS is returned in the error code, and the IDStatus object contains IDStatusItem[]. Each IDStatusItem contains the ID of the device that is being removed from the group. If the device is removed successfully from the group, the error code in the IDStatusItem returns ClmErrors.SUCCESS.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the IDStatus object with the error code and error message.

renameDeviceGroup

Synopsis

```
Status renameDeviceGroup(UserToken token, String group, String new_name) throws
RemoteException;
```

Description

This function renames a given device group. All devices contained by the group remain unchanged.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The current name of the device group.
new_name	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The new name for the device group.

Return

This function returns a Status object. If the operation is successful, ClmErrors.SUCCESS is returned in the error code.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the IDStatus object with the error code and error message.



Device Management Functions

This chapter provides information about the following device management functions:

- [asyncDiscoverDevices](#), page 5-1
- [asyncPollDeviceLicenseInfo](#), page 5-4
- [createDevicesByIPAddr](#), page 5-5
- [createDevicesByUDI](#), page 5-7
- [deleteAsyncOperationJobRecords](#), page 5-8
- [deleteDevices](#), page 5-9
- [getAsyncOperationJobRecords](#), page 5-10
- [getAsyncOperationStatus](#), page 5-11
- [killAsyncOperation](#), page 5-12
- [listAllAsyncOperationJobRecords](#), page 5-12
- [listAllDevicesInGroup](#), page 5-13
- [listAllGroupsByDevice](#), page 5-14
- [listDeviceIdsByFilter](#), page 5-14
- [listRunningAsyncOperationJobRecords](#), page 5-15
- [readDevices](#), page 5-16
- [reCreateDevices](#), page 5-17
- [setAsyncOperationJobComment](#), page 5-19
- [writeDevices](#), page 5-19

asyncDiscoverDevices

Synopsis

```
String asyncDiscoverDevices(UserToken token, String subnet, String subnet_mask, String group, DiscoveryAuthInfo dev_auth_info, Device.TransportMethod[] transports, IDStatusListener listener) throws RemoteException;
```

Description

This function generates an inventory of devices that are discovered in the given subnet and adds the devices to the specified device group.

The Cisco License Manager currently supports Cisco IOS Telnet and secure shell (ssh) for non-agent-enabled device discovery, thus prolonging the discovery time. If the network devices are configured the same way, you can provide the transport method (such as HTTP, Telnet, or ssh), which speeds up the discovery and reduces the discovery process time. If a null or empty string array is passed in, Cisco License Manager uses all supported methods to discover the devices.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements the IDStatusListener interface. When the operation is complete, the onStatus() method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
subnet	String, mandatory	Network address	The subnet to conduct the auto-discovery.
subnet_mask	String, mandatory	—	The subnet mask.
group	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The group to which the discovered devices should be added.
dev_auth_info	DiscoveryAuthInfo, mandatory	—	A DiscoveryAuthInfo is used when communicating with discovered devices to retrieve license information. If no authentication is needed for the devices in the network, this parameter can be set to null. DiscoveryAuthInfo contains an array of username and password pairs and an array of enable passwords. When Cisco License Manager tries to communicate with the device, the instances of DiscoveryAuthInfo are tried one at a time until they can be authenticated by the device. If Telnet is used, there are only three tries.

Parameter	Type	Value	Description
listener	IDStatusListener object, mandatory	—	An object that implements the IDStatusListener interface.
transports	Device. TransportMethod array, optional	HTTP, Telnet, ssh	If the devices in a subnet are set up with the same device discovery method, the device discovery process is speeded up because Cisco License Manager uses only the specified method to quickly determine if the device license is supported. This parameter specifies which connection transport is used. Cisco License Manager uses the specified transports by entry order. If null or zero length is entered, Cisco License Manager uses all available methods to discover the device.

Return

This function returns a request ID to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method. The listener will receive an IDStatus object which contains a list of IDStatusItem objects. Each IDStatusItem object contains the device ID of the discovered devices.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown. The input parameter of the onStatus() method in the IDStatusListener contains the error codes and messages. More than one error code and message may be contained in the IDStatus object. The following example shows the error codes and messages in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();

    }
}
```

asyncPollDeviceLicenseInfo

Synopsis

String asyncPollDeviceLicenseInfo(UserToken token, String[] dev_ids, IDStatusListener listener) throws RemoteException;

Description

This function communicates with the given devices, retrieving license information from them and storing the information in the inventory.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements the IDStatusListener interface. When the operation is complete, the onStatus() method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method. The listener will receive IDStatus when the operation is completed. The IDStatus contains the list of IDStatusItem, which includes the device ID and the error code and error message of the individual device poll operation.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown. The input parameter of the onStatus() method in IDStatusListener contains the error codes and messages. More than one error code and message may be contained in the IDStatus object. The following example shows the error codes and messages in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();
}
```

```

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

    // Get the individual object ID returned by the operation.
    String id = items[i].getID();

    // Get the individual error code corresponding to
    // the object ID.
    int item_err_code = items[i].getErrorCode();

    // Get the individual error message corresponding to
    // the object ID.
    String item_err_msg = items[i].getErrorMessage();
}
}

```

createDevicesByIPAddr

Synopsis

DeviceStatus createDevicesByIPAddr(UserToken token, String[] ips, String group, DeviceAuthentication[] dev_auth_info, Device.TransportMethod[] transports) throws RemoteException;

Description

This function creates device objects in the inventory using a given IP address and associates devices with the specified group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
ips	Array of string, mandatory	IP address	An array of IP addresses. Each address is used for creating a device object.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the device group.

Parameter	Type	Value	Description
dev_auth_info	Array of Device Authentication, mandatory	—	An array of DeviceAuthentication values is used when communicating with discovered devices to retrieve license information. Each entry in the array is used to connect to the IP address in the IP array. DeviceAuthentication contains username, password, and enable password.
transports	Device. TransportMethod array, optional	HTTP, Telnet, ssh	This parameter specifies which connection transport method is used for each IP entry. Cisco License Manager assumes that the caller knows the connection method used for each device.

Return

This function returns DeviceStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

DeviceStatus status = createDevicesByIPAddr(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
DeviceStatusItem[] items = status.getDeviceStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
Device device = items[i].getDevice();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an operation error occurs, a DeviceStatus object is returned with information about the error. To inspect the individual element status, you must traverse the DeviceStatusItem array within DeviceStatus. Each DeviceStatusItem contains the Device object, the error code, and the error message.

createDevicesByUDI

Synopsis

`DeviceStatus createDevicesByUDI(UserToken token, String[] udis)` throws `RemoteException`;

`DeviceStatus createDevicesByUDI(UserToken token, String[] udis, String group)` throws `RemoteException`;

Description

This function creates device objects in the inventory using a given unique device identifier (UDI). The second form of the function also adds the created devices to the specified group. If the group is not specified, the device is added to the default group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
udis	Array of string, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	An array of UDIs. Each UDI is used for creating a device object.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the device group.

Return

This function returns `DeviceStatus` objects. The following example shows the error code, error messages, and returned objects in the status:

```
DeviceStatus status = createDevicesByUDI(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
DeviceStatusItem[] items = status.getDeviceStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
Device device = items[i].getDevice();
```

```

// Get the individual error code corresponding to
// the object.
    int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
    String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an operation error occurs, a `DeviceStatus` object is returned with information about the error. To inspect the individual element status, you must traverse the `DeviceStatusItem` array within `DeviceStatus`. Each `DeviceStatusItem` contains the `Device` object, the error code, and the error message.



Note

This API can be used for creating a device that contains member devices. The following is an example of API code with a master switch and member switch in a switch stack:

```

// Create 3 device objects, where dev_objs[0] is the master switch,
// dev_objs[1] is member switch 1 and dev_objs[2] is member switch 2.

String dev_udi[] = {"MasterSwitch", "MemberSwitch1", "MemberSwitch2"};
Device[] dev_objs =
    LicenseManager.createDeviceByUDI(token, dev_ids[]);

// Associate 2 member switches to their master switch.
dev_objs[0].member_device_ids = new String[2];
dev_objs[0].member_device_ids[0] = dev_ids[1];
dev_objs[0].member_device_ids[1] = dev_ids[2];

// Write the changes to the data storage.
LicenseManager.writeDevice(token, dev_ids);

```

deleteAsyncOperationJobRecords

Synopsis

```
IDStatus deleteAsyncOperationJobRecords(UserToken token, String[] job_ids) throws
RemoteException;
```

Description

This function allows you to delete job records that are associated with your username.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
job_ids	Array of string, mandatory	—	Array of Job IDs. Each Job ID is the request ID that is returned from an asynchronous function call.

Return

This function returns an IDStatus object. If the operation is not successful, the IDStatus returns a non ClmErrors.SUCCESS error code. If the operation is successful, the IDStatus contains an IDStatusItem array and each IDStatusItem contains the job ID and the error status of the delete operation.

Error and Exception

When an overall function error occurs, this function returns none ClmErrors.SUCCESS value in the IDStatus. The Job ID and the error code can be retrieved from the IDStatusItem array.

If a system error prevents the operation from completing, a RemoteException is thrown.

deleteDevices

Synopsis

```
IDStatus deleteDevices(UserToken token, String[] dev_ids) throws RemoteException;
```

Description

This function deletes device objects from the inventory using the given device IDs. All devices belong to a group. If a device does not belong to a user-defined group, it belongs to the default group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs.

Return

This function returns IDStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

IDStatus status = deleteDevices(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual ID returned by the operation.
String id = items[i].getID();

// Get the individual error code corresponding to
// the ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the ID.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.

getAsyncOperationJobRecords

Synopsis

```
ClmJob getAsyncOperationJobRecords(UserToken token, String job_id) throws RemoteException;
```

Description

This function returns the ClmJob that is associated with the asynchronous function. The job_id is the request ID returned from the asynchronous call.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
job_id	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	Job ID returned from asynchronous operation.

Return

This function returns ClmJob. If job_id does not exist, this function returns null. Otherwise it returns the ClmJob for the specific job_id.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

getAsyncOperationStatus

Synopsis

```
ProgressStatus getAsyncOperationStatus(UserToken token, String job_id) throws
RemoteException;
```

Description

This function provides information about the progress of an asynchronous function. The job_id is essential to track the progress. If the task_id is not provided, an error in the ProgressStatus is returned.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
job_id	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	Job ID returned from asynchronous operation.

Return

This function returns `ProgressStatus`, and it includes the overall status indicating if this function has completed successfully or not. The detail of the `ProgressStatus` contains an array of target IDs and an array of integers indicating the progress of the individual targets (for example, 0 indicates complete, 1 indicates pending, and 2 indicates processing). Also included is a string for the function that is currently being performed and the overall progress percentage.

Error and Exception

When an overall function error occurs, this function returns `none 0` in the `ProgressStatus`. If the asynchronous task is completed, the entry of the task is removed and the overall status returns a `task_id` not found error. You have to check if the callback has been returned to determine if the `task_id` that is passed is incorrect.

If a system error prevents the operation from completing, a `RemoteException` is thrown.

killAsyncOperation

Synopsis

```
Status killAsyncOperation(UserToken token, String job_id) throws RemoteException;
```

Description

This function kills the asynchronous operation.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
job_id	String, mandatory	Up to 64 ASCII characters in the range of x21 to x7A	Job ID, which is the same as the request ID, returned from asynchronous operation.

Return

This function returns a `Status` object.

Error and Exception

When an overall function error occurs, this function returns a `none SUCCESS` value in `Status`.

If a system error prevents the operation from completing, a `RemoteException` is thrown.

listAllAsyncOperationJobRecords

Synopsis

```
ClmJobStatus listAllAsyncOperationJobRecords(UserToken token, boolean include_all_user)
throws RemoteException;
```

Description

This function returns all of the Cisco License Manager jobs that contain information about the asynchronous operation stored in the inventory that you initiated. This function returns all complete, incomplete, and running Cisco License Manager jobs.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
Include_all_user	Boolean, mandatory	True, False	If true, this parameter returns all records. If false, this parameter returns only records with the username.

Return

This function returns a ClmStatus object.

Error and Exception

When an overall function error occurs, this function returns a none SUCCESS value in the ClmJobStatus object. The ClmJob can be retrieved from the ClmJobStatusItem array.

If a system error prevents the operation from completing, a RemoteException is thrown.

listAllDevicesInGroup

Synopsis

```
String[] listAllDevicesInGroup(UserToken token, String group) throws RemoteException;
```

Description

This function returns an array of device IDs that belong to the given device group. If the group is null, this function returns an array of device IDs that belong to the default group.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
group	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The name of the device group, or null if the function is looking for ungrouped devices.

Return

This function returns an array of device IDs if the group is valid. Otherwise, if the group is invalid or empty, a non-null array of size zero is returned.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown. When an error occurs, this function returns null.

listAllGroupsByDevice

Synopsis

```
String[] listAllGroupsByDevice(UserToken token, String dev_id) throws RemoteException;
```

Description

This function returns an array of groups to which the given device belongs.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs.

Return

This function returns an array of groups if the device is valid.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

listDeviceIdsByFilter

Synopsis

