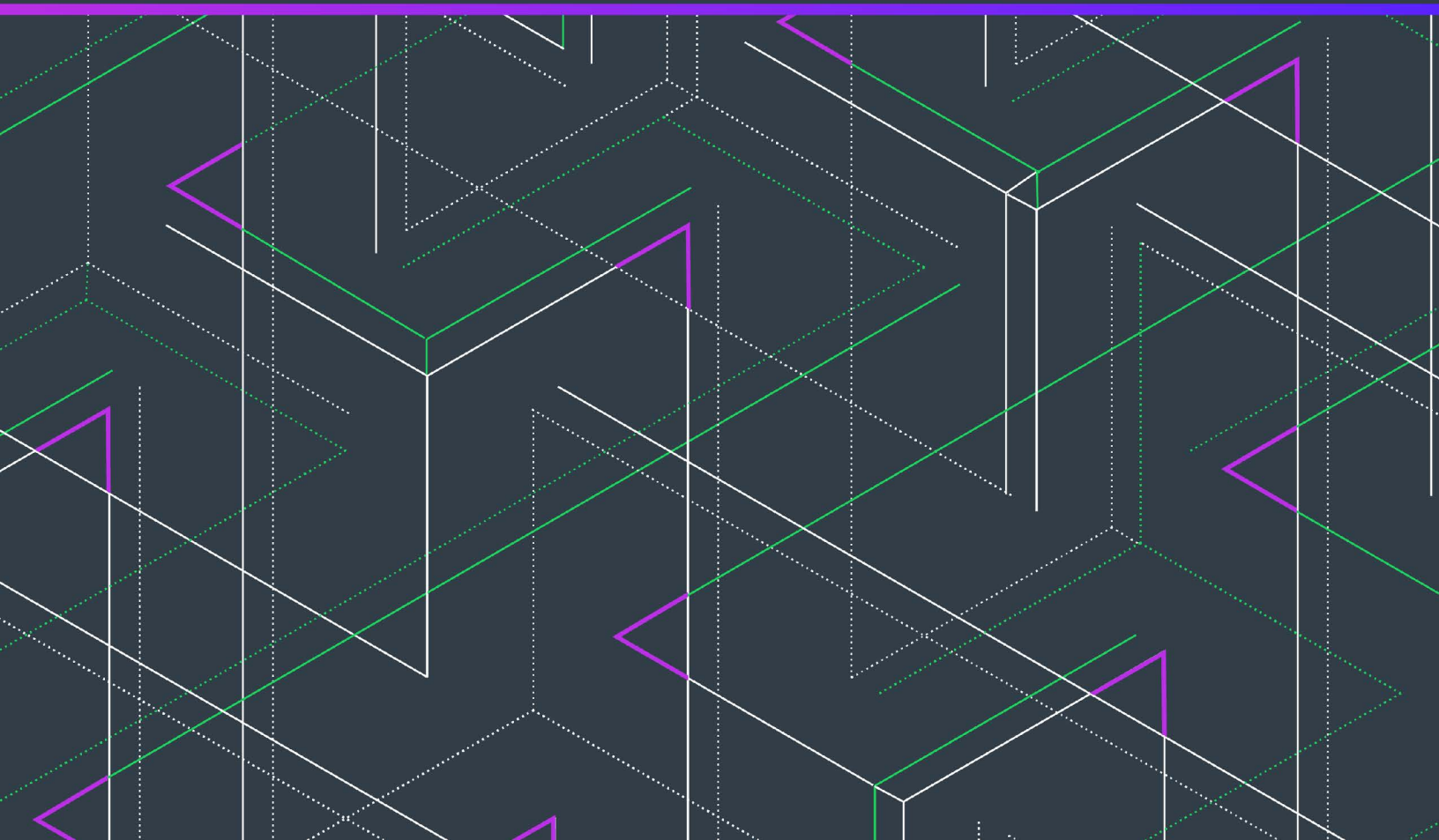
revenera

FlexNet Embedded 2026.09

License Server Administration Guide



Legal Information

Book Name: FlexNet Embedded 2026.09 License Server Administration Guide

Part Number: FNE-2026.09-LAG00

Product Release Date: September 2026

Copyright Notice

Intellectual Property

Restricted Rights Legend

Contents

FlexNet Embedded Local License Server Quick Start Reference

This “quick start” reference is geared towards those license server administrators who are familiar with the FlexNet Embedded local license server and need only a reference to commands and basic steps to get the server up and running.

The instructions in this “quick start” reference refer to the following components:

- The executable for the FlexNet Embedded local license server, `flexnetls`, to manage the license server service
- The FlexNet License Server Administrator command-line tool to configure and administer the server

Additionally, these instructions provide no background information about the task at hand and no details about the specific commands being used. If you need more information, refer to appropriate chapters in this book.

This “quick start” reference contains the following topics:

- Installing and Running the License Server
- Performing Other Service Maintenance Tasks
- License Server Administrator Command-line Tool: Command Summary

About “flexnetlsadmin” Commands

Refer to the following for additional assistance about entering the `flexnetlsadmin` commands listed in this “quick start” reference:

- `flexnetlsadmin` commands require the license server’s base URL (*licenseServer_baseURL*). For details about the base URL, see Base URL for the License Server in the Getting Started chapter.
- If administrative security is enabled on the license server, you might need to provide authorization credentials to perform certain `flexnetlsadmin` operations listed in next sections. For more information about providing these credentials (and about configuring and managing security), see Using the Command-line Tool to Manage Administrative Security in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- For a summary `flexnetlsadmin` command usage, see License Server Administrator Command-line Tool: Command Summary.

- For detailed descriptions of the `flexnetlsadmin` commands, see the chapter Using the FlexNet License Server Administrator Command-line Tool.

Installing and Running the License Server

This section covers the "quick start" steps used to install and start the FlexNet Embedded local license server as a service on Windows or Linux.

On a Linux platform, you install the license server as a systemd service, as described here.

For help in using `flexnetlsadmin` commands, see About "flexnetlsadmin" Commands.

Table -1 ▪ Installing and Running the FlexNet Embedded Local License Server

Task		Do this...	
		Windows	**Linux**
1	**Configure the installation and service**	Edit `flexnetls.settings` if needed to set the port number, your Java installation (JDK or JRE) path, hostid, log threshold, and other options.	Options added to the installation step (Step 2) enables you to modify configuration.
2	**Install the service**	Run `flexnetls.bat -install` ***Note ▪*** *You will be asked whether you want to install the MetaDaemon service, which is required for resilient hostid handling on physical machines (if enabled by the producer).*	Run sudo `sudo ./install-systemd.sh` to install and start with default configuration. Add options to modify configuration (see Configuration Values Editable from the Command Line).
3	**Start the service**	Run `flexnetls.bat -start` ***Note ▪*** *Depending on the security policies defined on your device, a warning message might be displayed, warning you to only run scripts that you trust. If such as warning is displayed, confirm that you want to allow each script to be executed.*	Installation starts the service, or run `sudo systemctl start flexnetls-producer_Name`
4	**Do a "health check" on the license server**	Run `flexnetlsadmin.bat -server licenseServer_baseURL -status`	Run `flexnetlsadmin -server licenseServer_baseURL -status`
5	**Activate license rights**	Run `flexnetlsadmin.bat -server licenseServer_baseURL -activate -id activation_ID -count n`	Run `flexnetlsadmin -server licenseServer_baseURL -activate -id activation_ID -count n`

Table -1 ▪ Installing and Running the FlexNet Embedded Local License Server (cont.)

Task		Do this...	
6	**Verify license rights**	Run `flexnetlsadmin.bat -server licenseServer_baseURL -features`	Run `flexnetlsadmin -server licenseServer_baseURL -features`
7	**Provide license server URL to clients**	Distribute the license server URL for handling capability requests (for example, `http:/licenseServer_baseURL/request`).	
8	**Administer/configure the license server**	See License Server Administrator Command-line Tool: Command Summary for the FlexNet License Server Administrator commands used to administer and configure the license server. If administrative security is enabled, certain operations require authorization credentials in order to perform them (see About "flexnetlsadmin" Commands).	

Performing Other Service Maintenance Tasks

The following provides a quick reference to other tasks used to maintain the license server as a service.

The tasks listed for the license server running on a Linux platform assume the service was installed using systemd.

For help in using `flexnetlsadmin` commands, see About "flexnetlsadmin" Commands.

Table -2 ▪ Additional Service Maintenance Tasks

Task	Do this...	
	Windows	**Linux**

Table -2 ▪ Additional Service Maintenance Tasks (cont.)

Task	Do this...	
Update service with changed options	1. Edit `flexnetls.settings` as needed 2. Run `flexnetls.bat -update` 3. Run `flexnetls.bat -start`  ***Note ▪*** *Depending on the security policies defined on your device, a warning message might be displayed, warning you to only run scripts that you trust. If such as warning is displayed, confirm that you want to allow each script to be executed.*	Update service settings: 1. Run `sudo systemctl stop flexnetls-producer_name` 2. Run `sudo gedit /etc/systemd/system/flexnetls-producer_name.service.d/flexnetls.conf` to make edits (or replace `gedit` with editor of choice) 3. Run `sudo systemctl daemon-reload` 4. Run `sudo systemctl start flexnetls-producer_name` Update local settings: 1. Run `sudo gedit /opt/flexnetls/demo/local-configuration.yaml` to make edits (or replace `gedit` with editor of choice) 2. Run `sudo systemctl restart flexnetls-producer_name`
Suspend service (server temporarily stops accepting capability requests)	Run `flexnetlsadmin.bat -server licenseServer_baseURL -status -suspend`	Run `flexnetlsadmin -server licenseServer_baseURL -status -suspend`
Resume service from suspension	Run `flexnetlsadmin.bat -server licenseServer_baseURL -status -resume`	Run `flexnetlsadmin -server licenseServer_baseURL -status -resume`
Stop service	Run `flexnetls.bat -stop`	Run `sudo systemctl stop flexnetls-producer_name`
Uninstall the service	1. Run `flexnetls.bat -stop` 2. Run `flexnetls.bat -uninstall` 3. Execute `sc delete FNLS-producer_name` (to clean up service components). 4. Delete server's installation files	1. Run `systemctl stop flexnetls-producer` 2. Run `systemctl disable flexnetls-producer` 3. Run `sudo rm /etc/systemd/system/flexnetls-producer.service` 4. Run `sudo rm -r /etc/systemd/system/flexnetls-producer.service.d`

License Server Administrator Command-line Tool: Command Summary

The following describes the command options and associated arguments used by the FlexNet License Server Administrator command-line tool:

- Usage
- Additional Information
- Command Summary

Usage

The following shows general command usage for the license server administrator and an enterprise user.

For a License Server Administrator

The following is command usage for a license server administrator performing general administrative tasks:

```
flexnetlsadmin -server licenseServer_baseURL [-authorize adminName {adminPassword|
    -passwordConsoleInput}] -command_option [command_option_arguments] [-command_option...]
```

The following is the command usage for a license server administrator managing user accounts when administrative security is enabled on the license server:

```
flexnetlsadmin -server licenseServer_baseURL -authorize adminName {adminPassword|
    -passwordConsoleInput} -users [-create|edit|delete]
```

For an Enterprise User

The command usage for an enterprise user is the following:

```
flexnetlsadmin -server licenseServer_baseURL [-authorize userName {userPassword|
    -passwordConsoleInput}] -command_option [command_option_arguments] [-command_option...]
```

Additional Information

Refer to the previous section About “flexnetlsadmin” Commands for additional assistance in providing the *licenseServer_baseURL* or required authorization credentials to run the `flexnetlsadmin` commands.

Command Summary

The follow table summarizes `flexnetlsadmin` command options and arguments.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options

Option	Arguments	Description
-help	(no argument)	Lists the syntax of all `flexnetlsadmin` commands.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-server	(no argument)	Sets the license server base URL required in every command as the first option. For a local license server, use the following URL: `flexnetlsadmin.bat -server http://` *`licenseServerURL`*`/api/1.0/instances/`*`instanceID`* For a CLS instance, use the URL that the producer provides, similar to this: `https://`*`siteID`*`.compliance.flexnetoperations.com/api/1.0/instances/`*`instanceId`* Alternatively, set this value as the environment user variable `FLEXNETLS_BASEURL` on your machine so that you no longer have to include this option in the commands. See License Server URL Designation for more information.  ***Note ▪*** *Use of the HTTPS protocol on the local license is strongly recommended when administrative security is enabled on the license server.*

