



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
- [getRehostableSKUsByDevice](#), page 7-7
- [getRehostInfo](#), page 7-7
- [initRehostLicense](#), page 7-8
- [listAllLicensesInPAK](#), page 7-9
- [obtainLicenseForRehost](#), page 7-10
- [readLicenses](#), page 7-11
- [rehostLicense](#), page 7-12
- [reObtainLicense](#), page 7-13
- [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.

getRehostableSKUsByDevice

Synopsis

```
RehostableSKUStatus getRehostableSKUsByDevice(UserToken token, String dev_id) throws
RemoteException;
```

Description

This function retrieves an array of SKUs available for license rehost on the specified input device. This function will also request SWIFT to query all SKUs whose licenses have been deployed on 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 64 ASCII characters in the range from x21 to x7A	ID of the specified device.

Return

The function returns a RehostableSKUStatus object. If there are no licenses obtained for the device, the RehostableSKU field in the returned RehostSKUInfoStatus is set to null.

Error and Exception

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

If an error occurs, the information 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 `Status` Object indication error

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 ClmError.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 Status Object indication error.

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.