```
String[] listDeviceIdsByFilter(UserToken token, String device_type, String device_model, String[] features) throws RemoteException;
```

Description

This function returns a list of device IDs that contains the device_type, device_model, and license feature.

Each filter can be set to null for no filtering. If all the filters are set to null, the operation is treated as an error and null is returned.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
device_type	String, optional	—	String that represents device type.
device_model	String, optional	—	String that represents device model.
features	Array of string, optional	—	Array of string that represent the license feature names.

Return

This function returns an array of string. Each string is a device ID.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null. An array length of zero means that no device is found.

listRunningAsyncOperationJobRecords

Synopsis

```
ClmJobStatus listRunningAsyncOperationJobRecords(UserToken token, boolean
include_all_user) throws RemoteException;
```

Description

This function returns all of the Cisco License Manager jobs that contain information about the running asynchronous operations that you initiated.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
include_all_user	Boolean, mandatory	True, False	If true, this parameter returns all records. If false, this parameter returns only records containing the username.

Return

This function returns a ClmStatus object.

Error and Exception

When an overall function error occurs, this function returns a none SUCCESS value in the ClmJobStatus object. The ClmJob can be retrieved from the ClmJobStatusItem array.

If a system error prevents the operation from completing, a RemoteException is thrown.

readDevices

Synopsis

```
DeviceStatus readDevices(UserToken token, String[] dev_ids) throws RemoteException;
```

Description

This function uses the given device ID to retrieve an array of device objects from the inventory.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of device IDs.

Return

This function returns DeviceStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```
DeviceStatus status = readDevices(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
DeviceStatusItem[] items = status.getDeviceStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
Device device = items[i].getDevice();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();
```

```

// Get the individual error message corresponding to
// the object.
    String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an operation error occurs, a `DeviceStatus` object is returned with information about the error. To inspect the individual element status, you must traverse the `DeviceStatusItem` array within `DeviceStatus`. Each `DeviceStatusItem` contains the Device object, the error code, and the error message.

reCreateDevices

Synopsis

```

DeviceStatus reCreateDevices (UserToken token, Devices[] devs, String group,
DeviceAuthentication[] dev_auth_info, Device.TransportMethod[] transports) throws
RemoteException;

```

Description

This function allows the user to modify existing device objects and provides an opportunity to change device authentication information and setting a new transport method. Cisco License Manager tries to reconnect to the device using newly provided information and verifies that the changes are correct.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
devs	Array of Device, mandatory	Device objects	An array of device objects that already exist in the inventory. If the device object does not exist in the inventory, it is created and placed in the inventory.
group	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the device group.

Parameter	Type	Value	Description
dev_auth_info	Array of DeviceAuthentication, mandatory	—	An array of DeviceAuthentication values is used when communicating with discovered devices to get license information. Each entry in the array is used to connect to the IP address in the IP array. DeviceAuthentication contains username, password, and enable password.
transports	Device. TransportMethod array, mandatory	HTTP, Telnet, ssh	This parameter specifies which connection transport is used for each IP entry. Cisco License Manager assumes that the caller knows the connection method used for each device.

Return

This function returns DeviceStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

DeviceStatus status = reCreateDevices (...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
DeviceStatusItem[] items = status.getDeviceStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
Device device = items[i].getDevice();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an operation error occurs, a DeviceStatus object is returned with information about the error. To inspect the individual element status, you must traverse the DeviceStatusItem array within DeviceStatus. Each DeviceStatusItem contains the Device object, the error code, and the error message.

setAsyncOperationJobComment

Synopsis

Status setAsyncOperationJobComment(UserToken token, String job_id, String comment) throws RemoteException;

Description

This function allows you to set the comment in a job records that are associated with your username.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
job_id	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The Job ID. Each Job ID is the request ID that is returned from an asynchronous function call.
comment	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The comment that describes the job.

Return

This function returns the Status object. If the operation is not successful, Status returns the none ClmErrors.SUCCESS error code.

Error and Exception

When an overall function error occurs, this function returns none ClmErrors.SUCCESS in the Status.

If a system error prevents the operation from completing, a RemoteException is thrown.

writeDevices

Synopsis

IDStatus writeDevices(UserToken token, Device[] devices) throws RemoteException;

Description

This function writes the given device objects into the inventory. The input device objects can be new instances of devices returned by the createDevices() function or existing ones retrieved from the inventory by the readDevices() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
devices	Array of devices, mandatory	—	An array of device objects, which can be new instances of a device class or existing ones retrieved from the inventory by the readDevice() function.

Return

This function returns IDStatus objects. The following example shows the error code, error message, and returned objects in the status:

```

IDStatus status = writeDevices(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual ID returned by the operation.
String id = items[i].getID();

// Get the individual error code corresponding to
// the ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the ID.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.



CHAPTER 6

Folder Management Functions

This chapter provides information about the following folder management functions:

- [addPAKsToFolder](#), page 6-1
- [createFolder](#), page 6-2
- [deleteFolder](#), page 6-3
- [listAllFolders](#), page 6-3
- [removePAKsFromFolder](#), page 6-4
- [renameFolder](#), page 6-5

addPAKsToFolder

Synopsis

```
IDStatus addPAKsToFolder(UserToken token, String[] pak_ids, String folder) throws  
RemoteException;
```

Description

This function adds product authorization keys (PAKs) to the specified folder. You can add PAKs to more than one folder. If you want to keep a PAK in a single folder, you should remove the PAK from any other folder by calling the `removePAKsFromFolder()` function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Parameter	Type	Value	Description
pak_ids	String array, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs to be added to the folder.
folder	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder.

Return

This function returns an IDStatus object. The error code will be SUCCESS if the operation is successful. IDStatus contains an array of IDStatusItem. Each IDStatusItem contains the PAK ID, error code, and error message.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When a system-level error occurs and the operation cannot be completed, the IDStatus contains overall operation status. The IDStatus error code returns none SUCCESS and is accompanied by an error message if the cause of the failure is known. For each individual ID, IDStatus contains IDStatusItem[]. Each IDStatusItem contains an ID, error code, and error message that shows the individual status.

createFolder

Synopsis

```
Status createFolder(UserToken token, String folder) throws RemoteException;
```

Description

This function creates a new folder with the given name.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
folder	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder.

Return

This function returns the Status object with a SUCCESS error code.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the Status object with the error code.

deleteFolder

Synopsis

```
Status deleteFolder(UserToken token, String folder) throws RemoteException;
```

Description

This function deletes a folder from the system. PAKs in this folder are not deleted. These PAKs will be moved to the default folder if they do not already belong to another folder.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
folder	String, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder.

Return

This function returns the Status object with the SUCCESS error code.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the Status object with the error code.

listAllFolders

Synopsis

```
String[] listAllFolders(UserToken token) throws RemoteException;
```

Description

This function returns a list of all folders in the system.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns a string array of folder names.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

removePAKsFromFolder

Synopsis

```
IDStatus removePAKsFromFolder(UserToken token, String[] pak_ids, String folder) throws RemoteException;
```

Description

This function removes PAKs from the specified folder. If a PAK belongs to only one folder, the operation fails. A PAK has to belong to at least one folder. To delete the PAKs from inventory, use the deletePAKs() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_ids	String array, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs to be removed from the folder.
folder	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder.

Return

This function returns an IDStatus object. The error code will be SUCCESS if the operation is successful. IDStatus contains an array of IDStatusItem. Each IDStatusItem contains a PAK ID, error code, and error message.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When a system-level error occurs and the operation cannot be completed, the IDStatus contains information about the overall operation status. The IDStatus error code returns none SUCCESS accompanied by an error message if the cause of failure is known.

renameFolder

Synopsis

```
Status renameFolder(UserToken token, String folder, String new_name) throws
RemoteException;
```

Description

This function renames a given PAK folder. All PAKs contained in the folder will remain unchanged.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
folder	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder.
new_name	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The new name for the PAK folder.

Return

This function returns the Status object with the SUCCESS error code.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns the Status object with the error code.



CHAPTER 7

License Inventory Management Functions

This chapter provides information about the following license inventory management functions:

- [asyncAnnotateLicenses](#), page 7-1
- [asyncDeployLicenses](#), page 7-3
- [asyncObtainLicense](#), page 7-4
- [getLicensesOnDevice](#), page 7-6
- [getRehostInfo](#), page 7-7
- [initRehostLicense](#), page 7-7
- [listAllLicensesInPAK](#), page 7-8
- [obtainLicenseForRehost](#), page 7-9
- [readLicenses](#), page 7-10
- [rehostLicense](#), page 7-12
- [reObtainLicense](#), page 7-12
- [resendLicense](#), page 7-13
- [revokeLicenseForRehost](#), page 7-14
- [writeLicenses](#), page 7-15

asyncAnnotateLicenses

Synopsis

```
String asyncAnnotateLicenses(UserToken token, String[] lic_ids, String[] annotation, IDStatusListener listener) throws RemoteException;
```

Description

This function allows you to annotate licenses with comments that you provide.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements the `StatusListener` interface. When the operation is complete, the `onStatus()` method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
lic_ids	Array of string, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	ID of License objects.
annotation	Array of string, mandatory	Text string up to 99 characters.	Text of the annotation for each license. Cisco License Manager does not check the length of the annotation parameter. Cisco IOS software returns an error if it exceeds the character limit.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID string to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method.

The listener will receive the IDStatus, which includes a list of IDStatusItem. Each IDStatusItem contains the license ID and the error code and error message of the operation.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The input parameter of the onStatus() method in the IDStatusListener contains the error code and messages. More than one error code and message may be contained in the IDStatus object. The following example shows the error code and messages in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
```

```

// the object ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object ID.
String item_err_msg = items[i].getErrorMessage();
    }
}

```

asyncDeployLicenses

Synopsis

```
String asyncDeployLicenses(UserToken token, String[] lic_ids, IDStatusListener listener)
throws RemoteException;
```

Description

This function deploys the given licenses to their target devices.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements StatusListener interface. When the operation is complete, the onStatus() method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
lic_ids	String array, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	An array of license ID.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID string to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method.

The listener will receive the IDStatus, which includes a list of IDStatusItem. Each IDStatusItem contains the license ID and the error code and error message of the operation.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The input parameter of the `onStatus()` method in the `IDStatusListener` contains the error code and messages. More than one error code and message may be contained in the `IDStatus` object. The following example shows the error code and messages in the `onStatus()` method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();
    }
}
```

asyncObtainLicense

Synopsis

```
String asyncObtainLicense(UserToken token, LicenseRequest[] lic_req, boolean deploy,
IDStatusListener listener) throws RemoteException;
```

Description

This function downloads the information that is associated with the given product authorization key (PAK) IDs from the Cisco Product License Registration Portal and stores the information in the inventory. The first function only obtains the licenses; the second function obtains the licenses and deploys them.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements `StatusListener` interface. When the operation is complete, the `onStatus()` method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
lic_reqs	Array of LicenseRequest, mandatory	—	An array of LicenseRequest.
deploy	Boolean, mandatory	True, False	True to ask the server to deploy all licenses obtained.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID string to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method.

The listener will receive the IDStatus, which includes a list of IDStatusItem. Each IDStatusItem contains the PAKid + "-::-" + SKU name + "-::-" + udi, and the error code and error message of this operation..

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The input parameter of the onStatus() method in the IDStatusListener contains the error code and messages. More than one error code and message may be contained in the IDStatus object. The following example shows the error code and messages in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();

    }
}
```

getLicensesOnDevice

Synopsis

LicenseStatus getLicensesOnDevice(UserToken token, String dev_id) throws RemoteException;

Description

This function retrieves license information that resides on the given device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	The Device ID string.

Return

This function returns LicenseStatus objects. The following example shows the error code, message and returned objects in the status:

```
LicenseStatus status = getLicensesOnDevice(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicenseStatusItem[] items = status.getLicenseStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
License license = items[i].getLicense();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}
```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs for an element in the input array, the error code and error message is contained in the returned status object.

getRehostInfo

Synopsis

```
RehostInfoStatus getRehostInfo(UserToken token, String[] dev_ids) throws RemoteException;
```

Description

This function returns the RehostInfo of each given device. Each RehostInfo contains a rehost request and a permission ticket or a rehost ticket.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_ids	String array, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	An array of device IDs.

Return

This function returns the RehostInfoStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The RehostInfoStatus object contains the none SUCCESS error code and error message if an operation error occurs. Otherwise, you must traverse the RehostInfoStatusItem array to retrieve all of the RehostInfo objects.

initRehostLicense

Synopsis

```
Status initRehostLicense(UserToken token, RehostRequest rehost_req) throws RemoteException;
```

Description

The limitation of rehosting from the Cisco Product License Registration Portal is that there can be only one PermissionTicket acquired per device until a new license is obtained. This means that there is only one PermissionTicket and one RehostTicket per device at any time.

This function is the first step of the rehost process. The process consists of several steps, including getting a permission ticket from the Cisco Product License Registration Portal, retrieving the rehost ticket from the device, sending the rehost ticket to the Cisco Product License Registration Portal to obtain the license, and deploying the license to the destination device.

The obtained PermissionTicket is stored in local storage and is later used to revoke the license from the source device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
rehost_req	RehostRequest, mandatory	—	An object that represents the request.

Return

This function returns the Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

Status contains a not equals SUCCESS error code and an error message if the operation is not successful.

listAllLicensesInPAK

Synopsis

```
String[] listAllLicensesInPAK(UserToken token, String pak_id) throws RemoteException;
```

Description

This function returns an array of License IDs that belong to the given PAK.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_id	String, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	The PAK ID that contains the licenses.

Return

This function returns an array of License ID contained by the PAK. If the pak_id is invalid, this function returns null.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

obtainLicenseForRehost

Synopsis

```
IDStatus obtainLicenseForRehost(UserToken token, RehostRequest rehost_req) throws
RemoteException;
```

Description

The limitation of rehosting from the Cisco Product License Registration Portal is that there can be only one PermissionTicket acquired per device until a new license is obtained. This means that there is only one PermissionTicket and one RehostTicket per device at any time.

This function is the third step of the rehost process. The process consists of several steps, including getting a permission ticket from the Cisco Product License Registration Portal, retrieving the rehost ticket from the device, sending the rehost ticket to the Cisco Product License Registration Portal to obtain the license, and deploying the license to the destination device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
rehost_req	RehostRequest, mandatory	—	An object that represents the request.

Return

This function returns the `LicenseStatus` object and returns obtained license objects.