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *adminName* {*adminPassword* | -passwordConsoleInput}	The `-authorize` command option and its related options and arguments are in effect only when administrative security is enabled on the license server. Position the `-authorize` option after the `-server` option and before any other options. Administrators can choose to enter their password directly in the command or, for security reasons, use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. The remaining options and arguments for `-authorize` are described next. See Using the Command-line Tool to Manage Administrative Security for details. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*	
	`[-users]`	Shows all current user accounts on the license server.
	`[-users -create` *`userName`* `{`*`userPassword`*`| -passwordConsoleInput} [`*`roles`*`]]`	Creates a new user account: • The *`userName`* and *`userPassword`* must meet the criteria specified in Credential Requirements. • To avoid exposing the password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. • Roles are optional. If no role is specified, ROLE_READ is assigned by default. Valid roles include ROLE_READ, ROLE_RESERVATIONS, ROLE_ADMIN, ROLE_OFFLINE, ROLE_DROP_CLIENT, and ROLE_PRODUCER. Multiple roles can be combined using a plus sign (+) without any spaces in between (for example, `ROLE_READ+ROLE_RESERVATIONS`). See User Roles Defining Administrative Privileges for details. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *adminName* {*adminPassword*| -passwordConsoleInput} (cont.)	[-users -edit *userName* {*password*|*newPassword*| -passwordConsoleInput} [*newRoles*]]	Updates the password or role or both for the user account identified by *userName*. ● The password is not optional. Enter either the user's current password or, if changing the password, the new password (which must meet the criteria listed in Credential Requirements). ● To avoid exposing the password as a command-line argument, you can use the -passwordConsoleInput option, which then issues a prompt to enter the password as console input. ● Assignment of new roles is optional. If one or more new roles are specified, they replace all current roles. If no roles are specified, the user's current role assignment remains in effect. For information about roles, see User Roles Defining Administrative Privileges.  ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*
	[-users -delete *userName*]	Deletes the user account specified by *userName*
	[*any options listed below*]	Authorizes your access, as needed, to operations specified by any of the options listed below.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *userName* {*userPassword*| -passwordConsoleInput}	[*specific options listed below*]	Authorizes an enterprise user's access to the specific operations to which that user has privileges. Users can choose to enter their password (as defined for their account) directly in the command or, for security reasons, use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*
-status	[-suspend | -resume]	Shows information about the license server, including its status (as active, inactive, or suspended), its build and version, the back-office URL, and other information. The `-suspend` option temporarily suspends the license server (that is, the license server is still running but does not accept capability requests from client devices). The `-resume` option ends the license server's suspended state so that the server can proceed with normal operations.
-hostid	(no argument)	Displays the license server's hostid value used to fulfill capability requests against a back-office server. If the server has multiple hostid values, the list contains the available hardware Ethernet addresses and dongle IDs. If virtual hosts are supported, the VM UUID will also be listed.
	-selected	Returns the current "selected" hostid—that is, the hostid currently used by the license server.
	-setactive *hostIdValue hostIdType*	Designates a new "selected" hostid. The new hostid must be one of the hostids returned by the `-hostid` command.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-activate	`-id` *`activation-id`* `[-count` *`value`*`]`	Activates licenses on the server by installing the specified number of copies of license rights (identified by the activation ID) from the back office. You can specify the `-activate` option multiple times to activate license rights from different activation IDs.
	`-activate -id` *`activation_id`* `-count` *`value`* `-O` *`filename`*	(Part 1 of an offline activation) Generates a capability request containing the activation ID and saves the request as a binary file. To generate the request with multiple activation IDs, repeat the `-activate` option for each ID.
	`-load` *`filename`*	(Part 2 of an offline activation) Processes the offline capability-response binary file from the back office to install licenses on the license server.
-licenses	`[-verbose]`	Retrieves summary information or details (with `-verbose`) about current license distribution to client devices.
-features	(no argument)	Shows details about the features in the current license pool on the server.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-config	`[-filter none | license-policy | sync | security | log | general | capability | failover]`	Lists all license-server policy settings. Use the `-filter` argument to list settings for only specific categories. The categories include: • `none`: All categories • `license-policy`: Policies for license distribution and usage • `sync`: Policies for synchronization with back office • `security`: The policy enabling administrative security on the license server • `log`: Logging parameters • `general`: Attributes identifying the license server • `capability`: Policies for polling the back-office for license updates • `failover`: License server failover You can specify multiple categories for the `-filter` option. For example, `flexnatlsadmin -config -filter sync log` shows synchronization and logging settings. (Use `-config` or `-config -filter none` to list all settings.) Those settings that you can edit are listed with an asterisk.
	`-set settingName=settingValue, settingName=settingValue...`	Overwrites the current value for the specified policy setting. You can provide multiple value pairs separated by commas. For additional syntax formats, see Override Policy Settings.  ***Note ▪*** *To determine which settings are editable, use "-config" to list the settings; editable settings are marked with an asterisk. Additionally, when you use "-o" to write settings to a file, only editable settings are written. Another way to apply edits to multiple settings is to use "-filter category -o <filename>" to write the editable settings to file, edit the settings as needed, and then use "-load <filename>" to apply the edits.*
	`-reset settingName,settingName...`	Resets one or more policy settings back to their original default values set by the producer. Separate multiple setting names by commas. For additional syntax formats, see Override Policy Settings.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-config (cont.)	`-o` *`filename`*	Writes the current, editable policy settings (except those is the `general` category) to the specified file in JSON format. This argument can be used with the `-filter none` option to write all editable settings to file or with the `-filter` *`category`* option (except for the `general` category) to limit the scope of editable settings written to the file.
	`-load` *`filename`*	Applies the edits to policy settings from the specified file to the existing policy settings. The contents of the file you are loading must be in JSON format. In a typical scenario, first run `-filter` *`category`* `-o` *`filename`* to write current editable settings to a file, edit the values as needed, and then apply the edits using the `-load` *`filename`* option.
-model	(no argument)	Displays the model definition that is currently active, showing the named license pools and rules.
	`-load` *`filename`*	Uploads the model definition to the license server. The contents of the file you are loading must be written according to the grammar in Model Definition Grammar and Syntax—EBNF. Only one model definition can be active for each license server instance. You cannot upload a model definition with the name `reservations` or `default`, because these names are reserved.  ***Important ▪*** *If you were previously using reservations, you must delete all reservation groups. Otherwise, uploading a model definition will fail.*
	`-delete`	Deletes the model definition that is currently applied to the license server. The default model definition is automatically applied to the license server.
-partitions	(no argument)	Displays all license pools (named and default) for the license server. The output lists information about every feature and every feature slice in each license pool (feature slices are portions of feature counts allocated to a license pool).

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-reservations	[-group *group_id*] [-o *filename*]	Provides a summary of the reservation groups, including each group's ID, currently available to the license server. The -group *group_id* option shows details—at the reservation (hostid) and reservation-entry (feature) levels—for the specified reservation group. The -o *filename* option writes the reservation information in JSON format (for all groups or for the specified group ID) to a file.
	-genjson *filename*	Generates a template file for reservations in JSON format.
	-load *filename*	Creates a reservation group and its reservation entries via a file containing the reservation definitions in JSON format.  ***Note ▪*** *To make changes to a reservation group, delete the group and recreate it with the changes to the reservation definitions.*  ***Important ▪*** *If you were previously using named license pools, you must delete the active model definition before creating reservations.*
	-delete -group *group_id*	Deletes the entire reservation group with all its reservations and reservation entries.
	-delete -group *group_id* -reservation *reservation_id*	Deletes the specified reservation within the specified reservation group.
-uploadPublicKey *clientPublicKeyFileName*	(no argument)	Uploads a file containing a client-side RSA public key (2048-bit DER-encoded) in octet-stream format to the license server. The producer will provide details about obtaining and uploading this key should such a key be required. License-server administrator credentials are required to use this option.
-binding	(no argument)	Shows both the current policy for a binding-break detection facility (enabled by the producer) and the current binding status as detected by this facility (ok, hard break, or soft break). If this facility is not enabled, the output indicates as such.

Table -3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-version	(no argument)	Shows the version and build information for the FlexNet License Server Administrator command-line tool.
-json	(no argument)	Displays output in JSON format instead of the default table layout.  ***Note ▪*** *You can query the JSON output of `flexnetlsadmin` with an external tool like jq. For information about jq, see https://stedolan.github.io/jq/manual/v1.6/.*

1

Introduction

The FlexNet Embedded license server provides functionality for serving and monitoring a counted pool of licenses for intelligent devices or software clients using products enabled with FlexNet Embedded licensing. The license server is designed to administer and enforce a pool of licenses within your enterprise, report license usage to the back office, and provide served-license status information in conjunction with client code that incorporates FlexNet Embedded.

This guide describes how to administer the FlexNet Embedded license server.

Local License Servers vs. License Servers Hosted in the Cloud

The FlexNet Embedded license server can be deployed as a local license server at your site or as a Cloud Licensing Service (CLS) instance, hosted in the Revenera Cloud. Both the FlexNet Embedded local license server and the CLS instance provide the same functionality.

Note that the instructions or descriptions in this book focus on the local license server, basically because the setup and deployment of the CLS instance and a good deal of its administration are performed in FlexNet Operations. However, this book directs you to the appropriate FlexNet Operations documentation to help you get started with the CLS instance. Additionally, the book includes the Managing a CLS Instance chapter to provide basic procedures for managing the CLS instance.

Finally, because both license servers share the same functional endpoints, license server administrator tools provided by the producer (for example, `flexnetlsadmin`) can be used to manage important features, such as license reservations, administrative security, and enterprise user accounts, for both license server types. This book notes these areas where functionality and procedures apply to both server types.

What's in this Guide

The *FlexNet Embedded License Server Administration Guide* includes the following chapters (in addition to the FlexNet Embedded Local License Server Quick Start Reference at the beginning of this book):

Table 1-1 ▪ Contents of the FlexNet Embedded License Server Administration Guide

Topic	Content
Introduction	Provides an overview of the book and list of conventions used in the book's contents.
FlexNet Embedded Local License Server Quick Start Reference	A reference to commands and basic steps to get the server up and running (geared towards license server administrators who are familiar with the FlexNet Embedded local license server).
Getting Started	Includes the requirements and procedures to help you configure, install, and start the FlexNet Embedded local license server and implement administrative security for both the local license server and a CLS instance.
Using the FlexNet License Server Administrator Command-line Tool	Describes how to use the FlexNet License Server Administrator command-line tool to manage the FlexNet Embedded local license server or a CLS instance and its license distribution. (The producer decides whether to provide this tool.)
More About License Server Functionality	Provides background and set-up information for types of functionality that might be enabled for your FlexNet Embedded local license server or CLS instance.
Managing a CLS Instance	Highlights procedures used to manage a version of the license server hosted in either the Revenera or producer's Cloud.
Reference: License Server Policy Settings	Provides a reference to the license server policy settings used to control licenser-server operations. Some settings are editable (and are marked as such in the reference).
Model Definition Grammar for Named License Pools	Describes the possible grammar structures of the language used to write a model definition for defining named license pools, and provides use case examples for sharing feature counts between license pools.
Logging Functionality on the Local License Server	Describes the logging functionality available for the local license server, including logging style, custom log configurations, and how to integrate license server logging with external systems.
Reference: Special Characters Allowed in User Password	Lists the special characters that are allowed in the user password for an enterprise user account.
Reference: Local License Server H2 Database Migration 1.4.x to 2.x	Provides instructions and best practices for migrating the local license server H2 database from 1.4.x to 2.x.

Product Support Resources

The following resources are available to assist you with using this product:

- Revenera Product Documentation
- Revenera Community
- Revenera Learning Center
- Revenera Support

Revenera Product Documentation

You can find documentation for all Revenera products on the Revenera Product Documentation site:

https://docs.revenera.com

Revenera Community

On the Revenera Community site, you can quickly find answers to your questions by searching content from other customers, product experts, and thought leaders. You can also post questions on discussion forums for experts to answer. For each of Revenera's product solutions, you can access forums, blog posts, and knowledge base articles.

https://community.revenera.com

Revenera Learning Center

The Revenera Learning Center offers free, self-guided, online videos to help you quickly get the most out of your Revenera products. You can find a complete list of these training videos in the Learning Center.

https://learning.revenera.com

Revenera Support

For customers who have purchased a maintenance contract for their product(s), you can submit a support case or check the status of an existing case by first logging into the Revenera Community, clicking **Support** on the navigation menu to open the **Support Hub** page, and then clicking the **Open New Case** or **Case Portal** button.

Contact Us

Revenera is headquartered in Itasca, Illinois, and has offices worldwide. To contact us or to learn more about our products, visit our website at:

http://www.revenera.com

You can also follow us on social media:

- X
- Facebook
- LinkedIn

- YouTube
- Instagram

2

Getting Started

This chapter focuses on how to configure, install, and start the FlexNet Embedded local license server at your site but also includes important set-up instructions that apply to both the local license server and a Cloud Licensing Service (CLS) instance. The following sections are included:

About Getting Started with a CLS Instance

Depending on the agreement with the producer, either you or the producer creates the Cloud Licensing Service (CLS) instance through FlexNet Operations. For more information about managing the CLS instance, refer to the help system for the FlexNet Operations End-User Portal; or use the Managing a CLS Instance chapter in this book as a quick reference for performing basic administration tasks.

Requirements for the Local License Server

The following sections list the requirements for the FlexNet Embedded local license server:

- Hardware Requirements
- Validated Platforms
- Virtual Machine Support
- Java Prerequisites

Important ▪ *Due to potential incompatibilities with trusted storage and other stored data, do not downgrade your current license server to a previous version.*

Hardware Requirements

The following are the minimum hardware requirements for the license server:

- **Disk**—500 MB
- **RAM**—4 GB
- **CPU**—2 GHz - 2 Cores

Validated Platforms

The following table lists the validated platforms for the FlexNet Embedded license server.

Table 2-1 ▪ Validated FlexNet Embedded License Server Platforms

Operating System	CPU Architecture	Operating System Versions
***Linux**	x86-64	• Ubuntu 24.04.1 LTS • Red Hat Enterprise Linux 10 • Red Hat Enterprise Linux 9 • RockyLinux 10 • RockyLinux 9 • AlmaLinux 9
	ARM64	• AlmaLinux 9 • RockyLinux 9 • Red Hat Enterprise Linux 9 • Ubuntu 24.04.1

Table 2-1 ▪ Validated FlexNet Embedded License Server Platforms

Operating System	CPU Architecture	Operating System Versions
****Windows**	x86-64	• Windows 10 and 11 • Windows Server 2016 • Windows Server 2019 • Windows Server 2022 • Windows Server 2025

* The FlexNet Embedded license server has not been validated on RedHat Enterprise Linux 7 and 8, but these platforms are still supported.

** .NET Framework 4.5 or later is required for Windows platforms.

For a list of the lifecycle timelines for the FlexNet Embedded local license server, see the FlexNet Embedded End-Of-Life Notice page.

