



## 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
- [listExpiringLicenseLines](#), page 8-7
- [readLicenseLines](#), page 8-8
- [writeLicenseLines](#), page 8-9

## 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.

## listExpiringLicenseLines

**Synopsis**

```

HashSet<ExpiredLicense> listExpiringLicenseLines(UserToken token, int days_to_expire)
throws RemoteException;

```

**Description**

This function returns a list of licenseline that will expire in `days_to_expire` days.

**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.
days_to_expire	int, mandatory	0 - 65535	Specify days to the expiration of licenses.

**Return**

This function returns `HashSet<ExpiredLicense>` objects.

**Error and Exception**

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

# 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.