```
IDStatus status = obtainLicenseForRehost (...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicenseStatusItem[] items = status.getLicenseStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
License license = items[i].getLicense();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}
```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

The `Status` object contains a none `SUCCESS` error code and error message if an operation error occurs. Otherwise, you must traverse the `IDStatusItem` array to retrieve all of the license objects.

readLicenses

Synopsis

```
LicenseStatus readLicenses(UserToken token, String[] lic_ids) throws RemoteException;
```

Description

This function retrieves an array of license objects from the inventory using the given device IDs. If the `lic_ids` parameter is null, this function retrieves all license objects from the inventory.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
lic_ids	String array, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	An array of License ID.

Return

This function returns LicenseStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

LicenseStatus status = readLicenses(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicenseStatusItem[] items = status.getLicenseStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
License license = items[i].getLicense();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.

rehostLicense

Synopsis

```
Status rehostLicense(UserToken token, RehostRequest rehost_req) throws RemoteException;
```

Description

This function sends requests to rehost a license from one device to another. This process contains several steps, including retrieving a permission ticket from the Cisco Product License Registration Portal, retrieving a rehost ticket from the device, and sending the rehost ticket to the Cisco Product License Registration Portal to obtain a license and deploy the newly-obtained license to the destination device. These steps are encapsulated by this function as a single operation. The obtained license is stored in local storage and can be used later to be deployed to destination devices.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
rehost_req	RehostRequest, mandatory	—	An object that represents the request.

Return

This function returns the Status object, which contains the error code and error message. If the operation is successful, the ClmErrors.SUCCESS error code is returned. If it is unsuccessful, the none ClmErrors.SUCCESS error code and the error message are returned.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns false.

reObtainLicense

Synopsis

```
IDStatus reObtainLicense(UserToken token, String dev_id) throws RemoteException;
```

Description

This function requests that the Cisco Product License Registration Portal resend the license. After licenses are received, it updates and synchronizes Cisco License Manager data storage. It does not deploy licenses to a device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	ID of device to which to resend the license.

Return

This function returns the IDStatus object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, it will return none SUCCESS error code and error message in the IDStatus object. Otherwise, it returns a SUCCESS error number in the IDStatus object and you go through the IDStatusItem array to get all the licenseline IDs that are reobtained.

resendLicense

Synopsis

```
Status resendLicense(UserToken token, String dev_id) throws RemoteException;
```

Description

This function resends licenses to a device to restore corrupted license files. The function requests all licenses that have been obtained from the Cisco Product License Registration Portal, saves them into the License Manager database, and then redeploys them to the device.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	ID of device to which to resend the license.

Return

This function returns the Status object, which contains the error code and error message. If the operation is successful, the CImError.SUCCESS error code is returned.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns false.

revokeLicenseForRehost

Synopsis

```
Status revokeLicenseForRehost (UserToken token, RehostRequest rehost_req) throws
RemoteException;
```

Description

This function can be used to handle the situation that rehost licenses failed in the middle of the task. It should be called only when PermissionTicket has been successfully obtained and stored in the Cisco License Manager inventory. The limitation of rehosting from the Cisco Product License Registration Portal is that there can be only one PermissionTicket acquired per device until a new license is obtained. This means that there is only one PermissionTicket and one RehostTicket per device at any time.

This function is the second step of the rehost process. The process consists of several steps, including retrieving a permission ticket from the Cisco Product License Registration Portal, retrieving the rehost ticket from the device, sending the rehost ticket to the Cisco Product License Registration Portal to obtain the license, and deploying the license to the destination device.

The obtained PermissionTicket is stored in local storage and is later used to revoke the license from the source device. It is removed if the revoke operation is successful, and the RehostTicket is stored in local storage for next step of rehost process.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
rehost_req	RehostRequest, mandatory	—	An object that represents the request.

Return

This function returns the Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

Status contains an error code not equals SUCCESS and an error message if the operation is not successful.

writeLicenses

Synopsis

```
IDStatus writeLicenses(UserToken token, License[] lics) throws RemoteException;
```

Description

This function writes the input license objects into the inventory. The inputLicense objects are existing ones retrieved from the inventory by the function readLicenses().

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents the user's authorization pass, which is obtained after the user invokes the login function and gets authenticated by the back-end server.
lics	Array of License, mandatory	License object with a valid license ID. ID is a string containing up to 64 ASCII characters in the range from x21 to x7A.	An array of License objects.

Return

This function returns IDStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```
IDStatus status = writeLicenses(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual ID returned by the operation.
String id = items[i].getID();

// Get the individual error code corresponding to
// the ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the ID.
String item_err_msg = items[i].getErrorMessage();
}
```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, an IDStatus object is returned with information about the error.



CHAPTER 8

License Line Management Functions

This chapter provides information about the following license line management functions:

- [asyncAnnotateLicenseLines](#), page 8-1
- [asyncDeployLicenseLines](#), page 8-3
- [getLicenseLinesByLicense](#), page 8-4
- [getLicenseLinesOnDevice](#), page 8-5
- [listExpiredLicenseLines](#), page 8-6
- [readLicenseLines](#), page 8-7
- [writeLicenseLines](#), page 8-8

asyncAnnotateLicenseLines

Synopsis

```
String asyncAnnotateLicenseLines(UserToken token, String[] licline_ids, String[]  
annotation, IDStatusListener listener) throws RemoteException;
```

Description

This function allows you to annotate license lines with comments.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements `StatusListener` interface. When the function is completed, the `onStatus()` method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
licline_ids	Array of string, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	ID of license line objects.
annotation	Array of string, mandatory	Text string up to 99 characters	Text of the annotation for each license line. Cisco License Manager does not check the length of the annotation parameter. Cisco IOS returns an error if the character limit is exceeded.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID string to the caller. When the operation is complete, the status is provided as the input parameter of onStatus() method. The listener will receive the IDStatus , which includes a list of IDStatusItem. Each IDStatusItem contains the licenseline ID and the error code and error message of the operation.

The following example shows the error code and message in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();

    }
}
```

```
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

The input parameter of the `onStatus()` method in the `IDStatusListener` contains the error code and messages. More than one error code and message may be contained in the `IDStatus` object.

asyncDeployLicenseLines

Synopsis

```
String asyncDeployLicenseLines(UserToken token, String[] licline_ids, IDStatusListener listener) throws RemoteException;
```

Description

This function deploys the given license lines to their target devices.

This function is nonblocking and returns a request ID to the caller immediately. While calling this function, the client program provides a listener object that implements the `StatusListener` interface. When the function is completed, the `onStatus()` method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
licline_ids	Array of string, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	An array of license line ID.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID string to the caller. When the operation is complete, the status is provided as the input parameter of `onStatus()` method. The listener will receive the `IDStatus`, which includes a list of `IDStatusItem`. Each `IDStatusItem` contains the licenseline ID and the error code and error message of the operation.

The following example shows the error code and message in the `onStatus()` method:

```
public void onStatus(IDStatus status) {
    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
}
```

```

String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object ID returned by the operation.
String id = items[i].getID();

// Get the individual error code corresponding to
// the object ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object ID.
String item_err_msg = items[i].getErrorMessage();
}
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

The input parameter of the `onStatus()` method in the `IDStatusListener` contains the error code and messages. More than one error code and message may be contained in the `IDStatus` object.

getLicenseLinesByLicense

Synopsis

`LicLineStatus getLicenseLinesByLicense(UserToken token, String lic_id)` throws `RemoteException`;

Description

This function retrieves license lines, which are contained in the given license.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
lic_id	String, mandatory	—	The license ID.

Return

This function returns `LicLineStatus` objects. The following example shows the error code, messages and returned objects in the status:

```
LicLineStatus status = getLicenseLinesByLicense (... , ...);
```

```

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicLineStatusItem[] items = status.getLicLineStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
LicenseLine liclines = items[i].getLicenseLine();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs for an element in the input array, the error code and error message are contained in the returned status object.

getLicenseLinesOnDevice

Synopsis

```
LicLineStatus getLicenseLinesOnDevice(UserToken token, String dev_id) throws
RemoteException;
```

Description

This function retrieves license lines that are contained in the given device ID.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	—	The device ID.

Return

This function returns `LicLineStatus` objects. The following example shows the error code, message, and returned objects in the status:

```

LicLineStatus status = getLicenseLinesOnDevice (... , ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicLineStatusItem[] items = status.getLicLineStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

    // Get the individual object returned by the operation.
    LicenseLine liclines = items[i].getLicenseLine();

    // Get the individual error code corresponding to
    // the object.
    int item_err_code = items[i].getErrorCode();

    // Get the individual error message corresponding to
    // the object.
    String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs for an element in the input array, information about the error is contained in the returned status object.

listExpiredLicenseLines

Synopsis

```
HashSet<Expired License> listExpiredLicenseLines (UserToken token) throws RemoteException;
```

Description

This function returns a list of license lines that have expired.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns `HashSet<ExpiredLicense>` objects. The following example shows the error code, message, and returned objects in the status:

```
HashSet<ExpiredLicense> _hash = listExpiredLicenseLines (... , ...);
```

```

    If(_hash==null || _hash.size()==0){
        System.out.println(" No Expired License found. ");
    }

    // Iterate through the list to get individual status.
    Iterator licIt = _hash.iterator();

    while(licIt.hasNext()) {
        ExpiredLicense _lic = (ExpiredLicense)licIt.next();
        String[] _licLine_ids=_lic.getLicLineIDs();
        String _dev_id=_lic.getDeviceID();
        String _feature=_lic.getFeatureName();
        String _version=_lic.getVersion();
    }

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

This function returns a set of `ExpiredLicense` objects. It returns null when an error occurs.

readLicenseLines

Synopsis

```

LicLineStatus readLicenseLines(UserToken token, String[] licline_ids) throws
RemoteException;

```

Description

This function retrieves an array of license line objects from the inventory using the given license line IDs.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
licline_ids	Array of string, mandatory	ID is a string containing up to 256 ASCII characters in the range from x21 to x7A	An array of license line IDs.

Return

This function returns `LicLineStatus` objects. The following example shows the error code, messages, and returned objects in the status:

```

LicLineStatus status = readLicenseLines(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

```

```

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
LicLineStatusItem[] items = status.getLicLineStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
LicenseLine license = items[i].getLicenseLine();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs for an element in the input array, information about the error is contained in the returned status object.

writeLicenseLines

Synopsis

```
IDStatus writeLicenseLines(UserToken token, LicenseLine[] liclines) throws
RemoteException;
```

Description

This function writes the given license line objects into the inventory. The input license line objects are existing ones retrieved from the inventory by the `readLicenseLines()` function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
licline	Array of LicenseLine, mandatory	License line object with a valid ID. ID is a string containing up to 256 ASCII characters in the range from x21 to x7A.	An array of license line objects.

Return

This function returns IDStatus objects. The following example shows the error code, messages and returned objects in the status:

```

IDStatus status = writeLicenseLines(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

    // Get the individual ID returned by the operation.
    String id = items[i].getID();

    // Get the individual error code corresponding to
    // the ID.
    int item_err_code = items[i].getErrorCode();

    // Get the individual error message corresponding to
    // the ID.
    String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs for an element in the input array, information about the error is contained in the returned status object.



CHAPTER 9

License Manager Class Constructors

This chapter provides information about the following Cisco License Manager class constructors:

- [LicenseManager \(Default\), page 9-1](#) (default constructor)
- [LicenseManager \(Nondefault\), page 9-1](#) (constructor using nondefault callback port)

LicenseManager (Default)

Synopsis

```
public LicenseManager() throws RemoteException;
```

Description

This is the default constructor. When the default constructor is used to create an instance of `LicenseManager`, TCP/IP port 1198 is used for the client-side callback function to receive results of asynchronous calls. If port 1198 is not available, the next available port in the range from 1198 to 1297 is used.

Input Parameters

Not applicable.

Return

Not applicable.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

Code

See [Appendix A, “Sample Client Program in Java”](#) for a sample of the code.

LicenseManager (Nondefault)

Synopsis

```
public LicenseManager(int callback_port) throws RemoteException;
```

Description

If you like to specify non-default port to be used for client-side callback function, you can use this constructor and specify the port to be used.

Input Parameters

Parameter	Type	Value	Description
callback_port	Integer, mandatory	Range from 1024 to 65535	The port on which the client-side callback function receives results of asynchronous calls.