Virtual Machine Support

Virtualization functionality in the FlexNet Embedded local license server supports the following:

- VMware ESXi 7.0.3
- VMware Workstation 17.0.0
- Microsoft Hyper-V 10.0.2 on Windows Server 2016 and Windows Server 2022
- Citrix XenServer 8.2
- Oracle VirtualBox 7.0.10
- QEMU-KVM 9.0.0-10
- Parallels 15.1.2 (not validated)
- Google Compute Cloud
- Amazon EC2

Note ▪ *For ARM platforms, virtual machine support has been validated on AWS and VMware Fusion.*

Java Prerequisites

The following are the Java prerequisites for the machine on which the FlexNet Embedded local license server is installed:

- Oracle Java 21 or OpenJDK 21
- A 64-bit JRE (not adhering to this requirement can cause the license server to fail to start)

- Windows only: The `JAVA_HOME` (or `JRE_HOME`) environment variable on your system set to the path for your default JDK (or JRE) installation.

Note ▪ *The license server requires only the JRE component. If JRE is your default Java installation, set the JRE_HOME environment variable; if JDK is your default installation, set JAVA_HOME.*

Overview: Administrator Experience on the Local License Server

The following table summarizes the basic tasks you perform as a license server administrator on the local license server. Each task description includes a link to further instructions in this book.

Table 2-2 ▪ Overview of the License Server Administrator Experience

Phase	Task	Description
1	**Configure, install, start the license server**	This phase involves installing and starting up the FlexNet Embedded local license server as a service. Before installing the license server, you have the option to configure settings that define the local environment in which the license server will run. (These settings can be edited any time after installation as well.) See Configuring, Installing, and Starting the Local License Server in this chapter.
2	**Edit local settings as needed**	The settings defining the license server's local environment at installation can be edited any time after installation as needed. See Editing the Local Settings Post-Installation in this chapter. Also see the More About License Server Functionality chapter for instructions for configuration you might need to perform to enable certain functionality on the server.
3	**Prepare to use the License Server Manager**	The software producer might have provided the FlexNet License Server Manager as a tool with which to administer your license server. The License Server Manager is available in two formats: ● Docker container (Windows and Linux; requires Docker to be installed) ● Executable (.exe) (Windows only) The License Server Manager is described in a separate manual, the FlexNet License Server Manager Guide. This manual is available on the Revenera Product Documentation site (https://docs.revenera.com).

Table 2-2 ▪ Overview of the License Server Administrator Experience (cont.)

Phase	Task	Description
4	**Manage administration security**	If administration security on the license server is enabled, you need to reset your password and optionally create other enterprise user accounts. See Managing Administrative Security on a Local License Server or CLS Instance for more information.
5	**Activate licenses on the license server**	A purchased set of product licenses needs to be activated on the license server before it can distribute the licenses to client devices running the licensed products. Any license rights pre-mapped to the license server in FlexNet Operations are automatically activated when the server starts up. However, in some cases, especially when the installed license server is "unknown" to FlexNet Operations, you need to manually activate licenses. To manually activate licenses on the license server, use the instructions provided by the software producer; or do one of the following: ● If the FlexNet License Server Administrator command-line tool is installed with the license server, refer to Activating License Rights on the Server in the chapter Using the FlexNet License Server Administrator Command-line Tool. ● If the FlexNet License Server Manager is installed with the server, refer to "Offline Server Updates View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).
6	**Administer the license server**	The administrator tool that the producer provides with the license server enables you to manage and monitor the server and its operations. For example, you can activate new licenses on the server, monitor license distribution among client devices, manage license reservations, or view and edit current configuration settings that define the server's environment and policies. If you are using one of the FlexNet Embedded license server administration tools, you can refer to the appropriate chapter for more information: ● Using the FlexNet License Server Administrator Command-line Tool ● FlexNet License Server Manager Guide, available from the Revenera Product Documentation site (https://docs.revenera.com)
--	**Uninstall the license server**	For various reasons, you might need to uninstall the local license server. See Uninstalling the Local License Server in this chapter for instructions.

Configuring, Installing, and Starting the Local License Server

Refer to the following sections for information and instructions needed to install and start (or uninstall) the local license server:

- Configure, Install, and Start on Windows
- Configure, Install, and Start on Linux
- Local License Server Components
- Setting the Server Time Zone
- Logging Functionality

Configure, Install, and Start on Windows

Use this procedure to install and start the FlexNet Embedded local license server as a service on a Windows platform.

Important ▪ *When upgrading your license server, uninstall the old license server service before installing and starting the service for the new license server. For instructions on uninstalling the license server service on Windows, see Uninstall on Windows.*

Your producer may provide one or both of the following installers:

- Command-based installer—Uses command-line interface for installation
- InstallAnywhere-based installer—Uses a graphical wizard interface

Choose the appropriate section below based on the installer type you received:

- Running the Command-Based Installer
- Running the InstallAnywhere-Based Installer

Running the Command-Based Installer

This section explains how to install and start the license server using the command-based installer.

Task ***To install and start the license server as a Windows service***

1. Unpack the files received from the software producer into an installation directory.

   Note ▪ *Make sure that the `producer-settings.xml` file is located in the same directory as `flexnetls.bat`. If the producer included the file `flexnetls-hostmetadata.exe`, ensure that it is placed in the `server` directory.*

2. Open the `flexnetls.settings` file (located in the installation directory) in a text editor, and update it with your local environment information; or leave the file as is to accept the default settings. (For example, you might want to change the `JAVA_HOME` value or uncomment and provide a value for the `PORT` setting.) When you update any setting, these rules apply:

   - Any setting value that uses a space must be enclosed in quotations
   - Insert no spaces before or after the equal sign (=) in the setting syntax (for example, `PORT=7071`).

   Refer to the following for a description of each setting:

   Table 2-3 ▪ Local Settings for the License Server on Windows

Setting	Description
Required Settings	
FLEXNETJAR	The Java executable file for FlexNet Embedded local license server (default is `flexnetls.jar`).
PUBSETTINGS	The license server configuration file generated by the producer (default is `producer-settings.xml`).
JAVA_HOME (or JRE_HOME)	The path for JDK or JRE installation that the license server should use. The `flexnetls` batch file uses this location to find the necessary `java.exe` and `jvm.dll` files. By default, the license server uses the value of your `JAVA_HOME` (or `JRE_HOME`) system environment variable to determine the Java installation location, as indicated by the `"%JAVA_HOME:"=%"` (or `"%JRE_HOME:"=%"`) value for this local setting. However, if you want the license server to use a different Java installation on your system, edit this local setting to override the server's use of the environment variable. (This override pertains to the license server only; your system continues to use the Java installation defined by the system environment variable.) To perform the override, replace the default value (for example, `"%JAVA_HOME:"=%"`) with the path for the desired Java installation.  ***Note ▪*** *The "flexnetls.settings" file includes both a JAVA_HOME and a JRE_HOME setting. If both values are set, JAVA_HOME is used.*
Optional Settings	
PORT	The listening port used by the license server. (If no value is specified here, the server uses 7070 by default.) If the machine on which the license server is running uses multiple network interfaces, you can use the `PORT` option to specify the interface that you want the license server to use. Simply include the IP address for the interface in square brackets, as shown in this example: `PORT=[127.0.0.1].1443`

Table 2-3 ▪ Local Settings for the License Server on Windows (cont.)

Setting	Description
ACTIVE_HOSTID	The hostid to use for the license server. The hostid can only be set using a configuration file if no hostid has yet been specified for the license server. Once a hostid has been set, it can only be changed using the FlexNet License Server Administrator or the FlexNet License Server Manager. The syntax is *value/type* (for example, `7200014f5df0/ETHERNET`). If no value is specified for this setting and no active hostid has yet been set for the license server, the license server uses, by default, the first available Ethernet address on the machine. If using a dongle ID or a hostid other than the first available Ethernet address, specify it here. For more information about hostids, see Understanding Hostids.  ***Important ▪*** *It is not recommended to change the hostid of a license server that has licenses mapped in the back office (FlexNet Operations). If the hostid is changed to a value that is different to that specified for the license server in the back office, any existing licenses mapped to the license server that is locked to the old hostid in the back office will be orphaned.* *To prevent this from happening, it is best practice to return the license server in the back office. During the return operation, the producer can transfer the licenses to a different device; this can be the same machine with the desired hostid. After the transfer, wait for the license server to synchronize with the back office (server synchronization occurs based on synchronization policies or on-demand).*  ***Important ▪*** *If a server hostid has been set using `ACTIVE_HOSTID`, the `server.hostType.order` and `licensing.enableBuiltinHostId` properties (if set) are ignored.*
HTTPS_SERVER_CONFIG	The path to the HTTPS “server” configuration file used to support *incoming* HTTPS from client devices.  ***Note ▪*** *The “server” configuration file is being deprecated and will be removed in a future release. Instead of the “server” configuration file, specify parameters to access the truststore file in the `https-in` setting in `local-configuration.yaml`. For more information about HTTPS setup on the license server, see Incoming HTTPS in the More About License Server Functionality chapter.*

Table 2-3 ▪ Local Settings for the License Server on Windows (cont.)

Setting	Description
HTTPS_CLIENT_CONFIG	The path to the HTTPS "client" configuration file used to support *outgoing* HTTPS communication to FlexNet Operations.  ***Note ▪*** *The "client" configuration file is being deprecated and will be removed in a future release. Instead of the "client" configuration file, specify parameters to access the truststore file in the* `https-out` *setting in* `local-configuration.yaml`*. For more information about HTTPS setup on the license server, see Outgoing HTTPS in the More About License Server Functionality chapter.*
SERVER_ALIAS	A user-defined name (sometimes called *host name*) for the license server. This name is added to server's capability requests to the back office, where it is then saved and used to identify the license server in the FlexNet Operations Producer and End User portals. One important use for this setting is that the alias can be included in the initial capability request sent at server registration, providing a helpful name by which users can identify the new server in the portals. If no alias is sent at registration, the server is identified by its hostid. For more information about hostids, see Understanding Hostids.
EXTENDED_SUFFIX	The suffix used for the Extended Host ID feature. Contact the software producer for details.
EXTRA_SYSPROPERTIES	One or more system properties (each in -D*key=value* format) that are passed to the Java Runtime system. The license server depends on the Java Runtime Environment to support certain network functionality such as specifying the HTTP proxy. For example, if you plan to have the license server communicate with the back office through an HTTP proxy, use this setting to identify the proxy parameters needed to configure the server. (For details, see Proxy Support for Communication with the Back Office in the More About License Server Functionality chapter.) The following shows example proxy parameters listed as -D system properties for this setting: `EXTRA_SYSPROPERTIES="-Dhttp.proxyHost=10.90.3.133 -Dhttp.proxyPort=3128 -Dhttp.proxyUser=user1a -Dhttp.proxyPassword=user1apwd35"` The entire set of parameters must be enclosed in double quotations, even if you specify only a single parameter, such as `EXTRA_SYSPROPERTIES="-Dhttp.proxyHost=10.90.3.133"`.

Table 2-3 ▪ Local Settings for the License Server on Windows (cont.)

Setting	Description
BACKUP_SERVER_HOSTID	The hostid of the back-up license server in a failover configuration with the current license server (as the main server), if the back-up server is "unknown"—that is, not registered—in FlexNet Operations. This setting adds the back-up server's hostid to the capability request sent by the current license server to the back office. The back office then automatically registers the back-up server in a failover configuration with the main server. This process saves the extra step of having to manually register the failover pair in FlexNet Operations. For more information, see License Server Failover in the More About License Server Functionality chapter. Make sure the back-up server's hostid is the same type (for example, "ETHERNET") as the hostid type of the current (main) license server. You might need to add this setting manually if it is not included as an available option in the settings file. For more information about hostids, see Understanding Hostids.

3. Open a command window as the Administrator, and navigate to the installation directory.
4. Execute `flexnetls.bat -install` to install the license server as a service.
5. You are prompted to install the MetaDaemon service (required for resilient hostid handling on physical machines):

   ```
   Do you want to install the MetaDaemon service? (Y/N)
   ```

   - Select **Y** if the producer has enabled resilient hostid handling.

     If **Y** is selected, a confirmation message indicates that the license server service and the MetaDaemon are being installed:

     ```
     Installing Meta Daemon Service FNLS-METADAEMON-<producer-name>[1.0.0]
     Installing service FNLS-<producer-name>
     ```

   - Select **N** if the producer has not enabled resilient hostid handling.

     If **N** is selected, a confirmation message indicates that only the license server service is being installed:

     ```
     Installing service FNLS-<producer-name>
     ```

Tip ▪ *For more information about resilient hostid handling, see Resilient Hostid Handling.*

6. Execute `flexnetls.bat -start` to start the service.

   Depending on the selection in the previous step, the corresponding confirmation message is displayed:

   - If only the license server service is installed:

     ```
     Service FNLS-<producer-name> successfully started
     ```

   - If both services are installed:

```
Meta Daemon Service FNLS-METADAEMON-<producer-name>[1.0.0] successfully started
Service FNLS-<producer-name> successfully started
```

Note ▪ *Depending on the security policies defined on your device, a warning message might be displayed, warning you to only run scripts that you trust. If such as warning is displayed, confirm that you want to allow each script to be executed.*

7. Confirm that the service is running by doing one of the following:
   - Execute `flexnetls.bat -status`. (The output should be `Service running`.)
   - In the Windows Services window (`services.msc`), check that the service **FlexNet License Server-*producer_name*** has started.
8. If you want to stop the service, run `flexnetls.bat -stop`. This will stop the license server service and the MetaDaemon service, if running.
9. To view the license server log, navigate to the server's logging directory (by default, `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*producer_name*`\logs`), and review the contents of the appropriate `.log` file.

Note ▪ *Once you install the license server as service, you cannot use any of the "flexnetls" non-service command-line options, such as "console" or "install".*

Running the InstallAnywhere-Based Installer

This section explains how to install and start the license server using the InstallAnywhere-based installer.

The InstallAnywhere-based installer is an .exe called `FlexnetLicenseServerWizardInstaller.exe`. If your producer has provided you with a zip called `flexnetls-x64_windows-<version>.zip`, `FlexnetLicenseServerWizardInstaller.exe` will be located in the `server` folder inside the zip, alongside with `producer-settings.xml`.