Return

Not applicable.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.



CHAPTER 10

Notification Handling Functions

This chapter provides information about the following notification handling functions:

- [deregisterNotificationListener](#), page 10-1
- [disableNotificationByEmail](#), page 10-2
- [enableNotificationByEmail](#), page 10-2
- [registerNotificationListener](#), page 10-3

deregisterNotificationListener

Synopsis

```
boolean deregisterNotificationListener(UserToken token, NotificationListener listener)  
throws RemoteException;
```

Description

This function removes the notification listener from the client program.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
listener	Notification Listener interface, mandatory	—	An object that was registered previously by addNotificationListener().

Return

This function returns true if the deregistration is successful and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns false.

disableNotificationByEmail

Synopsis

```
boolean disableNotificationByEmail(UserToken token) throws RemoteException;
```

Description

This function disables sending e-mail notifications to the user.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns true if it is successful and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns false.

enableNotificationByEmail

Synopsis

```
boolean enableNotificationByEmail(UserToken token) throws RemoteException;
```

Description

This function enables sending e-mail notifications to the user.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns true if it is successful and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns false.

registerNotificationListener

Synopsis

```
boolean registerNotificationListener(UserToken token, NotificationListener listener)
throws RemoteException;
```

Description

This function allows a client program to register a listener for notifications. When a notification has occurred, the `onNotification()` method in the listener object is invoked.

The client must use the `registerNotification()` application programming interface (API) in order to be able to receive notification. The client should implement the `NotificationListener` interface in order to be able to take action upon notification.

```
package com.cisco.nm.clm.common;
public interface NotificationListener {
    public void onNotification(Notification notification);
}
```

Clients should implement the `NotificationListener` interface as follows.

```
public class MyClient implements NotificationListener {
    //other fields and functions to do client's job

    //the action that will be taken on notification.
    public void onNotification(Notification notify) {
        System.out.println(notify.toString());
    }
}
```

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
listener	Notification Listener interface, mandatory	—	An object that implements the <code>NotificationListener</code> interface.

Return

This function returns true if the registration is successful and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns false.



CHAPTER 11

PAK Management Functions

This chapter provides information about the following product authorization key (PAK) management functions:

- [asyncDownloadPAKInfo](#), page 11-1
- [createPAKs](#), page 11-3
- [deletePAKs](#), page 11-4
- [listAllFoldersByPAK](#), page 11-6
- [listAllPAKsInFolder](#), page 11-6
- [listPAKContainFeatures](#), page 11-7
- [readPAKs](#), page 11-8
- [writePAKs](#), page 11-9

asyncDownloadPAKInfo

Synopsis

```
String asyncDownloadPAKInfo(UserToken token, String[] pak_ids, IDStatusListener listener)  
throws RemoteException;
```

Description

This function downloads the information that is associated with the given PAK IDs from the Cisco Product License Registration Portal and stores the information in the inventory.

This function is nonblocking and returns a request ID to the caller immediately. When calling this function, the client program provides a listener object that implements the `StatusListener` interface. When the function is completed, the `onStatus()` method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs that identify interested PAKs. A PAK ID is generated by using the createPAK function, passing a PAK name as a parameter to get the PAK ID.
listener	IDStatusListener object, mandatory	—	An object which implements IDStatusListener interface.

Return

This function returns a request ID to the caller. When the operation is complete, the status is provided as the input parameter of the onStatus() method. The listener received IDStatus upon operation complete. The IDStatus contains a list of IDStatusItem, which includes the PAK ID and the error code and error message of the operation in this PAK ID.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

The input parameter of the onStatus() method in IDStatusListener contains the error code and messages. More than one error code and message may be contained in the IDStatus object. The following example shows the error code and messages in the onStatus() method:

```
public void onStatus(String request_id, IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();
```

```
    }
}
```

createPAKs

Synopsis

```
PAKStatus createPAKs(UserToken token, String[] pak_ids) throws RemoteException;
```

```
PAKStatus createPAKs(UserToken token, String[] pak_ids, String folder) throws
RemoteException;
```

```
PAKStatus createPAKs(UserToken token, PAK[] paks) throws RemoteException;
```

```
PAKStatus createPAKs(UserToken token, PAK[] paks, String folder) throws RemoteException;
```

Description

This function creates PAK objects in the inventory. The input array of `pak_ids` is used as the name for each instance of the PAK object. If the input parameter `folder` is provided, the newly created PAK is added to the folder.

The third and fourth forms of the function take the array of PAK objects as the input parameter. These two functions allow you to create a PAK object and fill in the attributes.

Input Parameters

Parameter	Type	Value	Description
<code>token</code>	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
<code>pak_ids</code>	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs, used as Name for each instance of PAK object.
<code>folder</code>	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The name of the folder. Note Some forms of this function do not require a folder name.
<code>paks</code>	Array of PAK, mandatory	PAK object	An array of PAK objects.

Return

This function returns PAKStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```
PAKStatus status = createPAKs(..., ...);
```

```

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
PAKStatusItem[] items = status.getPAKStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
PAK pak = items[i].getPAK();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.

deletePAKs

Synopsis

```
IDStatus deletePAKs(UserToken token, String[] pak_ids) throws RemoteException;
```

Description

This function deletes PAK objects from the inventory using the given PAK IDs. The PAK IDs are obtained from the `createPAKs` or `readPAKs` functions. This function also removes a PAK from the folders that it belongs to.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs.

Return

This function returns IDStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

IDStatus status = deletePAKs(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual ID returned by the operation.
String id = items[i].getID();

// Get the individual error code corresponding to
// the ID.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the ID.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.

listAllFoldersByPAK

Synopsis

```
String[] listAllFoldersByPAK(UserToken token, String[] pak_id) throws RemoteException;
```

Description

This function returns a list of all folders to which the PAK belongs.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_id	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs, which is used as the ID for each instance of the PAK object.

Return

This function returns an array of folders to which the given PAK belongs. If the PAK does not exist, a non-null array of size zero is returned.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

listAllPAKsInFolder

Synopsis

```
String[] listAllPAKsInFolder(UserToken token, String folder) throws RemoteException;
```

Description

This function returns an array of PAK IDs that belong to the given folder. If the input folder is null, this function returns an array of PAK IDs in the default folder.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
folder	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	Name of the PAK folder, or null if the function is looking for ungrouped PAKs.

Return

This function returns an array of PAK IDs in the specified folder. If the folder does not exist, it returns null. If the folder is empty, it returns a non-null array of size zero.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

listPAKContainFeatures

Synopsis

```
String[] listPAKContainFeatures(UserToken token, String[] features) throws
RemoteException;
```

Description

This function returns a list of PAKs that contains all the license features in the given feature list.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
features	Array of string, mandatory	Features is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of string that represents the license feature names. The array of string is an AND relationship.

Return

This function returns an array of string. Each string is a PAK ID.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When error occurs, this function returns null. An array length of zero means no PAK was found.

readPAKs

Synopsis

```
PAKStatus readPAKs(UserToken token, String[] pak_ids) throws RemoteException;
```

Description

This function retrieves an array of PAK objects from the inventory using the given PAK IDs. PAK IDs are obtained from the creatPAKs function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
pak_ids	Array of string, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	An array of PAK IDs.

Return

This function returns PAKStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

PAKStatus status = readPAKs(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
PAKStatusItem[] items = status.getPAKStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

// Get the individual object returned by the operation.
PAK pak = items[i].getPAK();

// Get the individual error code corresponding to
// the object.
int item_err_code = items[i].getErrorCode();

// Get the individual error message corresponding to
// the object.
String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.

writePAKs

Synopsis

```
IDStatus writePAKs(UserToken token, PAK[] pak) throws RemoteException;
```

Description

This function writes the given PAK objects into the inventory. The input PAK objects can be new instances of PAK returned by the createPAKs() function or existing ones retrieved from the inventory by the readPAKs() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
paks	Array of PAK, mandatory	PAK object	An array of PAK objects.

Return

This function returns IDStatus objects. The following example shows the error code, error messages, and returned objects in the status:

```

IDStatus status = writePAK(..., ...);

// The general error code of the operation.
int err_code = status.getErrorCode();

// The general error message of the operation.
String err_msg = status.getErrorMessage();

// A list of status for each individual element in the
// bulk operation.
IDStatusItem[] items = status.getIDStatusItems()

// Iterate through the list to get individual status.
for (int i = 0; i < items.length(); i++) {

    // Get the individual ID returned by the operation.
    String id = items[i].getID();

    // Get the individual error code corresponding to
    // the ID.
    int item_err_code = items[i].getErrorCode();

    // Get the individual error message corresponding to
    // the ID.
    String item_err_msg = items[i].getErrorMessage();
}

```

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs on an element in the input array, a status object is returned with information about the error.



CHAPTER 12

Policy Management Functions

This chapter provides information about the following policy management functions:

- [asyncExecutePolicy](#), page 12-1
- [createPolicy](#), page 12-3
- [deletePolicy](#), page 12-4
- [enumerateDeviceFilterAttribute](#), page 12-4
- [enumerateSKUFilterAttribute](#), page 12-5
- [listAllPolicies](#), page 12-6
- [listFilteredDevices](#), page 12-6
- [listFilteredSKUs](#), page 12-7
- [readPolicy](#), page 12-7
- [writePolicy](#), page 12-8

asyncExecutePolicy

Synopsis

```
String asyncExecutePolicy(UserToken token, String policy_id, IDStatusListener listener);
```

Description

This function executes the given policy by using the policy filters to select devices and stock-keeping units (SKUs) and obtain and deploy licenses for those devices.

This function is nonblocking and returns a request ID to the caller immediately. When calling this function, a client program provides a listener object that implements StatusListener interface. When the function is completed, the onStatus() method in the listener object is invoked.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
policy_id	String, mandatory	ID is a string containing up to 64 ASCII characters in the range from x21 to x7A	ID of the Policy object.
listener	IDStatusListener, mandatory	—	The listener object.

Return

This function returns a request ID to the caller. When the operation is complete, the status is provided as the input parameter of onStatus() method.

The listener will receive the IDStatus, which includes a list of IDStatusItem. Each IDStatusItem contains the PAK ID + "-:-" + SKU name + "-:-" + UDI, and the error code and error message of the operation.

Error and Exception

The input parameter of the onStatus() method in the IDStatusListener contains the error code and messages. More than one error code and message may be contained in the IDStatus object. Each ID in the status object represents a device ID that participates in the execution of the policy.

The following example shows the error code and message in the onStatus() method:

```
public void onStatus(IDStatus status) {

    // The general error code of the operation.
    int err_code = status.getErrorCode();

    // The general error message of the operation.
    String err_msg = status.getErrorMessage();

    // A list of status for each individual element in the
    // bulk operation.
    IDStatusItem[] items = status.getIDStatusItems()

    // Iterate through the list to get individual status.
    for (int i = 0; i < items.length(); i++) {

        // Get the individual object ID returned by the
        // operation.
        String id = items[i].getID();

        // Get the individual error code corresponding to
        // the object ID.
        int item_err_code = items[i].getErrorCode();

        // Get the individual error message corresponding to
        // the object ID.
        String item_err_msg = items[i].getErrorMessage();
    }
}
```

```

    }
}

```

createPolicy

Synopsis

```

PolicyStatus createPolicy(UserToken token, String policy_name, SKUFilter sku_filter,
DeviceFilter dev_filter);

```

Description

This function creates a Policy object that represents a set of rules for selecting SKUs and devices from the inventory.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
policy_name	String, mandatory	—	Name of the Policy object.
sku_filter	SKUFilter, optional	—	The input SKUFilter object specifies the criteria for searching the devices that match the criteria. If the filter is set to null, filtering is not performed.
dev_filter	DeviceFilter, optional	—	The input DeviceFilter object specifies the criteria for searching the devices that match the criteria. If the filter is set to null, filtering is not performed.

Return

This function returns a PolicyStatus object. The Status and the Policy object are contained in PolicyStatus. The definition of PolicyStatus and Status class is as follows:

```

public class Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
}

public class PolicyStatusItem extends Status implements Serializable {
    private Policy m_policy;

    public Policy getPolicy() {
        return m_policy;
    }

    public void setPolicy(Policy policy) {
        this.m_policy = policy;
    }
}

```

Error and Exception

When an error occurs, the error code is contained in the PolicyStatus object.

deletePolicy

Synopsis

```
Status deletePolicy(UserToken token, String policy_name);
```

Description

This function deletes from the system the policy object identified by the given name.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
policy_name	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	Name of the Policy object.

Return

This function returns Status objects.

Error and Exception

When an error occurs, the information is contained in the returned status object.

enumerateDeviceFilterAttribute

Synopsis

```
String[] enumerateDeviceFilterAttribute(UserToken token, Policy.DeviceAttribute attrib);
```

Description

This function returns a list of all possible values for the given attribute in DeviceFilter.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
attrib	Policy.DeviceAttribute, mandatory	—	The attribute name can be one of the following: <ul style="list-style-type: none"> Policy.DeviceAttribute.MODEL Policy.DeviceAttribute.GROUP

Return

This function returns an array of string representing all possible values for the attribute.