Note ▪ *The installer automatically installs Java 21, which is required for the license server to run.*

Task ***To install and start the license server as a Windows service***

1. Unpack the files received from the software producer into an installation directory.

   Ensure that `FlexnetLicenseServerWizardInstaller.exe` and `producer-settings.xml` are located in the same folder.
2. Double-click `FlexnetLicenseServerWizardInstaller.exe`. InstallAnywhere guides you through the installation.
3. During the installation, you are prompted to provide the following information:
   a. **Service Name**—A customizable display name for the license server. This name will appear in the Task Manager under **Services**.

Default: `FlexNet Embedded Local License Server`.

b. **Port**—Specify the port number on which you want the server to run over HTTP.

Default: 7070.

c. **Logging Threshold**—Defines the logging threshold for the license server service. Available values: FATAL, ERROR, WARN, INFO, LICENSING, POLICY, or DEBUG.

Default: INFO.

d. **Https Certificate Path**—Specify the path to the HTTPS certificate file (.pfx or .p12). This is required only if you intend the server to run in HTTPS mode.

Default: `C:\ProgramData\flexnetls\fnedemo`

e. **Certificate Password**—Enter the password for the provided certificate. This is required only if you intend the server to run in HTTPS mode.

f. **HTTPS Port**—Specify the port number on which you want the server to run over HTTPS.

Default: 1443.

4. When InstallAnywhere shows the **Install Complete** dialog, click **Done** to close the installer.

The license server is installed as a service and starts automatically.

5. Confirm that the service is running: In the Windows Services window (`services.msc`), check that the service **FlexNet License Server-*producer_name*** has started.

6. To view the license server log, navigate to the server's logging directory (by default, `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`*`\logs`), and review the contents of the appropriate `.log` file.

If you want to stop the service, stop it in the Windows Services window.

Configure, Install, and Start on Linux

The following instructions describe how to use systemd to configure, install, and run the local license server as a Linux service:

- Files Required for Installation Using systemd
- Task Overview
- Verify systemd-Enablement on Linux System
- Run the Install Script
- Manage the Service

The content in this section assumes familiarity with the use of the systemd system for managing Linux services. You can find information about systemd at the following site:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Managing_Services_with_systemd.htm

Files Required for Installation Using systemd

The following files, distributed with the license server, support the installation and running of the license server service under systemd:

- `install-systemd.sh`
- `install-functions.sh`
- `producer-settings.xml`

Residing in the same directory as `flexnetls.jar`, these files work together to generate the components required to install and start the license server service in a systemd configuration.

Optionally, the producer might provide an `rpm` or `deb` wrapper that copies these required files to the proper location on your device and then installs the license server service. If so, use the instructions provided by the producer for installing the service.

For a list of all files (aside from the systemd installer files) that can be distributed with the license server, see Local License Server Components.

Task Overview

The following is an overview of tasks used to install and run the license server as a Linux service:

- **Task 1:** Verify that the Linux system on which the license server is being installed is systemd-enabled (see Verify systemd-Enablement on Linux System).
- **Task 2:** Run the `install-systemd.sh` installer (or your own wrapper), updating certain default configuration settings in the command line as needed (see Run the Install Script).
- **Task 3:** Manage the service (see Manage the Service).
- **Task 4:** (Optional) Edit configuration settings post-installation as needed (see Post-Installation Configuration on Linux).

Verify systemd-Enablement on Linux System

Before proceeding with the installation, you can run the following command to determine whether the Linux system on which the license server is being installed is systemd-enabled:

```
systemctl
```

If the system is systemd-enabled, the command runs successfully and lists all active units.

Run the Install Script

You must run `install-systemd.sh` to create and start the license server service using either the default configuration for the service or a configuration customized through command-line options. See the following sections for details:

- About the Install Script
- Install and Start the Service with the Default Configuration

- Install and Start the Service with a Modified Configuration
- Configuration Values Editable from the Command Line

The commands described in these next sections are run from the directory in which the systemd script files (see About the Install Script) and `flexnetls.jar` reside.

Prerequisite

You must run the install script as a root or `sudo` user.

About the Install Script

When run, the install script `install-systemd.sh` performs the following:

- Generates the service unit file called `/etc/systemd/system/flexnetls-`*`producer_name`*`.service`.
- On virtual machines, installs the VMUUID license daemon (also referred to as `fnlicense-daemon`) required for VM UUID reading (also required for resilient hostid handling, if enabled).
- Generates the configuration unit file called `/etc/systemd/system/flexnetls-`*`producer_name`*`.service.d/flexnetls.conf`. This file contains the configuration settings needed to start the service. It is created with default values, but you can include options in the install-script command line to generate this file with custom values. See Install and Start the Service with a Modified Configuration.

  You can also edit this configuration file post-installation as needed, as described in Post-Installation Configuration on Linux.
- Generates a `local-configuration.yaml` file in the `/opt/flexnetls/`*`producer`* directory. This file contains optional settings specific to the local service environment. Generally, these settings are initially disabled; you can edit or enable them as needed once the service is installed. See Edit “local-configuration.yaml” (Linux) for details.
- By default, enables the standard Syslog process for logging on the license server service.
- Starts the license server service.

Install and Start the Service with the Default Configuration

The following command installs and starts the license server service using the default configuration. For more about the default values used to configure the service, Configuration Values Editable from the Command Line.

Task ***To install and start the license server service using the default configuration***

1. Run the following:

   ```
   sudo ./install-systemd.sh
   ```

2. If you are installing the license server on a virtual machine, you will be prompted to install the VMUUID license daemon (also referred to as `fnlicense-daemon`):

   ```
   Install VMUUID license daemon? (Y/N)
   ```

Select **Y** to enable the VMUUID license daemon. It is required to enable VM UUID reading and for resilient hostid handling on virtual machines.

***Tip** ▪ For more information about resilient hostid handling, see Resilient Hostid Handling.*

***Note** ▪ If resilient hostid handling is enabled, ensure that the virtual machine has a valid UUID. You can use the **/hostids** REST endpoint to verify the hostids available to the license server.*

Install and Start the Service with a Modified Configuration

The following command installs and starts the license server service using a configuration customized through one or more specified command-line options. For a description of the configuration values that can be updated from the install-script command line, see the next section, Configuration Values Editable from the Command Line.

Task** **To install and start the license server service with updates to the default configuration

1. Run the install script, specifying one or more command-line options to change specific configuration settings (for example, the `user` value, as shown here):

   ```
   sudo ./install-systemd.sh  --user flexnetls01
   ```

2. If you are installing the license server on a virtual machine, you will be prompted to install the VMUUID license daemon (also referred to as `fnlicense-daemon`):

   ```
   Install VMUUID license daemon? (Y/N)
   ```

   Select **Y** to enable the VMUUID license daemon. It is required to enable VM UUID reading and for resilient hostid handling on virtual machines.

   ***Tip** ▪ For more information about resilient hostid handling, see Resilient Hostid Handling.*

   ***Note** ▪ If resilient hostid handling is enabled, ensure that the virtual machine has a valid UUID. You can use the **/hostids** REST endpoint to verify the hostids available to the license server.*

Configuration Values Editable from the Command Line

Use the following options with the `./install-system.sh` command to edit current configuration settings for the license server service. (Running `./install-system.sh --help` also lists these command-line options.)

Table 2-4 ▪ Configuration Values Editable from the Command Line

Setting	Description
--program-dir *dir*	The installation location of the license server service (default is `/opt/flexnetls/`*`producer_name`*).
--data-dir *dir*	The location of trusted storage (default is `/var/opt/flexnetls/`*`producer_name`*). This overrides the `server.trustedStorageDir` value in the `producer-settings.xml`.
--user *user_name*	The user name under which the service runs (default is `flexnetls`).
--group *group_name*	The group name under which the service runs (default is `flexnetls`).
--java_home *path* **or** **--jre_home *path***	The path for JDK or JRE installation that the license server should use. By default, the license server uses the value of your `JAVA_HOME` or `JRE_HOME` system environment variable, whichever is defined on your device, to determine the Java installation location. However, if you want the license server to use different Java installation on your system, provide the explicit path for the installation as either the `java_home` or `jre_home` value. (This override pertains to the license server only; your device in general continues to use the Java installation defined by the system environment variable.)
--disableswagger	Disables the Swagger endpoint. (When running `./install-systemd.sh` without `-disableswagger`, the Swagger endpoint is enabled by default.)
--enable-read-timeout	Enables the socket read timeout for incoming client connections. When enabled, the license server closes connections that do not complete data transmission within the configured timeout period. This option can help prevent stalled or incomplete client connections from consuming server resources in environments with large numbers of client connections. Default value: false (disabled) **Dependencies**: ● When enabled without specifying `--read-timeout`, the server uses the default timeout value of 15000 milliseconds (15 seconds). ● To use a custom timeout value, specify `--read-timeout` together with this option. ● Specifying `--read-timeout` alone does not enable the timeout feature. See also Troubleshooting.

Table 2-4 ▪ Configuration Values Editable from the Command Line (cont.)

Setting	Description
--read-timeout *milliseconds*	Specifies the socket read timeout value, in milliseconds, for client connections when read timeout is enabled. If a client does not complete data transmission within the configured period, the server closes the connection. This setting is intended for use with `--enable-read-timeout` and can help prevent resource consumption caused by stalled or incomplete connections. Default value: 15000 milliseconds (15 seconds) Valid range: 1000 to 60000 milliseconds **Dependencies**: • This value takes effect only when `--enable-read-timeout` is enabled. • If specified without `--enable-read-timeout`, the configured value is stored but the read timeout remains disabled. See also Troubleshooting.
--log-client-ip-enabled	Controls whether client IP addresses are recorded in the `flexnetls.log` log file. Enable this option to include client IP information in server logs for troubleshooting and diagnostic purposes. Disable it if logging client IP addresses is not required. See also Troubleshooting.
The following two settings—"port" and "logging-threshold"—do not exist in the `.conf` file. However, if you update either setting using the command-line option, the new value is captured in the `local-configuration.yaml` file during its creation and is subsequently used by the license server service. If neither setting is updated, the service uses the default value (or the value specified in `producer-settings.xml`) for the setting.	
--port *port*	The listening port used by the license server service (default is `7070`). If the machine on which the license server is running uses multiple network interfaces, you can use the `--port` option to specify the interface that you want the license server to use. Simply include the IP address for the interface in square brackets, as shown in this example: `--port [127.0.0.1].1443`
--logging-threshold *level*	The lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`. For example, if `FATAL` is set, only messages about fatal events are recorded. However, if `WARN` is set, fatal-event, error, and warning messages are recorded. (Default is `INFO`.) The `POLICY` threshold records information about the checkout process when feature selectors are used to filter features. For more information about logging levels, see section Logging Policies in Reference: License Server Policy Settings.

Manage the Service

Once the license server service is installed, you can manage the service using the systemd command `systemctl`. The following describes basic management functions for the service:

- Obtain the Service Status
- Stop the Service
- Start the Service
- Restart the Service
- Re-read All systemd Unit Files

Prerequisite

You must perform `systemctl` tasks as a root or `sudo` user.

Obtain the Service Status

The following command obtains the “active” or “not active” status of license server service.

Task ***To obtain the status of the license server service***

Run the following command:

```
sudo systemctl -l status flexnetls-producer_name
```

The `-l` switch disables truncation of lines in the output.

The following shows example status output. The current service status is highlighted in this excerpt:

```
flexnetls-fnedemo.service - FlexnetLS Local License Server.
Loaded: loaded (/etc/systemd/system/flexnetls-fnedemo.service; enabled; vendor preset: disabled)
Drop-In: /etc/systemd/system/flexnetls-fnedemo.service.d
└─flexnetls.conf
Active: active (running) since Tue 2018-11-22 14:28:25 GMT; 18h ago
...
```

Stop the Service

The following command stops the license server service.

Task ***To stop the license server service***

Use this command:

```
sudo systemctl stop flexnetls-producer_name
```

Start the Service

The following command starts the license server service.

Task ***To start the license server service***

Use this command:

```
sudo systemctl start flexnetls-producer_name
```

Restart the Service

The following command stops and then restarts the license server service without re-reading all the systemd unit files related to all services. This command is useful if you have made changes to the `local-configuration.yaml` file (a non-systemd file) and want the service to read this file, but not all the systemd unit files (see Edit “local-configuration.yaml” (Linux)).

Task ***To restart the license server service***

Run the following command:

```
sudo systemctl restart flexnetls-producer_name
```

Re-read All systemd Unit Files

The following command is used to re-read all systemd unit files related to all services. This command is useful when you have made changes manually to the `flexnetls.conf` file and want to apply the changes before restarting the service (see Edit the .conf File Manually).

Task ***To re-read all systemd unit files for all services***

Run the following command:

```
sudo systemctl daemon-reload
```

Local License Server Components

The following basic components are usually included in the FlexNet Embedded local license server installation. The location of these components is not listed, as the software producer determines their location within the server’s directory structure. Additionally, the software producer might provide a customized version of certain components, using names different from the ones listed here.

In a Windows Installation

These components are included in the license server installation on Windows.

Table 2-5 ▪ FlexNet Embedded Local License Server Components in an Installation on Windows

Component	Description
flexnetls.jar	The Java executable file for the FlexNet Embedded local license server.
flexnetls.bat	The batch file used to start the FlexNet Embedded local license server.
producer-settings.xml	The configuration file, generated by the producer, that defines policies for license server operations. Some of these settings can be overridden using the FlexNet License Server Administrator command-line tool or the FlexNet License Server Manager.
flexnetls.settings	The local settings file containing information needed to run the license server in your local environment. Any of these settings are editable.
local-configuration.yaml	A file containing optional local settings.  ***Note ▪*** *In a future release, this file will replace flexnetls.settings and the https configuration files.*
flexnetlsw.exe	File used to run the license server as a Windows service.
Truststore file provided by producer	(Optional) File containing root and intermediate certificates used to validate an HTTPS connection to FlexNet Operations. If the producer is using the Revenera-hosted version of FlexNet Operations, no truststore file is required. If the producer is using the “on-premises” version of FlexNet Operations and the server certificate has been signed by an unknown (private) certificate authority, then the truststore file is required.
regid.2009-06.com(...).swidtag	The software ID tag file for use with software asset management tools.
• **backofficeofflinesynctool.bat** • **serverofflinesynctool.bat**	(Optional) The set of tools allowing the license server to perform offline synchronization to the back office. The software producer determines whether to provide these tools. The tools also require the `flxPublicTools.jar`, `EccpressoAll.jar`, and `commons-codec-1.9.jar` files.
• **flexnetlsadmin.bat** • **flexnetlsadmin.jar**	(Optional) Files used to run the FlexNet License Server Administrator command-line tool. The software producer determines whether to provide this tool.

Table 2-5 ▪ FlexNet Embedded Local License Server Components in an Installation on Windows (cont.)

Component	Description
FlexnetLicenseServerManager-<version>.zip (Docker container) and/or FlexNetLicenseServerManagerInstaller-<version>.zip (executable)	(Optional) The FlexNet License Server Manager is provided as a separate package. The License Server Manager is available in two formats: ● Docker container (requires Docker to be installed) ● Executable (.exe) The software producer determines whether to provide this tool. For more information, see the FlexNet License Server Manager Guide, available from the Revenera Product Documentation site (https://docs.revenera.com).
flexnetls-hostmetadata.exe	(Optional) Your producer may have included this file to enable resilient hostid handling on physical machines. The executable collects physical machine identifiers during hostid validation. It must be placed in the `server` directory.  ***Tip*** *▪ For more information about resilient hostid handling, see Resilient Hostid Handling.*

In a Linux Installation

These components are included in the license server installation on Linux.

Table 2-6 ▪ FlexNet Embedded Local License Server Components in an Installation on Linux

Component	Description
flexnetls.jar	The Java executable file for the FlexNet Embedded local license server.
producer-settings.xml	The configuration file, generated by the producer, that defines policies for license server operations. Some of these settings can be overridden using the FlexNet License Server Administrator command-line tool or the FlexNet License Server Manager.
install-functions.sh **install-systemd.sh**	Files used to install and run the FlexNet Embedded local license server as a Linux service using systemd, as described in this chapter.
flexnetls.conf	A file generated during installation and containing the configuration settings needed to start the service. The file can be created with default values, or you can include options in the install-script command line to generate this file with custom values (see Configure, Install, and Start on Linux). You can also update the settings post-installation, as described in Post-Installation Configuration on Linux.

Table 2-6 ▪ FlexNet Embedded Local License Server Components in an Installation on Linux (cont.)

Component	Description
local-configuration.yaml	A file generated in the installation and containing optional settings specific to the local service environment (see Configure, Install, and Start on Linux). For the most part, these settings are disabled when the file is created, but you can edit or enable them as needed once the service is installed.
Truststore file provided by producer (optional)	Optional file containing root and intermediate certificates used to validate an HTTPS connection to FlexNet Operations. If the producer is using the Revenera-hosted version of FlexNet Operations, no truststore file is required. If the producer is using the "on-premises" version of FlexNet Operations and the server certificate has been signed by an unknown (private) certificate authority, then the truststore file is required.
regid.2009-06.com(...).swidtag	The software ID tag file for use with software asset management tools.
• **backofficeofflinesynctool** • **serverofflinesynctool**	(Optional) The set of tools allowing the license server to perform offline synchronization to the back office. The software producer determines whether to provide these tools. The tools also require the `flxPublicTools.jar`, `EccpressoAll.jar`, and `commons-codec-1.9.jar` files.
• **flexnetlsadmin.sh** • **flexnetlsadmin.jar**	(Optional) Files used to run the FlexNet License Server Administrator command-line tool. The software producer determines whether to provide this tool.
FlexnetLicenseServerManager-<version>.zip	(Optional) The FlexNet License Server Manager is provided as a separate package. The package is a Docker container which holds the image `revenera/flsm`. The software producer determines whether to provide this tool. For more information, see the FlexNet License Server Manager Guide, available from the Revenera Product Documentation site (https://docs.revenera.com).
• **bin** directory	(Optional) Special set of files needed to run `fnlicense-daemon`, a service used to capture the VM UUID on those Linux virtual platforms that do provide an appropriate mechanism for retrieving this ID. The `install-systemd.sh` script will check the presence of a Linux VM (using the `systemd-detect-virt` tool), and will then offer to install the systemd service `fnlicense-daemon`. If accepted, `fnlicense-daemon` will be installed. This service is socket-activated, so it will only run when needed.

Setting the Server Time Zone

The license server can either use its default time zone or Coordinated Universal Time (UTC) for determining a feature's expiry date and start date. Your producer can configure the time zone using the `licensing.defaultTimeZone` setting in the `producer-settings.xml` file. Valid values:

- **UTC**— If `UTC` is set, a feature's start date is the start of the specified day in Coordinated Universal Time (UTC). Equally, a feature will expire at the end of the day of the configured expiry date in UTC time. This is the default value.
- **SERVER**—If `SERVER` is set, a feature's start date is the start of the specified day in the server's default time zone. Equally, a feature will expire at the end of the day of the configured expiry date in the server's default time zone.

Changes to the time zone setting affect features as they are being provisioned onto the server. It does not affect features that are already on the server.

If you need to change your license server's time zone setting, contact the software producer.

For a description of license server policies, see Reference: License Server Policy Settings.

Logging Functionality

The logging functionality that is available for the local license server is described in detail in Logging Functionality on the Local License Server. The appendix covers how to configure the logging style and how to integrate license server logging with external systems.

Editing the Local Settings Post-Installation

The license server installation provides (or provides a means to generate) a local settings file that configures the license server for your specific environment. You can use the file as is with its default settings, or you can modify the settings as needed to reflect your environment. The previous section Configuring, Installing, and Starting the Local License Server describes how to update these settings before server installation. Use the following instructions to edit the settings any time *after* installation.

- Post-Installation Configuration on Windows
- Post-Installation Configuration on Linux
- Configuring the Cipher Choice Mechanism

Post-Installation Configuration on Windows

If you need to modify the license server's local settings (defined in the `flexnetls.settings` and `local-configuration.yaml` files) *after* you have installed the server as a service on a Windows platform, use either of these steps:

- Edit "flexnetls.settings"
- Edit "local-configuration.yaml" (Windows)

Edit “flexnetls.settings”

For a list of settings of `flexnetls.settings` (located in the installation directory), see Local Settings for the License Server on Windows.

Note that the `rate-limit` setting (used to control the license server load) and `loggingStyle` (for specifying rollover, JSON formatting and timestamp behavior in the server log) are available only in the `local-configuration.yaml` file (see Edit “local-configuration.yaml” (Windows)).

Task ***To modify local settings after installing the license server as a service on Windows***

1. As an Administrator, open a command prompt window, and navigate to the directory where the license server is installed.
2. Open the `flexnetls.settings` file (located in the installation directory) in a text editor, and edit the settings as needed. (For example, you can uncomment settings or change existing values.) See the previous section Configure, Install, and Start on Windows for details about the editing process.
3. Save the file.
4. Execute `flenxetls.bat -update` to update the settings.
5. Execute `flexnetls.bat -start` to start the license server service, or restart it from the Windows Services window.

Note ▪ *Depending on the security policies defined on your device, a warning message might be displayed, warning you to only run scripts that you trust. If such as warning is displayed, confirm that you want to allow each script to be executed.*

Edit “local-configuration.yaml” (Windows)

Use the following procedure to update the `local-configuration.yaml` file (located in the same directory as `flexnetls.jar`). This file contains optional settings for the license server service showing the default values, but commented out.

Edit Settings in the “local-configuration.yaml” file

Use the following procedure to enable, edit, or disable settings in `local-configuration.yaml` file.

Task ***To edit settings in the “local-configuration.yaml’ file***

1. As an Administrator, open a command prompt window, and navigate to the directory where `local-configuration.yaml` is installed.
2. In a text editor, open `local-configuration.yaml`. Uncomment (or comment out) lines and edit setting values as needed. For a description of the settings, see the next section.
3. Save the file.

4. Execute `flexnetls.bat -start` to start the license server service, or restart it from the Windows Services window.

Note ▪ *Depending on the security policies defined on your device, a warning message might be displayed, warning you to only run scripts that you trust. If such as warning is displayed, confirm that you want to allow each script to be executed.*

Settings in the "local-configuration.yaml" File

The following settings can be edited directly in the `configuration-local.yaml` file. Note that a space after the colon is required for all entries.

Table 2-7 ▪ Settings in the "local-configuration.yaml" File

Setting	Description
port: ***port***	The override for the listening port used by the license server, as described in Configuration Values Editable from the Command Line. You can edit this override value as needed. If no override is specified (that is, the setting is commented out), the service uses the default port `7070` (or the value defined in `producer-settings.xml`).

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
active-hostid: *value/type*	The hostid to use for the license server. The hostid can only be set using a configuration file if no hostid has yet been specified for the license server. Once a hostid has been set, it can only be changed using the FlexNet License Server Administrator or the FlexNet License Server Manager. The syntax is *value/type* (for example, `7200014f5df0/ETHERNET`). If no value is specified for this setting and no active hostid has yet been set for the license server, the license server uses, by default, the first available Ethernet address on the machine. If using a dongle ID or a hostid other than the first available Ethernet address, specify it here. For more information about hostids, see Understanding Hostids.  ***Important ▪*** *It is not recommended to change the hostid of a license server that has licenses mapped in the back office (FlexNet Operations). If the hostid is changed to a value that is different to that specified for the license server in the back office, any existing licenses mapped to the license server that is locked to the old hostid in the back office will be orphaned.* *To prevent this from happening, it is best practice to return the license server in the back office. During the return operation, the producer can transfer the licenses to a different device; this can be the same machine with the desired hostid. After the transfer, wait for the license server to synchronize with the back office (server synchronization occurs based on synchronization policies or on-demand).*  ***Important ▪*** *If a server hostid has been set using `active-hostid`, the `server.hostType.order` and `licensing.enableBuiltinHostId` properties (if set) are ignored.* For more information about hostids, see Understanding Hostids.

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
backup-hostid: *value/type*	The hostid of the back-up license server in a failover configuration with the current license server (as the main server), if the back-up server is "unknown"—that is, not registered—in FlexNet Operations. The syntax is *value/type* (for example, `7200014f5df0/ETHERNET`). For more information about hostids, see Understanding Hostids. This setting adds the back-up server's hostid to the capability request sent by the current license server to the back office. The back office then automatically registers the back-up server in a failover configuration with the main server. This process saves the extra step of having to manually register the failover pair in FlexNet Operations. For more information, see License Server Failover in the More About License Server Functionality. Make sure the back-up server's hostid is the same type (for example, "ETHERNET") as the hostid type of the current (main) license server.
extended-suffix: *suffix*	The suffix used for the Extended Host ID feature. Contact the software producer for details.
server-alias: *alias*	A user-defined name (sometimes called *host name*) for the license server. This name is added to server's capability requests to the back office, where it is then saved and used to identify the license server in the FlexNet Operations Producer and End User portals. One important use for this setting is that the alias can be included in the initial capability request sent at server registration, providing a helpful name by which users can identify the new server in the portals. If no alias is sent at registration, the server is identified by its hostid.
logging-threshold: *level*	The override for the lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`. You can edit this override value as needed. If no override is specified (that is, the setting is commented out), the service uses the default level `INFO` (or the value defined in `producer-settings.xml`). For more information about logging levels, see section Logging Policies in Reference: License Server Policy Settings.

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
loggingStyle: *style*	The logging style determines rollover, JSON formatting and timestamp behavior. If no logging style is specified, the default `DAILY_ROLLOVER` is used. **Logging style values:** `DAILY_ROLLOVER`—The log file is closed at midnight local time, compressed with gzip, and a new log file is started. Timestamp values use the local time zone. This is the default if no logging style is specified. `DAILY_ROLLOVER_UTC`—Same as `DAILY_ROLLOVER`, but with UTC timestamp. `CONTINUOUS`—Legacy value. Rollover is handled as `DAILY_ROLLOVER`. `CONTINUOUS_UTC`—Legacy value. Rollover is handled as `DAILY_ROLLOVER_UTC`. `JSON_ROLLOVER`—Logs are formatted using JSON. The log file is closed at midnight local time, compressed with gzip (suitable for filebeat), and a new log file is started. Always uses ISO 8601 UTC timestamps. `JSON`—Logs are emitted as JSON to stdout only (suitable for Docker). Always uses ISO 8601 UTC timestamps.  ***Note ▪*** *The Docker-friendly `JSON` logging style is not supported when the local license server is run as a Windows or Linux service (because it only writes to stdout). If set, the style will be changed to `DAILY_ROLLOVER`.*
jni-helper-directory: *path*	If the default location for the JNI shared library (`/tmp`) is not suitable (for example, because execute permission is required), use this setting to supply an alternative directory path. The directory has to exist and be writable by the server.
disable-xforwarded-for: *value*	Disable `X-Forwarded-For` (XFF) headers (`disable-xforwarded-for=true`) if a trusted proxy is not present. This setting provides useful security hardening if the `security.ip.whitelist` license server policy has been configured and no gateway or reverse proxy server is in use.  ***Important ▪*** *Setting `disable-xforwarded-for=true` when the license server is behind a reverse proxy will result in:* • *Incorrect IP addresses in access logs* • *Swagger user interface won't work correctly*