Error and Exception

When an error occurs, this function returns null.

enumerateSKUFilterAttribute

Synopsis

```
String[] enumerateSKUFilterAttribute(UserToken token, Policy.SKUAttribute attrib);
```

Description

This function returns a list of all possible values for the given attribute in SKUFilter.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
attrib	Policy.SKUAttribute, mandatory	—	The attribute name can be Policy.SKUAttribute.FEATURE_NAME.

Return

This function returns an array of string representing all possible values for the attribute.

Error and Exception

When an error occurs, this function returns null.

listAllPolicies

Synopsis

```
String[] listAllPolicies(UserToken token);
```

Description

This function returns a list of all policies that can be accessed by the user, including both the private policies owned by this user and all public policies.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns an array of string representing the IDs of the policies.

Error and Exception

When an error occurs, this function returns null.

listFilteredDevices

Synopsis

```
String[] listFilteredDevices(UserToken token, DeviceFilter dev_filter);
```

Description

This function generates a list of devices obtained by running the given device filter.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_filter	DeviceFilter, mandatory	—	The input DeviceFilter object specifies the criteria for searching the list of devices.

Return

This function returns an array of string representing the IDs of the devices.

Error and Exception

When an error occurs, this function returns null.

listFilteredSKUs

Synopsis

```
SKUIdentifier[] listFilteredSKUs(UserToken token, SKUFilter sku_filter);
```

Description

This function returns a list of SKU identifiers resulting from running the SKU filter in the policy. The SKUIdentifier class used in this function is defined as follows:

```
class SKUIdentifier {
    String pak;                // PAK of the selected SKU
    String sku_name;          // name the selected SKU
}
```

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
sku_filter	SKUFilter, mandatory	—	The input SKUFilter object specifies the criteria for searching the list of devices.

Return

This function returns an array of SKUIdentifier objects.

Error and Exception

When an error occurs, this function returns null.

readPolicy

Synopsis

```
PolicyStatus readPolicy(UserToken token, String policy_name);
```

Description

This function retrieves a Policy object from data storage using the given name.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
policy_name	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	Name of the Policy object.

Return

This function returns a PolicyStatus object. The status and the policy object are contained in PolicyStatus.

Error and Exception

When an error occurs, the error code is contained in the PolicyStatus object.

writePolicy

Synopsis

```
Status writePolicy(UserToken token, Policy policy);
```

Description

This function writes the given Policy objects into data storage. The input Policy objects can be new instances of Policy returned by the createPolicy() function or existing ones retrieved from the data storage by the readPolicy() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
policy	Policy, mandatory	—	Policy objects, which can be new instances of Policy class or existing ones retrieved from the inventory by the readPolicy() function.

Return

This function returns Status objects.

Error and Exception

When an error occurs, the information is contained in the returned status object.



CHAPTER 13

Report Management Functions

This chapter provides information about the following report management functions:

- [generateReport](#), page 13-1
- [readReport](#), page 13-2

generateReport

Synopsis

Status `generateReport(UserToken token, ReportSubject subject, DeviceFilter filter)` throws `RemoteException`;

Description

This function generates the specified report and stores the report data at the server. If prior report data exists, a call to this function overwrites the previous report. Only one copy of the report data is stored. This function may take a long time to perform. Only one report can be generated at a time.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server. Only an administrator can generate a report.
subject	ReportSubject, mandatory	One of the enumeration values defined in ReportSubject	Enumeration value that represents the subject of the report to be generated.
filter	DeviceFilter, optional	—	The input DeviceFilter object specifies the criteria for searching the defined devices. If the filter is set to null, no filtering is performed. For an audit trail, null should be entered.

Return

This function returns Status objects.

Error and Exception

When an error occurs, information about the error is contained in the returned status object. When a call fails because of a Remote Method Invocation (RMI)-related error, a RemoteException is thrown.

readReport

Synopsis

```
ReportStatus readReport(UserToken token, ReportSubject subject, OutputFormat
output_format) throws RemoteException;
```

Description

This function returns the report most recently generated by the generateReport() function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
subject	ReportSubject, mandatory	One of the enumeration values defined in ReportSubject	Enumeration value that represents the subject of the report to be generated.
output_format	OutputFormat, mandatory	HTML, text	Enumeration value that represents HTML or text format.

Return

This function returns a ReportStatus object containing the report, the error code, and the error message.

Error and Exception

When an error occurs, this function returns ReportStatus which contains the none ClmErrors.SUCCESS error code. When the call fails because of a Remote Method Invocation (RMI)-related error, a RemoteException is thrown.



CHAPTER 14

Scheduler Management Functions

This chapter provides information about the following scheduler management functions:

- [listScheduler](#), page 14-1
- [registerScheduler](#), page 14-2
- [unregisterScheduler](#), page 14-2

listScheduler

Synopsis

```
Schedule[] listScheduler(UserToken token) throws RemoteException;
```

Description

This is a simple scheduling function that allows you to list a scheduled task from the scheduler.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns an array of schedule objects if a schedule has been set. It returns null if a schedule has not been set.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

registerScheduler

Synopsis

```
Status registerScheduler(UserToken token, Schedule.ClmTask task, Schedule execute_time)
throws RemoteException
```

Description

This is a simple scheduling function that allows you to schedule the time, frequency, and specific Cisco License Manager action to be performed.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
task	Schedule.ClmTask, mandatory	—	Schedule.ClmTask object containing a Cisco License Manager-supported task string to be executed through the scheduler.
Execute_time	Schedule, mandatory	—	Schedule object that contains when to execute and frequency of execution.

Return

This function returns a Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, the Status object returns none SUCCESS error code and error message.

unregisterScheduler

Synopsis

```
Status unregisterScheduler(UserToken token, Schedule.ClmTask taskName) throws
RemoteException;
```

Description

This is a simple scheduling function that allows you to remove a scheduled task from the scheduler.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
taskName	Schedule.ClmTask, mandatory	—	Schedule.ClmTask object containing a Cisco License Manager-supported task string to be executed through the scheduler.

Return

This function returns a Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, the Status object returns none SUCCESS error code and error message.



CHAPTER 15

Troubleshooting Utility Functions

This chapter provides information about the following troubleshooting utility functions:

- [checkCiscoPortalConnection](#), page 15-1
- [checkDeviceConnection](#), page 15-1

checkCiscoPortalConnection

Synopsis

```
Status checkCiscoPortalConnection(UserToken token) throws RemoteException;
```

Description

This function checks the health of the Cisco Product License Registration Portal connection.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns a Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, the Status object returns none SUCCESS error code and error message.

checkDeviceConnection

Synopsis

```
Status checkDeviceConnection(UserToken token, String dev_id) throws RemoteException
```

Description

This function checks the health of a device connection.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
dev_id	String, mandatory	Name is a string containing up to 64 ASCII characters in the range from x21 to x7A	The device ID where the connection is to be checked.

Return

This function returns a Status object.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, the Status object returns none SUCCESS error code and error message.



CHAPTER 16

User Login and Logout Functions

This chapter provides information about the following user login and logout functions:

- [checkUserEULAStatus](#), page 16-1
- [login](#), page 16-2
- [logout](#), page 16-3

checkUserEULAStatus

Synopsis

```
EulaInfo checkUserEULAStatus(String user, String server_host, int port)  
throws RemoteException;
```

Description

This function checks to see if the user has already signed an end-user license agreement (EULA).

Input Parameters

Parameter	Type	Value	Description
username	String, mandatory	—	The username.
server_host	String, mandatory	Hostname or IP address	The hostname or IP address where the Cisco License Manager server is running. If the Cisco License Manager server and the client program are running on the same host, this argument can be set to null.
port	Integer, mandatory	Range from 1024 to 65535	The port on which the Cisco License Manager server is serving clients. If the Cisco License Manager server and the client program are running on the same host, this argument can be set to zero.

Return

This function returns a `EulaInfo` object. This object contains EULA information (such as whether EULA is signed, when it was signed, and so forth).

Error and Exception

When an error occurs, this function returns null.

If a system error prevents the operation from completing, a `RemoteException` is thrown.

login

Synopsis

```
UserToken login(String user, String password, String server_host, int port, int
idle_timeout, EulaInfo eula) throws RemoteException;
```

Description

Use this function to log into the system and acquire a `UserToken` that gives you authorization on subsequent function calls. A user can only log into the system one session at a time. A second login by the same user automatically logs out the previous login session.

Input Parameters

Parameter	Type	Value	Description
username	String, mandatory	—	The username.
password	String, mandatory	—	The user password.
server_host	String, mandatory	Hostname or IP address	The hostname or IP address where the Cisco License Manager server is running. If the Cisco License Manager server and the client program are running on the same host, this argument can be set to null.
port	Integer, mandatory	Range from 0 to 65535	The port on which the Cisco License Manager server is serving clients. If the Cisco License Manager server and the client program are running on the same host, this argument can be set to zero.
idle_timeout	Integer, mandatory	Range from 0 to 2147483657	Time interval, in minutes, allowed for idling before the token expires. When it is set to 0, the token never expires.
eula	EulaInfo, mandatory	—	The <code>EulaInfo</code> object contains whether EULA is accepted and whether to ask for EULA again in the next session.

Return

This function returns a `UserToken` object that represents the user's authorization pass. If the EULA is not accepted, the authorization pass is rejected.

Error and Exception

When an error occurs, this function returns null.

If a system error prevents the operation from completing, a `RemoteException` is thrown.

logout

Synopsis

```
void logout(UserToken token) throws RemoteException;
```

Description

This function logs the user out of the system and deletes the UserToken object.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

None.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.



CHAPTER 17

User Management Functions

This chapter provides information about the following user management functions:

- [createUser](#), page 17-1
- [deleteUser](#), page 17-2
- [listAllUsers](#), page 17-3
- [readUserInfo](#), page 17-3
- [writeUserInfo](#), page 17-4

createUser

Synopsis

```
UserInfo createUser(UserToken token, String username, String password, Role role) throws  
RemoteException;
```

Description

This function creates a new user. The user must log in using the administrator role to have permission to invoke this function. If a role is not entered, a default role of REPORTMGR is assigned.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
username	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The username.

Parameter	Type	Value	Description
password	String, mandatory	Up to 64 ASCII characters in the range from x21 to x7A	The user password.
role	Role, mandatory	Enumeration value	Role is an enumeration value for the given user.

Return

This function returns a UserInfo object if the operation is successful.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

deleteUser

Synopsis

```
boolean deleteUser(UserToken token, String username) throws RemoteException;
```

Description

This function deletes a user from the system. The user must log in using the administrator role to have permission to invoke this function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
username	String, mandatory	—	The username.

Return

This function returns true if the operation is successful, and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns false.

listAllUsers

Synopsis

```
String[] listAllUsers(UserToken token) throws RemoteException;
```

Description

This function returns a list of usernames in the system. The user must log in using the administrator role to have permission to invoke this function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns a string array representing the names of users.

Error and Exception

If a system error prevents the operation from completing, a RemoteException is thrown.

When an error occurs, this function returns null.

readUserInfo

Synopsis

```
UserInfo readUserInfo(UserToken token, String username) throws RemoteException;
```

Description

This function retrieves user information. If token is the only input, it retrieves the user information associated with token. If both token and username are present, it retrieves the information associated with username. The latter can only be called by a user with the Administrator role.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.

Return

This function returns UserInfo objects.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns null.

writeUserInfo

Synopsis

```
boolean writeUserInfo(UserToken token, UserInfo user) throws RemoteException;
```

Description

This function writes the given `UserInfo` object into the inventory. The input `UserInfo` object must exist and be retrieved from the inventory by the `readUserInfo()` function.

Input Parameters

Parameter	Type	Value	Description
token	UserToken, mandatory	—	A token that represents your authorization pass, which is obtained after you invoke the login function and are authenticated by the back-end server.
user	UserInfo object, mandatory	—	UserInfo object.

Return

This function returns true if the operation is successful and false if it is unsuccessful.

Error and Exception

If a system error prevents the operation from completing, a `RemoteException` is thrown.

When an error occurs, this function returns false.