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
https-in: *parameters*	Parameters used to access the "server" certificate from a certificate authority (CA) to support *incoming* HTTPS from client devices. For details, see Incoming HTTPS in the More About License Server Functionality chapter. Parameters also control the enabled cipher list. For more information, see Configuring the Cipher Choice Mechanism. These parameters replace the "server" configuration file used for HTTPS in previous FlexNet Embedded license server versions. To use a "server" configuration file from a previous license server version (instead of these parameters), specify the path to the "server" configuration file in the `HTTPS_SERVER_CONFIG` setting in the `flexnetls.settings` file. For more information, see Edit "flexnetls.settings".  ***Note ▪*** *The "server" configuration file is being deprecated and will be removed in a future release.*
https-out: *parameters*	Parameters to access the truststore containing root and intermediate certificates to validate the license server's *outgoing* HTTPS communication to FlexNet Operations. This setting is not required for Revenera-hosted FlexNet Operations instances. If you have the "on-premises" version of FlexNet Operations and the HTTPS server certificate has been signed by an unknown (private) certificate authority, you must define this setting. For details, see Outgoing HTTPS in the More About License Server Functionality chapter. These parameters replace the "client" configuration file used for HTTPS in previous FlexNet Embedded license server versions. To use a "client" configuration file from a previous license server version (instead of these parameters), specify the path to the "client" configuration file in the `HTTPS_CLIENT_CONFIG` setting in the `flexnetls.settings` file. For more information, see Edit "flexnetls.settings".  ***Note ▪*** *The "client" configuration file is being deprecated and will be removed in a future release.*

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
rate-limit: *value*	The maximum number of capability requests per second that the license server will accept. By default, the local license server has no limit for the number of capability requests it will accept at a given time. However, if a license server experiences a high load of capability requests, a degradation in response time can result. This option is intended for use only in situations where this type of exceptional load is expected, as such a limit helps to enable consistent response times for accepted requests. A number of factors affect the rate at which the server is capable of handling requests, including the specification of the server hardware and the complexity of the requests. Therefore, the `rate-limit` value should be determined only through load-testing on a mirror of the production environment using a representative client load. If this setting is currently not included in the `.yaml` file, you can manually add it to the file to define the necessary rate limit.
readTimeoutEnabled	Enables the socket read timeout for incoming client connections. When enabled, the license server closes connections that do not complete data transmission within the configured timeout period. This option can help prevent stalled or incomplete client connections from consuming server resources in environments with large numbers of client connections. Default value: false (disabled) **Dependencies**: ● When enabled without specifying `readTimeout`, the server uses the default timeout value of 15000 milliseconds (15 seconds). ● To use a custom timeout value, specify `readTimeout` together with this option. ● Specifying `readTimeout` alone does not enable the timeout feature. See also Troubleshooting.

Table 2-7 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
readTimeout: ***milliseconds***	Specifies the socket read timeout value, in milliseconds, for client connections when read timeout is enabled. If a client does not complete data transmission within the configured period, the server closes the connection. This setting is intended for use with `readTimeoutEnabled` and can help prevent resource consumption caused by stalled or incomplete connections. Default value: 15000 milliseconds (15 seconds) Valid range: 1000 to 60000 milliseconds **Dependencies**: ● This value takes effect only when `readTimeoutEnabled` is enabled. ● If specified without `readTimeoutEnabled`, the configured value is stored but the read timeout remains disabled. See also Troubleshooting.
logClientIpEnabled	Controls whether client IP addresses are recorded in the `flexnetls.log` log file. Enable this option to include client IP information in server logs for troubleshooting and diagnostic purposes. Disable it if logging client IP addresses is not required. Default value: true (enabled) See also Troubleshooting.

Post-Installation Configuration on Linux

The following sections describe how to edit current settings in the `flexnetls.conf` and `local-configuration.yaml` files after the license server service is installed:

- Edit Settings in "flexnetls.conf"
- Edit the DEFINES Setting
- Edit "local-configuration.yaml" (Linux)
- Define Command-Line Options for HTTPS Configuration Files

Edit Settings in "flexnetls.conf"

You can use either of the following methods to update configuration settings in the `/etc/systemd/system/flexnetls-`*`producer_name`*`.service.d/flexnetls.conf` unit file:

- Edit the .conf File Manually
- Re-install the Service with Command-line Options to Update Settings

Edit the .conf File Manually

Use the following procedure to manually edit the contents of the `flexnetls.conf` unit file.

Task ***To edit the flexnetls.conf file manually***

1. Stop the license server service:

   ```
   sudo systemctl stop flexnetls-producer_name
   ```

2. As a root or `sudo` user, edit the `/etc/systemd/system/flexnetls-producer_name.service.d/flexnetls.conf` file in a text editor (for example, gedit or GNU nano).

3. Once the edits are complete, run the following command to re-read all the systemd unit files to capture the configuration changes:

   ```
   sudo systemctl daemon-reload
   ```

4. Start the license server service:

   ```
   sudo systemctl start flexnetls-producer_name
   ```

Re-install the Service with Command-line Options to Update Settings

The following command reinstalls the license server service, using command-line options to update specific settings in the `flexnetls.conf` file. For a description of the configuration settings that can be updated from the install-script command line, see Configuration Values Editable from the Command Line.

```
sudo ./install-systemd.sh  --user flexnentls1 --group flexnetls1
```

If you include the `--overwrite` option in this command, the re-installation generates a new `.conf` file and a new `.yaml` file with the specified updates; any changes you previously made to these files are lost.

Edit the DEFINES Setting

By design, the `DEFINES` setting in the `flexnetls.conf` unit file has no corresponding command-line option to update its value; therefore, you need to edit the setting manually in the configuration file. For instructions on manually editing this file, see Edit the .conf File Manually.

This setting is used to define one or more system properties (each in -D*key=value* format) that the license server needs to pass to the Java Runtime system to support certain network functionality, such as an HTTP proxy.

For example, if you plan to have the license server communicate with the back office through an HTTP proxy, use this setting to identify the proxy parameters needed to configure the server. (For details, see Proxy Support for Communication with the Back Office in the More About License Server Functionality chapter.) The following shows example proxy parameters listed as -D system properties for this setting:

```
DEFINES="-Dhttp.proxyHost=10.90.3.133 -Dhttp.proxyPort=3128 -Dhttp.proxyUser=user1a
-Dhttp.proxyPassword=user1apwd35"
```

By default, this setting has no properties defined.

Edit “local-configuration.yaml” (Linux)

Use the following procedure to update the `local-configuration.yaml` file generated in the `/opt/flexnetls/producer` directory during installation. This file contains optional settings for the license server service that are, by default, disabled.

Edit Settings in the “local-configuration.yaml” file

Use the following procedure to enable, edit, or disable settings in `local-configuration.yaml` file.

Task ***To edit settings in the “local-configuration.yaml’ file***

1. As a root or `sudo` user, edit the `local-configuration.yaml` file in an editor such `gedit`. Uncomment (or comment out) lines and edit setting values as needed. For a description of the settings, see the next section.
2. Run the following command to stop and restart the service to apply the configuration changes:

   ```
   sudo systemctl restart flexnetls-producer_name
   ```

Settings in the “local-configuration.yaml” File

The following settings can be edited directly in the `configuration-local.yaml` file. Note that a space after the colon is required for all entries.

Table 2-8 ▪ Settings in the “local-configuration.yaml” File

Setting	Description
port: ***port***	The override for the listening port used by the license server, as described in Configuration Values Editable from the Command Line. You can edit this override value as needed. If no override is specified (that is, the setting is commented out), the service uses the default port `7070` (or the value defined in `producer-settings.xml`).

Table 2-8 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
active-hostid: *value/type*	The hostid to use for the license server. The hostid can only be set using a configuration file if no hostid has yet been specified for the license server. Once a hostid has been set, it can only be changed using the FlexNet License Server Administrator or the FlexNet License Server Manager. The syntax is *value/type* (for example, `7200014f5df0/ETHERNET`). If no value is specified for this setting and no active hostid has yet been set for the license server, the license server uses, by default, the first available Ethernet address on the machine. If using a dongle ID or a hostid other than the first available Ethernet address, specify it here. For more information about hostids, see Understanding Hostids.  ***Important ▪*** *It is not recommended to change the hostid of a license server that has licenses mapped in the back office (FlexNet Operations). If the hostid is changed to a value that is different to that specified for the license server in the back office, any existing licenses mapped to the license server that is locked to the old hostid in the back office will be orphaned.* *To prevent this from happening, it is best practice to return the license server in the back office. During the return operation, the producer can transfer the licenses to a different device; this can be the same machine with the desired hostid. After the transfer, wait for the license server to synchronize with the back office (server synchronization occurs based on synchronization policies or on-demand).*  ***Important ▪*** *If a server hostid has been set using `active-hostid`, the `server.hostType.order` and `licensing.enableBuiltinHostId` properties (if set) are ignored.*
backup-hostid: *value/type*	The hostid of the back-up license server in a failover configuration with the current license server (as the main server), if the back-up server is "unknown"—that is, not registered—in FlexNet Operations. The syntax is *value/type* (for example, `7200014f5df0/ETHERNET`). For more information about hostids, see Understanding Hostids. This setting adds the back-up server's hostid to the capability request sent by the current license server to the back office. The back office then automatically registers the back-up server in a failover configuration with the main server. This process saves the extra step of having to manually register the failover pair in FlexNet Operations. For more information, see License Server Failover in the More About License Server Functionality. Make sure the back-up server's hostid is the same type (for example, "ETHERNET") as the hostid type of the current (main) license server.

Table 2-8 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
extended-suffix: *suffix*	The suffix used for the Extended Host ID feature. Contact the software producer for details.
server-alias: *alias*	A user-defined name (sometimes called *host name*) for the license server. This name is added to server's capability requests to the back office, where it is then saved and used to identify the license server in the FlexNet Operations Producer and End User portals. One important use for this setting is that the alias can be included in the initial capability request sent at server registration, providing a helpful name by which users can identify the new server in the portals. If no alias is sent at registration, the server is identified by its hostid.
logging-threshold: *level*	The override for the lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`, as described in Configuration Values Editable from the Command Line. You can edit this override value as needed. If no override is specified (that is, the setting is commented out), the service uses the default level `INFO` (or the value defined in `producer-settings.xml`). For more information about logging levels, see section Logging Policies in Reference: License Server Policy Settings.
loggingStyle: *style*	The logging style determines rollover, JSON formatting and timestamp behaviour. If no logging style is specified, the default `DAILY_ROLLOVER` is used. **Logging style values:** `DAILY_ROLLOVER`—The log file is closed at midnight local time, compressed with gzip, and a new log file is started. Timestamp values use the local time zone. This is the default if no logging style is specified. `DAILY_ROLLOVER_UTC`—Same as `DAILY_ROLLOVER`, but with UTC timestamp. `CONTINUOUS`—Legacy value. Rollover is handled as `DAILY_ROLLOVER`. `CONTINUOUS_UTC`—Legacy value. Rollover is handled as `DAILY_ROLLOVER_UTC`. `JSON_ROLLOVER`—Logs are formatted using JSON. The log file is closed at midnight local time, compressed with gzip (suitable for filebeat), and a new log file is started. Always uses ISO 8601 UTC timestamps. `JSON`—Logs are emitted as JSON to stdout only (suitable for Docker). Always uses ISO 8601 UTC timestamps.  ***Note ▪*** *The Docker-friendly `JSON` logging style is not supported when the local license server is run as a Windows or Linux service (because it only writes to stdout). If set, the style will be changed to `DAILY_ROLLOVER`.*

Table 2-8 ▪ Settings in the “local-configuration.yaml” File (cont.)

Setting	Description
jni-helper-directory: *path*	If the default location for the JNI shared library (`/tmp`) is not suitable (for example, because execute permission is required), use this setting to supply an alternative directory path. The directory has to exist and be writable by the server.
disable-xforwarded-for: *value*	Disable `X-Forwarded-For` (XFF) headers (`disable-xforwarded-for=true`) if a trusted proxy is not present. This setting provides useful security hardening if the `security.ip.whitelist` license server policy has been configured and no gateway or reverse proxy server is in use.  ***Important ▪*** *Setting `disable-xforwarded-for=true` when the license server is behind a reverse proxy will result in:* ● *Incorrect IP addresses in access logs* ● *Swagger user interface won't work correctly*
https-in: *parameters*	Parameters used to access the “server” certificate from a certificate authority (CA) to support *incoming* HTTPS from client devices. For details, see Incoming HTTPS in the More About License Server Functionality chapter. Parameters also control the enabled cipher list. For more information, see Configuring the Cipher Choice Mechanism. These parameters replace the “server” configuration file used for HTTPS in previous FlexNet Embedded license server versions. To use a “server” configuration file from a previous license server version (instead of these parameters), you need to set up a command-line option to specify the file path and name. For more information, see Define Command-Line Options for HTTPS Configuration Files.  ***Note ▪*** *The “server” configuration file is being deprecated and will be removed in a future release.*

Table 2-8 ▪ Settings in the "local-configuration.yaml" File (cont.)

Setting	Description
https-out: *parameters*	Parameters to access the truststore containing root and intermediate certificates to validate the license server's *outgoing* HTTPS communication to FlexNet Operations. This setting is not required for Revenera-hosted FlexNet Operations instances. If you have the "on-premises" version of FlexNet Operations and the HTTPS server certificate has been signed by an unknown (private) certificate authority, you must define this setting. For details, see Outgoing HTTPS in the More About License Server Functionality chapter. These parameters replace the "client" configuration file used for HTTPS in previous FlexNet Embedded license server versions. To use a "client" configuration file from a previous license server version (instead of these parameters), you need to set up a command-line option to specify the file path and name. For more information, see Define Command-Line Options for HTTPS Configuration Files.  ***Note ▪*** *The "client" configuration file is being deprecated and will be removed in a future release.*
rate-limit: *value*	The maximum number of capability requests per second that the license server will accept. By default, the local license server has no limit for the number of capability requests it will accept at a given time. However, if a license server experiences a high load of capability requests, a degradation in response time can result. This option is intended for use only in situations where this type of exceptional load is expected, as such a limit helps to enable consistent response times for accepted requests. A number of factors affect the rate at which the server is capable of handling requests, including the specification of the server hardware and the complexity of the requests. Therefore, the `rate-limit` value should be determined only through load-testing on a mirror of the production environment using a representative client load. If this setting is currently not included in the `.yaml` file, you can manually add it to the file to define the necessary rate limit.

Table 2-8 ▪ Settings in the “local-configuration.yaml” File (cont.)

Setting	Description
readTimeoutEnabled	Enables the socket read timeout for incoming client connections. When enabled, the license server closes connections that do not complete data transmission within the configured timeout period. This option can help prevent stalled or incomplete client connections from consuming server resources in environments with large numbers of client connections. Default value: false (disabled) **Dependencies**: ● When enabled without specifying `readTimeout`, the server uses the default timeout value of 15000 milliseconds (15 seconds). ● To use a custom timeout value, specify `readTimeout` together with this option. ● Specifying `readTimeout` alone does not enable the timeout feature. See also Troubleshooting.
readTimeout: ***milliseconds***	Specifies the socket read timeout value, in milliseconds, for client connections when read timeout is enabled. If a client does not complete data transmission within the configured period, the server closes the connection. This setting is intended for use with `readTimeoutEnabled` and can help prevent resource consumption caused by stalled or incomplete connections. Default value: 15000 milliseconds (15 seconds) Valid range: 1000 to 60000 milliseconds **Dependencies**: ● This value takes effect only when `readTimeoutEnabled` is enabled. ● If specified without `readTimeoutEnabled`, the configured value is stored but the read timeout remains disabled. See also Troubleshooting.
logClientIpEnabled	Controls whether client IP addresses are recorded in the `flexnetls.log` log file. Enable this option to include client IP information in server logs for troubleshooting and diagnostic purposes. Disable it if logging client IP addresses is not required. Default value: true (enabled) See also Troubleshooting.

Define Command-Line Options for HTTPS Configuration Files

The `local-configuration.yaml` includes the `https-in` and `https-out` options used to define the parameters for the license server's HTTPS communications with FlexNet Embedded clients and FlexNet Operations. These two options replace the "server" and "client" HTTPS configuration files used in previous license server versions. However, you can still use these configuration files (instead of the `https-in` and `https-out` options) by providing command-line options in the `flexnetls.conf` unit file to identify the paths to these files.

Task ***To identify the path for an HTTPS configuration file***

Use the instructions in Edit the .conf File Manually to edit (or add) the `Environment="OPTIONS="` line in the `flexnetls.conf` unit file to specify one or both command-line options:

- To specify the command-line option that identifies the path and name for the "server" configuration file, enter the following:

  ```
  Environment="OPTIONS=--https-server-configuration filePathAndName"
  ```

- To specify the command-line option that identifies the path and name for the "client" configuration file path, enter the following:

  ```
  Environment="OPTIONS=--https-client-configuration filePathAndName"
  ```

- To specify both command-line options, enter the following:

  ```
  Environment="OPTIONS=--https-server-configuration filePathAndName --https-client-configuration
  filePathAndName"
  ```

Configuring the Cipher Choice Mechanism

When enabling HTTPS on a local license server, license server administrators have to be careful about the choice of protocols and TLS ciphers. Failure to do this will leave the license server vulnerable to attacks.

Older TLS protocol versions (SSLv3, TLSv1.1) should not be active, not even for protocol re-negotiation.

Older ciphers (RC4, RSA for key exchange) and export-grade ciphers should not be enabled. AES should use GCM mode rather than CBC in order to avoid padding attacks, and AEAD (Authenticated Encryption with Associated Data) should be used for integrity checking wherever possible.

The enabled protocols in the local license server are TLSv1.2 and TLSv1.3. The local license server will automatically enable TLSv1.3 if it's available.

All known vulnerable ciphers are disabled in the default configuration. The enabled cipher list can be controlled through the configuration property `tlsCipherSuites` in the `https-in` section of `local-configuration.yaml`:

```
# Choice of TLS cipher suites. One of MODERN, COMPATIBLE or WEAK.
tlsCipherSuites: COMPATIBLE
```

The following table describes the cipher suites that are available:

Table 2-9 ▪ Cipher Suites

Cipher Suite	Available Ciphers	Notes
MODERN TLS 1.3	`"TLS_AES_128_GCM_SHA256", "TLS_AES_256_GCM_SHA384", "TLS_CHACHA20_POLY1305_SHA256"`	There is a high chance of TLS handshake failures with MODERN unless all clients served by the local license server support TLS 1.3.
COMPATIBLE	`"TLS_AES_128_GCM_SHA256", "TLS_AES_256_GCM_SHA384", "TLS_CHACHA20_POLY1305_SHA256", "ECDHE-ECDSA-AES128-GCM-SHA256", "ECDHE-RSA-AES128-GCM-SHA256", "ECDHE-ECDSA-AES256-GCM-SHA384", "ECDHE-RSA-AES256-GCM-SHA384", "ECDHE-ECDSA-CHACHA20-POLY1305", "ECDHE-RSA-CHACHA20-POLY1305", "DHE-RSA-AES128-GCM-SHA256", "DHE-RSA-AES256-GCM-SHA384"`	Always use AES in GCM mode rather than CBC (to avoid padding attacks). Avoid deprecated and unsafe ciphers. This is the default setting.
WEAK	N/A	No constraints. Vulnerable ciphers are disabled at the JSSE level if at all. This choice would only be used when there is a need to communicate with old clients that get TLS handshake failures with COMPATIBLE.

Upgrading the Local License Server

You may need to upgrade the local license server to install a newer server release or to apply updated configuration policies provided by your producer.

You can upgrade the local license server while retaining the older license server data and configuration.

Refer to the following sections for details:

- Upgrading the Local License Server on Windows
- Upgrading the Local License Server on Linux

Upgrading the Local License Server on Windows

Steps to upgrade the local license server differ slightly, depending on your chosen installer:

- Upgrading Using the Command-Based Installer
- Upgrading Using the InstallAnywhere Installer

Upgrading Using the Command-Based Installer

Task ***To upgrade the local license server on Windows using the command-based installer***

1. As a precautionary measure, create a backup. Should errors occur during the update, you can restore the previous version from the backup.

   Back up the following files:

   - Current license server's installation directory
   - Trusted storage files (the `.ks`, `.db`, and `.0` files). The default storage location is `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`*.
   - Current license server's configuration file (`producer-settings.xml`, in the `bin\tools` directory)

     For more information, see Show General Configuration Information for the License Server.

2. Uninstall the local license server that is currently installed.

   For more information, see Uninstalling the Local License Server.

   Important ▪ *Do not delete the trusted storage files. This will result in the loss of license server data and configurations.*

3. Install the latest version of the local license server.

   For more information, see Configuring, Installing, and Starting the Local License Server.

4. Optional: If you want to continue to use the previous configuration, copy the `producer-settings.xml` file from the backup taken in Step 1 to the new installation directory.

If you are changing the trusted storage file location, you will lose the license server data unless you copy the trusted storage files (from the backup taken in Step 1) to the new location.

Upgrading Using the InstallAnywhere Installer

Task ***To upgrade the local license server on Windows using the InstallAnywhere installer***

1. As a precautionary measure, create a backup. Should errors occur during the update, you can restore the previous version from the backup.

   Back up the following files:

   - Current license server's installation directory
   - Trusted storage files (the `.ks`, `.db`, and `.0` files). The default storage location is `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`*.
   - Current license server's configuration file (`producer-settings.xml`, in the "bin\tools" directory)

     For more information, see Show General Configuration Information for the License Server.

2. Uninstall the local license server that is currently installed.

For more information, see Uninstalling the Local License Server.

Important ▪ *Do not delete the trusted storage files. This will result in the loss of license server data and configurations.*

3. Delete the following directories created during the previous installation:
   - In `C:\ProgramData`:
     - `flexnetls`
     - `flexnetsas`
   - In `C:\Windows\ServiceProfiles\NetworkService`:
     - `flexnetls`
4. Navigate to the `server` directory within the previous license server installation.
5. Delete all files in this directory except `producer-settings.xml`.
6. Place the new executable, `FlexnetLicenseServerWizardInstaller.exe`, into the `server` directory.
7. Run the executable with Administrator privileges.

If you are changing the trusted storage file location, you will lose the license server data unless you copy the trusted storage files (from the backup taken in Step 1) to the new location.

Upgrading the Local License Server on Linux

Task ***To upgrade the local license server on Linux***

1. As a precautionary measure, create a backup. Should errors occur during the update, you can restore the previous version from the backup.

   Back up the following files:
   - Current license server's installation directory
   - Trusted storage files (the `.ks`, `.db`, and `.0` files). The default storage location is `/var/opts/flexnetls/`*`producer_name`*`.`
   - Current license server's configuration file (`producer-settings.xml`, in the "bin\tools" directory)

     For more information, see Show General Configuration Information for the License Server.
2. Uninstall the local license server that is currently installed.

   For more information, see Uninstalling the Local License Server.

   Important ▪ *Do not delete the trusted storage files. This will result in the loss of license server data and configurations.*
3. Install the latest version of the local license server.

For more information, see Configuring, Installing, and Starting the Local License Server.

Important ▪ *Customers who require VMUUID detection for SE-Linux enabled platforms should also perform the following steps:*

Stop the service:

```
systemctl stop fnlicense-daemon.socket
```

Remove the socket file from `/etc/systemd/system/`:

```
rm /etc/systemd/system/fnlicense-daemon.socket
```

4. Optional: If you want to continue to use the previous configuration, copy the `producer-settings.xml` file from the backup taken in Step 1 to the new installation directory.

If you are changing the trusted storage file location, you will lose the license server data unless you copy the trusted storage files (from the backup taken in Step 1) to the new location.

Important Information When Upgrading from Release 2020.12 to the Latest Version

The local license server upgrade fails when attempting to upgrade from release 2020.12 to 2025.01. To ensure a successful upgrade, follow the appropriate migration path based on your operating system.

Migration Path for local license server builds (2012.07 and earlier)

Windows:

1. Upgrade from 2020.12 to 2021.07.
2. Then upgrade from 2021.07 to the latest version (2025.07 or later).

Linux:

1. Back up the `local-configuration.yaml` file from `/opt/flexnetls/fnedemo`.
2. Upgrade to 2021.07.
3. Then choose one of the following options:
   - Option 1:
     a. Delete `local-configuration.yaml` from `/opt/flexnetls/fnedemo`.
     b. Install the latest version (2025.07 or later)
     c. Copy the previously backed-up version of `local-configuration.yaml` to `/opt/flexnetls/fnedemo`.
   - Option 2:
     a. Install the latest version (2025.07 or later)
     b. Stop the license server service.
     c. Replace `local-configuration.yaml` with the previously backed-up version.

d. Run `systemctl daemon-reload` to update the license server service.

Uninstalling the Local License Server

Use the appropriate procedure to uninstall the FlexNet Embedded local license server service:

- Uninstall on Windows
- Uninstall on Linux

Uninstall on Windows

Use these steps to uninstall the license server service on Windows.

Important Note for Uninstalling the License Server

If you installed the license server using the command-based installation, you must follow the instructions in Uninstalling a Command-Based License Server to uninstall it.

If you installed the license server using the InstallAnywhere installer, you must follow the instructions in Uninstalling Using the InstallAnywhere Uninstaller to uninstall it.

Uninstalling a Command-Based License Server

This section describes the steps you need to perform if the license server was installed using the command-based installer.

Task ***To uninstall the license server service on Windows***

1. As an administrator, open a command prompt and navigate to the license server's installation directory.
2. Execute the command `flexnetls.bat -stop` to stop the service.
3. Execute the command `flexnetls.bat -uninstall` to uninstall the license server service. (If you attempt to uninstall the service before stopping it, a message appears indicating that you need to stop the service first.)
4. To ensure there are no hanging instances or services, execute `sc delete FNLS-producer_name` as an administrator.

   Either the command will fail with the message `The specified service does not exist as an installed service`, or it will succeed with the message `[SC] DeleteService SUCCESS`. Both outcomes indicate that the service is no longer present.
5. Delete the license server component files from the installation folder.
6. Optionally, delete these files (listed here with their default locations):
   - The trusted storage files in `C:\Windows\ServiceProfiles\NetworkService\flexnetls\producer_name` (the `.ks`, `.db`, and `.0` files)
   - The log files in `C:\Windows\ServiceProfiles\NetworkService\flexnetls\producer_name\logs`

Important ▪ *Deleting the trusted storage files erases all the licenses and their usage data along with configuration information. This can be safely done in your test environment where there is no impact on production data. In some cases, Revenera support may ask you to delete trusted storage files to resolve an issue. For general upgrades, we do not recommend deleting the trusted storage files.*

Trusted storage and log locations are defined by the license server policies `server.trustedStorageDir` and `logging.directory`, respectively, the defaults for which are based on `${bases.dir}`. Depending on the values set for these policies on your server, your trusted storage and log files might be in locations different from those mentioned in this step. See Reference: License Server Policy Settings.

Uninstalling Using the InstallAnywhere Uninstaller

If you installed the license server using the InstallAnywhere installer, the installer will have placed an uninstaller called `FlexNet Embedded Local License ServerUninstaller.exe` in the installation directory. The default location is `C:\ProgramData\flexnetls\<producer_name>\_FlexNet Embedded Local License Server_installation`.

Task ***To uninstall the license server service on Windows using the InstallAnywhere uninstaller***

1. Stop the service: In the Windows Services window (`services.msc`), locate the service **FlexNet License Server-*producer_name***. Right-click the service and click **Stop**.
2. Double-click the uninstaller, `FlexNet Embedded Local License ServerUninstaller.exe`. InstallAnywhere uninstalls the license server.
3. Delete the license server component files from the installation folder.
4. Optionally, delete these files (listed here with their default locations):
   - The trusted storage files in `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`* (the `.ks`, `.db`, and `.0` files)
   - The log files in `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`*`\logs`

   Important ▪ *Deleting the trusted storage files erases all the licenses and their usage data along with configuration information. This can be safely done in your test environment where there is no impact on production data. In some cases, Revenera support may ask you to delete trusted storage files to resolve an issue. For general upgrades, we do not recommend deleting the trusted storage files.*

   Trusted storage and log locations are defined by the license server policies `server.trustedStorageDir` and `logging.directory`, respectively, the defaults for which are based on `${bases.dir}`. Depending on the values set for these policies on your server, your trusted storage and log files might be in locations different from those mentioned in this step. See Reference: License Server Policy Settings.