APPENDIX **A**

Sample Client Program in Java

The following text is a client program in Java that completes the basic functions in Cisco License Manager. When you connect to Cisco License Manager, you must provide the following information:

- IP address of Cisco License Manager host
- Login name, password, port, and prompt

```
package com.cisco.nm.clm.sdk;

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

import java.rmi.RemoteException;

public class CLMClientSample {

    // Listener for asynchronous calls to receiving IDStatus.
    public class MyIDStatusListener implements IDStatusListener {
        public void onStatus(String request_id, IDStatus status) {
            System.out.println("Received status for request " + request_id);
        }
    }

    public static void main(String[] args) {

        .
        .
        .

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

        try {
            // login to the LM server located in the same host
            // with no session timeout.
            EulaInfo eula= new EulaInfo();
            eula. I_have_read_and_I_accept_this_EULA=true;
            UserToken token = lm_instance.login("admin", "cisco",
                "localhost", 1099, 0, eula);

            // Create a devices using IP address.
            String[] dev_ips = {"192.168.98.120"};
            DeviceStatus dev_status =
```

```

        lm_instance.createDevicesByIPAddr(token, dev_ips);
        Device device = dev_status.getDeviceStatusItems()[0].getDevice();

        // Create a PAK object.
        String[] pak_ids = {"1XSAU930025"};
        PAKStatus pak_status = lm_instance.createPAKs(token, pak_ids);

        // Download PAK information from Cisco.
        MyIDStatusListener my_listener = new MyIDStatusListener();
        String req_id = lm_instance.asyncDownloadPAKInfo(token, pak_ids,
            my_listener);

        // Wait a while for PAK info download to complete....
        .
        .
        .

        // Read the PAK objects.
        pak_status = lm_instance.readPAKs(token, pak_ids);
        PAK pak = pak_status.getPAKStatusItems()[0].getPAK();

        // Prepare License request using the SKU list in the PAK object.
        LicenseRequest[] lic_req = new LicenseRequest[1];
        lic_req[0] = new LicenseRequest();
        lic_req[0].setSKUSelection(pak.getSkuList());
        lic_req[0].setDeviceID(device.getDeviceID());

        // Obtain license.
        req_id = lm_instance.asyncObtainLicense(token, lic_req, my_listener);

        // Assume we have 2 license obtained and stored in
        // the data storage, we can deploy them to the device.
        String[] lic_ids = {"lic00001", "lic0002"};
        req_id = lm_instance.asyncDeployLicenses(token, lic_ids, my_listener);

    } catch (RemoteException ex) {
        ex.printStackTrace();
    }
}
}

```



APPENDIX **B**

Data Object Definitions

The following are definitions of data objects that are used by the LicenseManager class. Getter methods are used for getting the value of class member data. Setter methods are used for setting new values for class member data. This section provides the getter and setter methods for each class and hides the private member data.

```
// Data structure for UserToken information.
public class UserToken implement Serializable {
    private String token;// token generated by the server during login.
    private String username; //username in the server account.
    private int m_timeout = 0; //timeout for the token, 0 represents permanent
    private boolean m_ssl_enabled = false;
}

// Data structure for Group information.
public class Group implement Serializable {
    private String m_group;//group name
    private HashSet<String> m_group_access_list; //group access list
    private HashSet<String> m_devices; //devices belong to this group
}

// Data structure for PAK information.
public class PAK implement Serializable {
private String m_internal_id; //internal encrypted PAK id, use as db key
    private String m_pak_name; //PAK name
    private SKU[] m_sku_list; // list of SKU
    private Date m_last_download; //last time download occurred
    private int m_download_status;//status for the download
    private String m_download_by;//username who perform the download
    private String m_pak_type; //PAK type

    private String m_owner; //username who created this PAK
    private String m_display_name; //PAK name to be displayed
    private HashSet<String> m_pak_access_list; //access list for the PAK
}

// Data structure for SKU information.
public class SKU implement Serializable {
    private String m_type; //SKU type, such as counted, uncounted
    private String m_name; //name of the SKU
    private String m_desc; //description for this SKU
    private int m_ordered_qty; //quantity of the SKU ordered
    private int m_available_qty; //quantify of the SKU left
    private String[] m_features; // feature name of the SKU
    private String m_entitlement; //PAK internal id
    private String[] m_licenseid_list; //list of licenseline id
    private String m_count;//count available in the ordered SKU
}
```

```

}

// Data structure for the obtained license.
public class License implement Serializable {
    public enum ObtainSource { CISCO, PREINSTALL, UNKNOWN };
    public enum DeployStatus { DEPLOYED, UNDEPLOYED, PARTIALLY_DEPLOYED};

    private String m_license_id; //license id as db key, this is generated by CLM server
    private String[] m_feature_names; //feature names contain in the
    private String[] m_feature_versions; //feature version
    private String m_file_name; //filename that contains this License,
not used by CLM
    private String m_device_id; //device UDI this license associated with.
    private Date m_obtain_date; //license obtain date
    private ObtainSource m_obtain_source; //where this license obtained from
    private String m_obtain_by; //username who obtained the license
    private DeployStatus m_deploy_status; //license deploy status
    private String[] m_lic_line_ids; //licenseline ids contain in this license
    private String m_pak_id; // PAK id where this license used to obtained from Cisco.com
}

// Data structure for the obtained licenseline.
public class LicenseLine implement Serializable {
    public enum LLDeployStatus { DEPLOYED, UNDEPLOYED };
    public enum LLType { EVALUATION, EXTENSION, PERMANENT, UNKNOWN};
    public enum LLExpireType { DAY_BASE, DATE_BASE };
    public enum LLState { ACTIVE_IN_USE, ACTIVATE_NOT_IN_USE, INACTIVATE, UNKNOWN }

    private String m_license_line_id; //licenseline ID to identify this licenseline, used
as db key
    private String m_feature_name; //feature name of this licenseline
    private String m_license_xml; //xml string for this licenseline
    private String m_udi; //device UDI this licenseline associated with
    private String m_license_id; //license ID of this licenseline associated with
    private String m_pak_id; //PAK ID of this licenseline associated with
    private LLDeployStatus m_deploy_status = LLDeployStatus.UNDEPLOYED; //Licenseline
deploy status

    private String m_annotation; //Annotation string for this licenseline
    private String m_sku_name; //SKU name this licenseline associated with
    private String m_deployed_by_user; //username who deployed this licenseline
    private Date m_deployed_date; //The date of this licenseline be deployed
    private LLState m_state; //The state of this licenseline read from the device.
    private boolean m_eula_required; //Is EULA required from this licenseline?
    private LLType m_type; //Licenseline type, check LLType enum for options.
    private ExpiryInfo m_expiry_info; //Expiry information if this licenseline is an
expiry license
    private UsageInfo m_usage_info; //Usage left for this licenseline if this licenseline
is an expiry license.

    private long m_index; //Storage index in the device for this licenseline
    private String m_storage; //Storage name of this licenseline is kept
    private CountInfo m_count_info; //CountInfo that keep how many count in the licenseline
//if this is a counted licenseline

    private String m_version; //licenseline feature version
}

Public ExpiryInfo implements Serializable{
    private LLExpireType expire_type; // DAY_BASE, DATE_BASE
    private int expire_days; //number of days to expire for DAY_BASE
    private Date start day; //start date for DATE_BASE
    private Date end_day; //end date for DATE_BASE
}

```

```

public class UsageInfo {
    public long m_totalUsage; //Total usage in milli-seconds read from the device
    public long m_usageLeft; //Usage left in milli-seconds read from the device
}

public class CountInfo {
    public int m_totalCount; //Total count of the licenseline read from the device
    public int m_usageCount; //Count used of the licenseline read from the device
}

// Data structure for RehostInfo
Public class RehostInfo implement Serializable{
    private String permission_ticket; //Permission Ticket returned from Cisco.com
    private String rehost_ticket; //Rehost Ticket after Using the Permission Ticket to
revoke a licenseline
    private SKU[] sku_selection; //SKU selected for Rehost
    private String src_device_id; //source device UDI
    private String dest_device_id; //detination device UDI
}

// Data structure for devices.
public class Device implement Serializable {
    public enum PollStatus { FAILURE, SUCCESS };
    public enum DeviceState { DEV_CONNECT, DEV_CANNOT_PING, DEV_CANNOT_AUTHENTICATE,
DEV_CANNOT_CONNECT }
    public enum TransportMethod {HTTP, TELNET, SSH};
    public enum LicenseOperation {INSTALL, CLEAR, ANNOTATE, REVOKE};
    private String m_pid; //device product ID
    private String m_vid; //device version ID
    private String m_sn; // device serial number
    private String m_device_id; // device ID, same as m_udi
    private String m_ipaddr; // device IP address
    private String m_udi; // device UDI, a string contains m_pid, m_vid and m_sn, used as
key in db
    private String m_hostname; //hostname of the device
    private String m_display_name; //display name for the device, usually same as hostname
or UDI,
if not given by user
    private String[] m_device_access_info; // device access info list, first string is the
TransportMethod,
        //second string is device HTTP url if HTTP/HTTPS is used
        //third string is license agent version.
    private String[] m_licensable_features; //license feature names that are supported by
the IOS image of the device
    private Date m_last_polling_date; //last performed poll date
    private PollStatus m_last_polling_status; //last polled status
    private String m_polled_by; //username who initiated the poll
    private String[] m_member_device_ids; //device UDI list of the child devices,
applicable to master device in
// stackable devices.
    private String m_parent_device_id; //device UDI of the parent device, applicable to
the child devices
//in stackable devices.
    private String m_credential; //device credential, internal generated string to assure
device credibility.
    private DeviceAuthentication m_device_auth; //device login information such as
username/password
    private String m_device_model; //device model

```

```

    private String m_device_type; //device type
    private HashSet<String> m_device_access_list; //device access list to control user
access.
    private DeviceState m_state; //device connectivity state, check values of DeviceState
    private LicenseOperation[] m_device_lic_capabilities;//device capability toward
licenses,
//such as support resend, rehost
    private String m_sw_version; //software version number for this device
    private FeatureInfo[] m_feature_info; //a data structure that organizes licenselines
by feature name
    private String[] m_feature_versions; //feature version list mapping to
m_licensable_feature list.
    private boolean m_rma; //boolean value to indicate if this device is RMA'd
    private boolean m_dirty;//boolean value to indicate if this device is dirty and
requires polling.
    private int m_slot; //slot number of this device,
//applicable to stackable devices.
}

// Data structure for UserInfo.
public class UserInfo implements Serializable {
    public enum Role { ADMIN, INVENTORYMGR, PAKMGR, LICENSEMGR, REPORTMGR}
    private String m_username;
    private String m_password;
    private String m_first_name;
    private String m_last_name;
    private String m_company_name;
    private String m_email_address;
    protected boolean m_email_enabled; //enable email notification
    private String m_cco_username; //Cisco.com username
    private String m_cco_password;//Cisco.com password
    private Role m_role; //Associated Role of this user
    private boolean m_has_agree_eula; //boolean value to indicate this user has agree to
EULA
    private boolean m_do_not_ask_eula_again; //boolean value to indicate the user do not
want to be queried again

        //once the EULA is agreed.
    private Date m_eula_accepted_time; //Date the EULA is accepted by the user
    private boolean m_show_exp_lic; //boolean value to indicate when user
// login, GUI will show expired licenses.
}

// Data structure to be used when requesting for a license.
public class LicenseRequest {
    private SKU[] sku_selection;
    private String device_id;
    private int request_status;
}

// Data structure to be used when requesting license re-host.
public class RehostRequest {
    private SKU[] sku_selection;
    private String src_device_id;
    private String dest_device_id;
}

// Data structure for device username and password.
public class DeviceAuthentication implements Serializable {
    private String username;
    private String password;
    private String enable_password;
}

```

```

Public class UserPwd {
    public String username;
    public String password;
}

public class DiscoveryAuthInfo {
    private UserPwd[] auth_pairs;
    private String[] enable_password;
}

// Data structure for FeatureInfo
Public class FeatureInfo implements Serializable{
    private LinkedList<LicenseLineInfo> ll_info_list; //list of licenselineInfo
    private long total_expiry_usage; //total usage left for this license feature, the
value is in milli-seconds
    private String feature_name ; //feature name
    private String feature_version; //feature version
}

public class LicenseLineInfo implements Serializable {
    public String m_licline_id; //licenseline ID
    public UsageInfo m_usage_info; //exipring usage information for each licenseline
    public LicenseLine.LLType m_type; //licenseline type
    public CountInfo m_count_info; //structure contains count of licenseline if the
licenseline is of counted license.
    public LicenseLine.LLDeployStatus m_deploy_status; //licenseline deploy status
    public LicenseLine.LLState m_state; //licenseline deploy state
}

// Data structure for returning ID and status
public class Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
}

public class IDStatus extends Status implements Serializable {
    private IDStatusItem[] m_item_list;
}

public class IDStatusItem extends Status implements Serializable {
    private String m_id;
}

// Data structure for returning PAK and status
public class Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
}

public class PAKStatus extends Status implements Serializable {
    private PAKStatusItem[] m_item_list;
}

public class PAKStatusItem extends Status implements Serializable {
    private PAK m_pak;
}

// Data structure for returning License and status
public class Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
}

public class LicenseStatus extends Status implements Serializable {
    private LicenseStatusItem[] m_item_list;
}

public class LicenseStatusItem extends Status implements Serializable {
    private License m_license;
}

```

```

}

public class LicLineStatus extends Status implements Serializable {
    private LicLineStatusItem[] m_item_list;
}

public class LicLineStatusItem extends Status implements Serializable {
    private LicenseLine m_licline;
}

public class GroupStatus extends Status implements Serializable {
    private GroupStatusItem[] m_item_list;
}

public class GroupStatusItem extends Status implements Serializable {
    private Group m_group;
}

public class RehostInfoStatus extends Status implements Serializable {
    private RehostInfoStatusItem[] m_item_list;
}

public class RehostInfoStatusItem extends Status implements Serializable {
    private RehostInfo m_group;
}

public class EulaStatus extends Status implements Serializable {
    private boolean accepted_eula;
    private boolean do_not_ask_again;
}

public class ClmJobStatusItem extends Status implements Serializable{
    private ClmJob m_job;
}

public class ClmJobStatus extends Status implements Serializable {
    private ClmJobStatusItem[] m_item_list;
}

// Data structure for returning Device and status
public class Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
}

public class DeviceStatus extends Status implements Serializable {
    private DeviceStatusItem[] m_item_list;
}

public class DeviceStatusItem extends Status implements Serializable {
    private Device m_device;
}

// Data structure for returning ID and status
public class IDStatus extends Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
    private IDStatusItem[] m_item_list;
}

public class IDStatusItem extends Status implements Serializable {
    private int m_error_code;
    private String m_error_message;
    private String m_id;
}

// Data structure for returning asynchronous Progress

```

```

public class ProgressStatus extends Status implements Serializable {
private int m_error_code;
private String m_error_message;
    private String[] current_process_ids;
    private int total_task;
    private int num_processed;
    private int overall_percentage;
    private String[] function_history;
private int num_dev_found;
private Date exec_time;
}

// Interface for asynchronous request status listener.
public interface IDStatusListener {
    public void onStatus(IDStatus status);
}

// Data structure for notifications.
public class Notification {
public enum Operation={LIC_INSTALL, LIC_CLEAR, LIC_REVOKE, LIC_ANNOTATE,
    DEV_CONNECT, DEV_CANNOT_PING, DEV_CANNOT_AUTHENTICATE,
    DEV_CANNOT_CONNECT, DEV_RMA, DEV_RMA_BUT_RUNNING }
    public enum DevModel={TBD};
    public enum DevType={UNKNOWN};
    public enum Severity={SEVERE, MODERATE, LOW, INFO};

private String message; //message of what this notification is about
private Date time_stamp;//Date of the notification sent
private String dev_id; //device ID of the involved device
private String dev_name; //device name of the involved device
private DevType dev_type; //device type
private DevModel dev_model; //device model
private Severity severity; //severity level, using X.733 standard
private string[] group; //group list the device associated with
private Operation op; //enum value to indicate defined operation
private string licline_id; //involved licenseline ID
private string feature_name; //involved feature name
}

// Interface for notification listener.
public interface NotificationListener {
    public void onNotification(Notification notification);
}

public class AuditTrail implements Serializable {
    public enum OperationStatus { SUCCESS, FAILURE }

    private String m_id;
    private String m_user;
    private Date m_start_time;
    private Date m_end_time;
    private String m_operation;
    private OperationStatus m_operation_status;
    private String m_msg;
    private String m_affected; //involved elements, usually a list of device IDs,
licenseline IDs or PAK IDs.
}

// Data structure for EulaInfo statement and agreement
public class EulaInfo {
    private const String eula_agreement_statement="
End User License Agreement
IMPORTANT: PLEASE READ THIS END USER LICENSE AGREEMENT

```

CAREFULLY. DOWNLOADING, INSTALLING OR USING CISCO OR CISCO-SUPPLIED SOFTWARE CONSTITUTES ACCEPTANCE OF THIS AGREEMENT.

CISCO IS WILLING TO LICENSE THE SOFTWARE TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT. BY DOWNLOADING OR INSTALLING THE SOFTWARE, OR USING THE EQUIPMENT THAT CONTAINS THIS SOFTWARE, YOU ARE BINDING YOURSELF AND THE BUSINESS ENTITY THAT YOU REPRESENT (COLLECTIVELY, "CUSTOMER") TO THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, THEN CISCO IS UNWILLING TO LICENSE THE SOFTWARE TO YOU AND (A) DO NOT DOWNLOAD, INSTALL OR USE THE SOFTWARE, AND (B) YOU MAY RETURN THE SOFTWARE FOR A FULL REFUND, OR, IF THE SOFTWARE IS SUPPLIED AS PART OF ANOTHER PRODUCT, YOU MAY RETURN THE ENTIRE PRODUCT FOR A FULL REFUND. YOUR RIGHT TO RETURN AND REFUND EXPIRES 30 DAYS AFTER PURCHASE FROM CISCO OR AN AUTHORIZED CISCO RESELLER, AND APPLIES ONLY IF YOU ARE THE ORIGINAL END USER PURCHASER.

The following terms of this End User License Agreement ("Agreement") govern Customer's access and use of the Software, except to the extent (a) there is a separate signed agreement between Customer and Cisco governing Customer's use of the Software or (b) the Software includes a separate "click-accept" license agreement as part of the installation and/or download process. To the extent of a conflict between the provisions of the foregoing documents, the order of precedence shall be (1) the signed agreement, (2) the click-accept agreement, and (3) this End User License Agreement.

License. Conditioned upon compliance with the terms and conditions of this Agreement, Cisco Systems, Inc. or its subsidiary licensing the Software instead of Cisco Systems, Inc. ("Cisco"), grants to Customer a nonexclusive and nontransferable license to use for Customer's internal business purposes the Software and the Documentation for which Customer has paid the required license fees. "Documentation" means written information (whether contained in user or technical manuals, training materials, specifications or otherwise) specifically pertaining to the Software and made available by Cisco with the Software in any manner (including on CD-Rom, or on-line).

Customer's license to use the Software shall be limited to, and Customer shall not use the Software in excess of, a single hardware chassis or card or such number and types of agent(s), concurrent users, sessions, IP addresses, port(s), seat(s), server(s), site(s), features and feature sets as are set forth in the applicable Purchase Order which has been accepted by Cisco and for which Customer has paid to Cisco the required license fee.

Unless otherwise expressly provided in the Documentation, Customer shall use the Software solely as embedded in, for execution on, or (where the applicable documentation permits installation on non-Cisco equipment) for communication with Cisco equipment owned or leased by Customer and used for Customer's internal business purposes. NOTE: For evaluation or beta copies for which Cisco does not charge a license fee, the above requirement to pay license fees does not apply.

General Limitations. This is a license, not a transfer of title, to the Software and Documentation, and Cisco retains ownership of all copies of the Software and Documentation. Customer acknowledges that the Software and Documentation contain trade secrets of Cisco, its suppliers or licensors, including but not limited to the specific internal design and structure of individual programs and associated interface information. Accordingly, except as otherwise expressly provided under this Agreement, Customer shall have no right, and Customer specifically agrees not to:

(i) transfer, assign or sublicense its license rights to any other person or entity, or use the Software on unauthorized or secondhand Cisco equipment, and Customer acknowledges that any attempted transfer, assignment, sublicense or use shall be void;

(ii) make error corrections to or otherwise modify or adapt the Software or create derivative works based upon the Software, or permit third parties to do the same;

(iii) reverse engineer or decompile, decrypt, disassemble or otherwise reduce the Software to human-readable form, except to the extent otherwise expressly permitted under applicable law notwithstanding this restriction;

(iv) use or permit the Software to be used to perform services for third parties, whether on a service bureau or time sharing basis or otherwise, without the express written authorization of Cisco; or

(v) disclose, provide, or otherwise make available trade secrets contained within the Software and Documentation in any form to any third party without the prior written consent of Cisco. Customer shall implement reasonable security measures to protect such trade secrets.

To the extent required by law, and at Customer's written request, Cisco shall provide Customer with the interface information needed to achieve interoperability between the Software and another independently created program, on payment of Cisco's applicable fee, if any. Customer shall observe strict obligations of confidentiality with respect to such information and shall use such information in compliance with any applicable terms and conditions upon which Cisco makes such information available.

Software, Upgrades and Additional Copies. For purposes of this Agreement, "Software" shall include (and the terms and conditions of this Agreement shall apply to) computer programs, including firmware, as provided to Customer by Cisco or an authorized Cisco reseller, and any upgrades, updates, bug fixes or modified versions thereto (collectively, "Upgrades") or backup copies of the Software licensed or provided to Customer by Cisco or an authorized Cisco reseller. NOTWITHSTANDING ANY OTHER PROVISION OF THIS AGREEMENT: (1) CUSTOMER HAS NO LICENSE OR RIGHT TO USE ANY ADDITIONAL COPIES OR UPGRADES UNLESS CUSTOMER, AT THE TIME OF ACQUIRING SUCH COPY OR UPGRADE, ALREADY HOLDS A VALID LICENSE TO THE ORIGINAL SOFTWARE AND HAS PAID THE APPLICABLE FEE FOR THE UPGRADE OR ADDITIONAL COPIES; (2) USE OF UPGRADES IS LIMITED TO CISCO EQUIPMENT FOR WHICH CUSTOMER IS THE ORIGINAL END USER PURCHASER OR LESSEE OR WHO OTHERWISE HOLDS A VALID LICENSE TO USE THE SOFTWARE WHICH IS BEING UPGRADED; AND (3) THE MAKING AND USE OF ADDITIONAL COPIES IS LIMITED TO NECESSARY BACKUP PURPOSES ONLY.

Proprietary Notices. Customer agrees to maintain and reproduce all copyright and other proprietary notices on all copies, in any form, of the Software in the same form and manner that such copyright and other proprietary notices are included on the Software. Except as expressly authorized in this Agreement, Customer shall not make any copies or duplicates of any Software without the prior written permission of Cisco.

Term and Termination. This Agreement and the license granted herein shall remain effective until terminated. Customer may terminate this Agreement and the license at any time by destroying all copies of Software and any Documentation. Customer's rights under this Agreement will terminate immediately without notice from Cisco if Customer fails to comply with any provision of this Agreement. Upon termination, Customer shall destroy all copies of Software and Documentation in its possession or control. All confidentiality obligations of Customer and all limitations of liability and disclaimers and restrictions of warranty shall survive termination of this Agreement. In addition,

the provisions of the sections titled "U.S. Government End User Purchasers" and "General Terms Applicable to the Limited Warranty Statement and End User License" shall survive termination of this Agreement.

Customer Records. Customer grants to Cisco and its independent accountants the right to examine Customer's books, records and accounts during Customer's normal business hours to verify compliance with this Agreement. In the event such audit discloses non-compliance with this Agreement, Customer shall promptly pay to Cisco the appropriate license fees, plus the reasonable cost of conducting the audit.

Export. Software and Documentation, including technical data, may be subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software and Documentation.

U.S. Government End User Purchasers. The Software and Documentation qualify as "commercial items," as that term is defined at Federal Acquisition Regulation ("FAR") (48 C.F.R.) 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in FAR 12.212. Consistent with FAR 12.212 and DoD FAR Supp. 227.7202-1 through 227.7202-4, and notwithstanding any other FAR or other contractual clause to the contrary in any agreement into which this End User License Agreement may be incorporated, Customer may provide to Government end user or, if this Agreement is direct, Government end user will acquire, the Software and Documentation with only those rights set forth in this End User License Agreement. Use of either the Software or Documentation or both constitutes agreement by the Government that the Software and Documentation are "commercial computer software" and "commercial computer software documentation," and constitutes acceptance of the rights and restrictions herein.

Limited Warranty

Subject to the limitations and conditions set forth herein, Cisco warrants that commencing from the date of shipment to Customer (but in case of resale by an authorized Cisco reseller, commencing not more than ninety (90) days after original shipment by Cisco), and continuing for a period of the longer of (a) ninety (90) days or (b) the warranty period (if any) expressly set forth as applicable specifically to software in the warranty card accompanying the product of which the Software is a part (the "Product") (if any): (a) the media on which the Software is furnished will be free of defects in materials and workmanship under normal use; and (b) the Software substantially conforms to the Documentation. The date of shipment of a Product by Cisco is set forth on the packaging material in which the Product is shipped. Except for the foregoing, the Software is provided AS IS. This limited warranty extends only to the Customer who is the original licensee. Customer's sole and exclusive remedy and the entire liability of Cisco and its suppliers and licensors under this limited warranty will be (i) replacement of defective media and/or (ii) at Cisco's option, repair, replacement, or refund of the purchase price of the Software, in both cases subject to the condition that any error or defect constituting a breach of this limited warranty is reported to Cisco or the party supplying the Software to Customer, if different than Cisco, within the warranty period. Cisco or the party supplying the Software to Customer may, at its option, require return of the Software as a condition to the remedy. In no event does Cisco warrant that the Software is error free or that Customer will be able to operate the Software without problems or interruptions. In addition, due to the continual development of new techniques for intruding upon and attacking networks, Cisco does not warrant that the Software or any equipment, system or network on which the Software is used will be free of vulnerability to intrusion or attack.

Restrictions. This warranty does not apply if the Software, Product or any other equipment upon which the Software is authorized to be used (a) has been altered, except by Cisco or its authorized representative, (b) has not been installed, operated, repaired, or maintained in accordance with instructions supplied by Cisco, (c) has been subjected to abnormal physical or electrical stress, misuse, negligence, or accident; or (d) is licensed, for beta, evaluation,

testing or demonstration purposes for which Cisco does not charge a purchase price or license fee.

DISCLAIMER OF WARRANTY. EXCEPT AS SPECIFIED IN THIS WARRANTY, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OR CONDITION OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, SATISFACTORY QUALITY, NON-INTERFERENCE, ACCURACY OF INFORMATIONAL CONTENT, OR ARISING FROM A COURSE OF DEALING, LAW, USAGE, OR TRADE PRACTICE, ARE HEREBY EXCLUDED TO THE EXTENT ALLOWED BY APPLICABLE LAW AND ARE EXPRESSLY DISCLAIMED BY CISCO, ITS SUPPLIERS AND LICENSORS. TO THE EXTENT AN IMPLIED WARRANTY CANNOT BE EXCLUDED, SUCH WARRANTY IS LIMITED IN DURATION TO THE EXPRESS WARRANTY PERIOD. BECAUSE SOME STATES OR JURISDICTIONS DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, THE ABOVE LIMITATION MAY NOT APPLY. THIS WARRANTY GIVES CUSTOMER SPECIFIC LEGAL RIGHTS, AND CUSTOMER MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION. This disclaimer and exclusion shall apply even if the express warranty set forth above fails of its essential purpose. General Terms Applicable to the Limited Warranty Statement and End User License Agreement

Disclaimer of Liabilities. REGARDLESS WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE OR OTHERWISE, IN NO EVENT WILL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY LOST REVENUE, PROFIT, OR LOST OR DAMAGED DATA, BUSINESS INTERRUPTION, LOSS OF CAPITAL, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL, OR PUNITIVE DAMAGES HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY OR WHETHER ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE OR OTHERWISE AND EVEN IF CISCO OR ITS SUPPLIERS OR LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event shall Cisco's or its suppliers' or licensors' liability to Customer, whether in contract, tort (including negligence), breach of warranty, or otherwise, exceed the price paid by Customer for the Software that gave rise to the claim or if the Software is part of another Product, the price paid for such other Product. BECAUSE SOME STATES OR JURISDICTIONS DO NOT ALLOW LIMITATION OR EXCLUSION OF CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU. Customer agrees that the limitations of liability and disclaimers set forth herein will apply regardless of whether Customer has accepted the Software or any other product or service delivered by Cisco. Customer acknowledges and agrees that Cisco has set its prices and entered into this Agreement in reliance upon the disclaimers of warranty and the limitations of liability set forth herein, that the same reflect an allocation of risk between the parties (including the risk that a contract remedy may fail of its essential purpose and cause consequential loss), and that the same form an essential basis of the bargain between the parties.

The validity, interpretation, and performance of this Warranty and End User License shall be controlled by and construed under the laws of the State of California, United States of America, as if performed wholly within the state and without giving effect to the principles of conflict of laws, and the State and federal courts of California shall have jurisdiction over any claim arising under this Agreement. The parties specifically disclaim the UN Convention on Contracts for the International Sale of Goods. Notwithstanding the foregoing, either party may seek interim injunctive relief in any court of appropriate jurisdiction with respect to any alleged breach of such party's intellectual property or proprietary rights. If any portion hereof is found to be void or unenforceable, the remaining provisions of the Agreement shall remain in full force and effect. Except as expressly provided herein, this Agreement constitutes the entire agreement between the parties with respect to the license of the Software and Documentation and supersedes any conflicting or additional terms contained in any purchase order or elsewhere, all

of which terms are excluded. This Agreement has been written in the English language, and the parties agree that the English version will govern.

Supplemental End User License Agreement for Trial Use of Software with Cisco License Manager Software
 IMPORTANT-READ CAREFULLY: This Supplemental License Agreement for Trial Use of Software with Cisco License Manager Software ("Supplement") contains additional limitations on the license to the Software provided to Customer under the End User License Agreement between Customer and Cisco. Capitalized terms used in this Supplement and not otherwise defined herein shall have the meanings assigned to them in the End User License Agreement. To the extent that there is a conflict among any of these terms and conditions applicable to the Software, the terms and conditions in this Supplement shall take precedence.

By installing, allowing to be installed, downloading, accessing or otherwise using the Software or using the equipment that contains this Software, Customer agrees to be bound by the terms of this Supplement. If Customer does not agree to the terms of this Supplement, Customer may not install, download, access or otherwise use the Software. Customer may retain a third party ("Contractor") to install the Software for Customer, provided that (i) Customer and Contractor agree that the provisioning of installation services by Contractor to Customer creates an agency relationship and (ii) Customer shall remain fully responsible and liable for compliance by Customer and Contractor with the terms of the End User License Agreement and this Supplement. When used below, the term "server" refers to a central processor unit owned or leased by Customer or otherwise embedded in equipment provided by Cisco.

Section 1. Customer hereby agrees that the terms of this Supplement shall govern use of all Cisco Software provided to Customer solely for trial use (e.g., a license of 30 or 60 days) on any devices in your network that use the Software in conjunction with the license of Cisco License Manager, unless otherwise specifically agreed by Customer and Cisco. In the event of a conflict in terms between this Supplement and the terms of the End User License Agreement the terms of this Supplement shall apply.

Section 2. DISCLAIMER OF WARRANTY. ALL SOFTWARE LICENSED UNDER THIS SUPPLEMENT IS PROVIDED "AS IS". ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OR CONDITION OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, SATISFACTORY QUALITY, NON-INTERFERENCE, ACCURACY OF INFORMATIONAL CONTENT, OR ARISING FROM A COURSE OF DEALING, LAW, USAGE, OR TRADE PRACTICE, ARE HEREBY EXCLUDED TO THE EXTENT ALLOWED BY APPLICABLE LAW AND ARE EXPRESSLY DISCLAIMED BY CISCO, ITS SUPPLIERS AND LICENSORS. TO THE EXTENT AN IMPLIED WARRANTY CANNOT BE EXCLUDED, SUCH WARRANTY IS LIMITED IN DURATION TO THE EXPRESS WARRANTY PERIOD. BECAUSE SOME STATES OR JURISDICTIONS DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, THE ABOVE LIMITATION MAY NOT APPLY. THIS WARRANTY GIVES CUSTOMER SPECIFIC LEGAL RIGHTS, AND CUSTOMER MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION. This disclaimer and exclusion shall apply even if the express warranty set forth above fails of its essential purpose.

Section 3. You hereby agree that the trial use of the Software provided for any device in your network under this Supplement is terminable at will by Cisco. The specific license term for any Software licensed under this Supplement and any additional terms covering use of the Software on specific

devices may be set forth separate documentation by Cisco. The Software provided under this Supplement may be shut down or terminated by Cisco after expiration of the term of each specific license. Cisco reserves the right to terminate or shut down any such Software electronically or by any other means available. While alerts or such messages may be provided, it is your sole responsibility to monitor your usage of any Software to ensure that your systems and networks are prepared for the shut down of the Software. You also hereby agree that Cisco will not have any liability whatsoever for any damages, including, but not limited to, direct, indirect, special, or consequential damages related to any Software being shutdown or terminated.

By clicking the "accept" button or typing "yes" Customer indicates that it has read and agrees to be bound by all the terms provided herein including the terms of the End User License Agreement and the Supplement.":

```

    private boolean I_have_read_and_I_accept_this_EULA;
    private boolean do_not_ask_me_again;
    private Date eula_accepted_time;
}

// Data structure for Schedule
public class Schedule implements Serializable{
    public enum ClmTask { POLL_LICENSES, CHECK_EXPIRING_LICENSES};
    public enum Frequency { EVERY_DAY, EVERY_TWO_DAYS, EVERY_THREE_DAYS, EVERY_FOUR_DAYS,
EVERY_FIVE_DAYS, EVERY_SIX_DAYS, WEEKLY, EVERY_EIGHT_DAYS, EVERY_NINE_DAYS,
EVERY_TEN_DAYS, EVERY_ELEVEN_DAYS, EVERY_TWELVE_DAYS, EVERY_THIRTEEN_DAYS, BI_WEEKLY };
    private Calendar m_starting_date;

    private String m_name;//task name
    private Frequency m_frequency; //frequency of the scheduled task
    private int m_notify_if_less_than; //number of days to be notify if the license is
    expiring.
}

public class Policy {
    public enum DeviceAttribute {
        MODEL, GROUP
    };
    public enum SKUAttribute {
        FEATURE_NAME
    };
    String id;                // ID of the policy
    String name;              // name of the policy
    String owner;             // owner of the policy
    String desc;              // description of the policy
    boolean is_public;        // is this policy public?
    DeviceFilter dev_filter;  // the device filer
    SKUFilter sku_filter;     // the SKU filter
    SKUIdentifier sku_in_use; // the SKU in use by the policy
    String executed_by;       // user who executes the policy
    Date executed_at;         // time when executing the policy
}

public class SKUFilter {
    String feature_name;      // specify the interested feature name
}

public class DeviceFilter {
    String type;              // specify the interested type of device (for future use)
    String model;             // specify the interested model of device
}

```

```

        String sw_version;           // specify the interested dev sw version (for future use)
        String group;                // specify the interested device group
        String ip_lower_limit;       // specify the upper limit of IP addr in a range
        String ip_upper_limit;       // specify the lower limit of IP addr in a range
    }

    public class SKUIdentifier {
        String pak;                  // PAK of the selected SKU
        String sku_name;             // name the selected SKU
    }

    //report structure
    enum OutputFormat={ HTML, TEXT };
    enum ReportSubject={LICENSE_DISCREPANCY, REDEPLOYABLE_LICENSE,
        UNUSED_LICENSE, DEVICE_SUMMARY, LICENSE_EXPIRY,
        RMA_DISCREPANCY, AUDITTRAIL};

    //ClmJob structure
    public class ClmJob implements Serializable{
        private String m_job_id; // Job ID to identify a Job in DB, it is the same as request
        ID
        private Date m_start_date; //date when the Job started.
        private Date m_end_date; //date when the Job ends
        private String m_username; //username who initiates the job
        private String m_func_name; //function name of the job is performed
        private boolean m_safe_to_kill=true; //boolean value to indicate if it is safe to
        cancel this task
        private String m_comment; //human readable statement for this job
        private ProgressStatus m_progress; //ProgressStatus for this job
        private boolean m_job_done=false; //boolean value to indicate the job has completed.
        private ModuleLifeCycleIf m_module_handler; //an interface to handle cancel job
        instruction
        private IDStatus m_idstatus; //the IDStatus of the operation after it is completed.
    }

```



APPENDIX **C**

Access Control

This appendix provides information about the following levels of access control in Cisco License Manager, as well as the rules of control:

- [User Control, page C-15](#)
- [Device Control, page C-17](#)
- [PAK Control, page C-17](#)
- [Rules of Control, page C-17](#)

User Control

Cisco License Manager supports the following role-based user management for granular access to resources and features:

- Administrator
- Inventory management
- PAK management
- License management
- Report management

The following table provides a matrix of operations each role can perform. If an API is not listed in this table, it is open to all users.

Operation	Administrator	Inventory Management	PAK Management	License Management	Report Management
createUser	X				
deleteUser	X				
addUserToDeviceAccessList	X				
removeUserFromDeviceAccessList	X				
removeAccessListFromDevice	X				
addUserToGroupAccessList	X				
removeUserFromGroupAccessList	X				
removeAccessListFromGroup	X				

User Control

Operation	Administrator	Inventory Management	PAK Management	License Management	Report Management
addUserToPAKAccessList ¹	X	X	X		
removeUserFromPAKAccessList ¹	X	X	X		
asynDiscoverDevices	X				
asynPollDeviceLicenseInfo ²	X	X			
createDevicesByIPAddr	X				
reCreateDevicesByIPAddr	X				
createDevicesByUDI	X				
checkDeviceConnection ²	X	X	X	X	X
readDevices ²	X	X	X	X	X
writeDevices ²	X	X			
deleteDevices	X				
createDeviceGroup	X	X			
renameDeviceGroup	X	X			
deleteDeviceGroup	X	X			
addDevicesToGroup	X	X			
removeDeviceFromGroup	X	X			
asyncDownloadPAKInfo ¹	X	X	X		
createPAKs	X	X	X		
readPAKs ^{1,3}	X	X	X	X	X
writePAKs ¹	X	X	X		
deletePAKs ¹	X	X	X		
asyncObtainLicense ²	X	X	X	X	
writeLicenses	X	X	X	X	
getLicensesOnDevice ²	X	X	X	X	
asyncDeployLicenses ²	X	X	X	X	
rehostLicenses	X	X	X	X	
resendLicense	X	X	X	X	
reObtainLicense ²	X	X	X	X	
asyncAnnotateLicense	X	X	X	X	
initRehostLicenses	X	X	X	X	
revokeLicenseFroRehost	X	X	X	X	
obtainLicenseForRehost	X	X	X	X	
getRehostInfo	X	X	X	X	
writeLicenseLines	X	X	X	X	
getLicenseLinesOnDevice ²	X	X	X	X	
asyncDeployLicenseLines ²	X	X	X	X	
asyncAnnotateLicensesLine	X	X	X	X	

Operation	Administrator	Inventory Management	PAK Management	License Management	Report Management
listExpiredLicenseLines	X	X	X	X	
createFolder	X	X	X	X	
renameFolder	X	X	X	X	
deleteFolder	X	X	X	X	
addPAKsToFolder	X	X	X	X	
removePAKsFromFolder	X	X	X	X	
generateReport	X	X	X	X	X
readReport	X	X	X	X	X

1. Subject to PAK access control.
2. Subject to device and device group access control.
3. User in PAK Management or higher roles can see PAK ID in plain text. Users in License Management and lower roles can only see the last few letters/digits of PAK ID.

Device Control

An access list is associated with each device and each device group. An access list contains a list of usernames that are allowed to access a particular device or device groups. If no access list exists, the device or device group is open to all users. Only users in an Administrator role can modify the access list.

PAK Control

Each PAK has a owner and also an access list associated with it. PAK owner is the creator of the PAK. Only the PAK owner or users in an Administrator role can modify PAK access list.

Rules of Control

For users in an Administrator role:

- A user in an Administrator role can perform all operations, no matter whether his/her name is in the access list or not.

For users not in an Administrator role:

- Only PAK owner and users in its access list can perform operations on PAK object.
- If both device access list and the access list of its parent group(s) are empty, users in an Inventory Management role can perform operations on this device.
- If a device access list or the access lists of the device group(s) that contain this device are not empty, only users listed in the device or group access list can operate on the device.