Uninstall on Linux

Use these steps to uninstall the license server service on Linux.

Task ***To uninstall the license server on Linux***

1. Run the following commands in this order:

   g. `sudo systemctl stop flexnetls-producer` (to shut down the service)

   h. `sudo systemctl disable flexnetls-producer` (to make the service ineligible to start on system reboot)

   i. `sudo rm /etc/systemd/system/flexnetls-producer.service`

   j. `sudo rm -r /etc/systemd/system/flexnetls-producer.service.d`

2. Optionally, remove these files (listed here with their default locations):

   - The trusted storage files in `/var/opts/flexnetls/producer_name` (the `.ks`, `.db`, and `.0` files)
   - The log files in `/var/opts/flexnetls/producer_name/logs`

   Important ▪ *Deleting the trusted storage files erases all the licenses and their usage data along with configuration information. This can be safely done in your test environment where there is no impact on production data. In some cases, Revenera support may ask you to delete trusted storage files to resolve an issue. For general upgrades, we do not recommend deleting the trusted storage files.*

   Trusted storage and log locations are defined by the license server policies `server.trustedStorageDir` and `logging.directory`, respectively, the defaults for which are based on `${bases.dir}`. Depending on the values set for these policies on your server, your trusted storage and log files might be in locations different from those mentioned in this step. See Reference: License Server Policy Settings.

Understanding Hostids

FlexNet Embedded uses system identifiers, called *hostids*, for identification. When the FlexNet Embedded local license server communicates with the back office, FlexNet Operations, to obtain its pool of licenses, the back office uses the server's hostid to identify that particular server. Similarly, when the client requests a license from the license server, the client includes a hostid in the request to which the license server binds the licenses sent in the response.

Hostid Types

FlexNet Embedded supports the following standard hostid types (also sometimes referred to as *host types*). If no hostid is specified, the license server looks up the hostids in the native DLL component of the local license server, in the order in which they are listed in the following table:

Table 2-10 ▪ License server hostid types

Hostid	Description
PUBLISHER_DEFINED	In addition to the standard hostid types (`ETHERNET`, `FLEXID9`, `FLEXID10`, and `VM_UUID`), the software producer can optionally specify a producer-defined hostid value to identify the server to the back office. This hostid is defined in `producer_settings.xml`.
ETHERNET	The unique identifier assigned to each network interface controller (NIC).
FLEXID9	The hostid of the Aladdin dongle attached to the server.
FLEXID10	The hostid of the Wibu-Systems dongle attached to the server.
VM_UUID	Available if virtual machine detection is enabled.

Specifying a Server Hostid

If the license server is run with back-office support, the producer must specify a hostid in the back office during license server creation. The workflow for specifying a server hostid depends on whether a local license server or a CLS instance is used. In implementations with a local license server, you as the license server administrator are required to identify the hostid and communicate it to the producer. When using a CLS instance, the server hostid is based on the server instance ID (generated when the instance is created). During synchronization from the back office (which occurs based on synchronization policies or on-demand), the back office sends the hostid along with other information to the license server.

Instead of waiting for the synchronization to occur, you can also specify the license server's hostid manually. It is vital that the hostid that you set on the local license server matches the hostid that is specified in the back office. If you change the hostid on the license server to a value that is different to that specified in the back office, any licenses already mapped to the license server that is locked to the old hostid in the back office will be orphaned.

Changing a Hostid

If you need to change the local license server's hostid when it already has licenses mapped in the back office, it is best practice to return the license server in the back office. During the return operation, you can transfer the licenses to a different device; this can be the same machine with the desired hostid. After the transfer, wait for the license server to synchronize with the back office.

You can perform the steps above through the End-User Portal (see Returning a Device in the FlexNet Operations End-User Portal Help Library, available on the Revenera Product Documentation site).

Important Note When Changing a Hostid

You must delete the existing trusted storage before activating the licenses on the new license server. If the license server's hostid is changed, the used counts and served device information tracked in the previous hostid are no longer available in the local license server.

Selecting and Specifying a Local License Server Hostid

This section describes the available hostid selection options and points you to the related configuration settings and administrative APIs.

- **Use the default hostid lookup behavior.** The license server looks up a hostid in the native DLL component, using the hostid types in the order listed in Hostid Types.
- **Control the lookup order of hostid types.** The producer can set the order in which a local license server looks up the hostid. See Specifying the Order of Hostid Types.
- **Automatically select the built-in Ethernet address.** Configure the local license server to use the built-in Ethernet address as the hostid. See Automatically Selecting the Built-In Ethernet Address.
- **Manually specify a hostid.** Allow the license server administrator to explicitly set the server hostid (value/type pair). See Manually Specifying a Hostid.
- **Enable resilient hostid handling for environments where identifiers can change.** In environments prone to changing hostids, the producer can enable resilient hostid handling. See Resilient Hostid Handling.

Specifying the Order of Hostid Types

The producer can specify the order in which the local license server picks the hostid type, using the property `server.hostType.order` in `producer-settings.xml`. For more information about this property, see Reference: License Server Policy Settings, entry server.hostType.order.

Automatically Selecting the Built-In Ethernet Address

The producer can configure the local license server to pick the built-in Ethernet address as hostid, using the property `licensing.enableBuiltinHostId` in `producer-settings.xml`. For more information about this property, see Reference: License Server Policy Settings, entry licensing.enableBuiltinHostId.

Limitations

In the following scenarios, setting the built-in Ethernet address as hostid using `licensing.enableBuiltinHostId` is likely to fail:

- If a user's machine is set to use a randomized wireless MAC address.
- If the option to automatically disable the wireless network connection when an Ethernet cable is connected is enabled.

Therefore, if the Ethernet address is to be used as hostid, the relevant configurations must be disabled on the user's machine.

Deleting the Selected Built-in Hostid

If `licensing.enableBuiltinHostId` is enabled (`true`), the first built-in Ethernet address is selected as hostid. However, in the scenarios mentioned under "Limitations", above, the built-in Ethernet address might change, because the order of hostids is dynamic and changes depending on new built-in hostids becoming available or being removed.

To retain a stable built-in hostid across multiple launches of the license server, the server stores the selected built-in hostid in its database and fetches this hostid for each launch, instead of getting it from the dynamic hostid list.

If, at a later point in time, you no longer want to use the built-in hostid stored in the database, you can call the new REST endpoint **/hostids/selectedbuiltin** and delete the selected built-in hostid from the database. This forces the license server to fetch the first built-in hostid from the dynamic hostid list, instead of retrieving it from its database.

Task ***To delete the selected built-in hostid***

Call the JSON REST endpoint `http://licenseServerHostName/api/1.0/hostids/selectedbuiltin` using `DELETE`.

For a successful `DELETE`, the status code 200 is returned, but no response is provided.

Manually Specifying a Hostid

You can specify the server hostid (a value/type pair) in one of the following ways:

- Using the FlexNet License Server Administrator using the following option:

  ```
  -hostid -setactive hostIdValue hostIdType
  ```

  See Table 3-3 in Using the FlexNet License Server Administrator Command-line Tool.

- Using the FlexNet License Server Manager—see "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

- Using the configuration files (depending on your platform):

  - `flexnetls.settings`, setting `ACTIVE_HOSTID` (Windows)—see Table 2-3 in Configure, Install, and Start on Windows.

  - `configuration-local.yaml`, setting `active-hostid` (when using systemd on Linux)—see Edit "local-configuration.yaml" (Linux).

  Important ▪ *You can only set the hostid using a configuration file if no hostid has yet been specified for the license server. Once a hostid has been set, it can only be changed using the FlexNet License Server Administrator or the FlexNet License Server Manager.*

Important ▪ *If a server hostid has been set manually, the `server.hostType.order` and `licensing.enableBuiltinHostId` properties (if set) are ignored.*

Resilient Hostid Handling

Hostid handling in modern computing environments is inherently fragile due to its reliance on system identifiers that can change over time, such as network interface attributes and other hardware or OS-derived values. In both physical and virtual environments, systems are no longer static.

Routine and legitimate activities can modify these identifiers without altering the actual identity of the machine. When such changes occur, the license server may no longer recognize the system as the originally licensed device. This results in hostid mismatches, leading to unexpected license checkout failures, licensing service disruption, and the need for manual intervention to restore licensing continuity.

This section describes scenarios in which the local license server hostid can change and explains how resilient hostid handling addresses these situations.

Handling Scenarios of Changing Hostids

The following use cases highlight common situations that can lead to licensing disruption and describe why resilient hostid handling may be needed. In all use cases, the target is for licensing to remains stable across network changes so valid customers do not encounter unexpected license failures or require producer operator or support engineer intervention.

Table 2-11 ▪ Legitimate scenarios of hostid changes

Use Case	Scenario	Impact
Removable or changing network adapters	A customer provisions and activates volume license counts on a local license server device. Activation succeeds, and the server is bound to the fingerprints captured when connected to a corporate network. The user later switches to a different network (for example, home Wi-Fi or a different docking station).	The change in network adapter results in a different Ethernet identifier being used for hostid calculation. The back office treats the same machine as a new device, causing the license server not to allow checkout without any user or product-initiated change. In most cases, this situation is flagged as suspected cloning during an otherwise legitimate change or maintenance event.
OS-Level identifier randomization (privacy and security)	Customers enable operating system privacy or security features that randomize hardware or network identifiers as part of standard OS configuration.	Randomized identifiers can lead to a new hostid being generated, breaking the association with existing licenses. This behavior can appear unpredictable to customers and is difficult to diagnose.
Legitimate hardware or virtualization changes	Customers perform routine system operations such as hardware upgrades, system restores, virtual machine snapshots, migrations, or reboots.	These legitimate changes can modify underlying identifiers used for hostid calculation, causing the local license server to reject the machine as unlicensed and interrupt application availability.

About Resilient Hostid Handling

Your producer can enable resilient hostid handling for your local license server to improve license stability in environments where system identifiers may change.

Resilient hostid handling is currently supported for:

- Windows: physical and virtual machines (command-based installer only)
- Linux: virtual machines only

When resilient hostid handling is enabled, the local license server uses additional system information to determine its hostid. This allows the hostid to remain stable even when individual identifiers—such as Ethernet IDs—change due to legitimate system or environment changes.

Implementing Resilient Hostid Handling

To enable resilient hostid handling on the local license server, your producer must set the policy `binding.enable.resilient.hostid` in the `producer-settings.xml` file.

- `true`—Resilient hostid handling is enabled.
- `false`—Resilient hostid handling is disabled.

During the command-based installation on Windows, you are prompted whether to install the MetaDaemon service. The MetaDaemon service is required for resilient hostid handling on physical machines (see Running the Command-Based Installer).

In Windows installations, the `flexnetls-hostmetadata.exe` executable is required to collect physical machine identifiers used during hostid validation. This file must be present in the `server` directory, regardless of whether resilient hostid handling is enabled.

Note ▪ *Resilient hostid handling is supported only when the license server is installed using the command-based installer. The InstallAnywhere installer does not include the MetaDaemon service and therefore does not support this capability.*

Retrieving Server Hostids

Task ***To retrieve a server hostid, do one of the following***

- Call the JSON REST endpoint `http://licenseServerHostName/api/1.0/hostids (or /hostids/selected)`
- Execute `flexnetlsadmin.bat -server licenseServer_baseURL -config`

  (`flexnetlsadmin.bat` is located in the `enterprise` directory)

Preparing to Use the FlexNet License Server Manager

Your software producer might provide you with one or both of the following administration tools to help you maintain the server and manage license distribution in your enterprise:

- FlexNet License Server Administrator command-line tool (included with the license server)

- FlexNet License Server Manager (which uses a UI). This tool is available in two formats:
  - as a Docker image in the form of a .zip file (Windows and Linux)
  - as an executable (.exe) (Windows only)

Setting up and using the License Server Manager is described in a separate manual, the FlexNet License Server Manager Guide, which is available from the Revenera Product Documentation site (https://docs.revenera.com). For installation instructions, see "FlexNet License Server Manager Installation" in the FlexNet License Server Manager Guide.

Base URL for the License Server

Administrative tools, such as the FlexNet License Server Administrator command-line tool or a custom administrative tool provided by the producer, require that you input the license server's base URL to perform operations. While the same basic structure is used for the CLS instance and the local license server URLs, some syntactical differences exist. See the following sections for more information about the base URL:

- Base URL for a CLS Instance
- Base URL for a Local License Server
- With Administrative Security Enabled
- Option to Disable Certain Verifications on Server When Using HTTPS
- Exceptions to the Base URL
- "licenseServer_baseURL" as Base URL Designation

Base URL for a CLS Instance

The base URL for a CLS instance, which is provided by the producer, explicitly includes the server's instance ID. The following is the format for the base URL for a CLS instance.

```
https://siteID.compliance.flexnetoperations.com/api/1.0/instances/instanceId
```

An example might be the following (with XYZ10203040 as the instance ID):

```
flexnetlsadmin -server https://ABC.compliance.flexnetoperations.com/api/1.0/instances/
    XYZ10203040 ...
```

Base URL for a Local License Server

The base URL for a local license server includes the server's host name and port number and uses the tilde "~" character in place of the server's instance ID. The license server name can be either a network name (such as `myserver.enterprise.com`) or an IP address.

```
http://licenseServerHostName:port/api/1.0/instances/~
```

An example might be the following:

```
flexnetlsadmin -server http://11.11.1.111:7070/api/1.0/instances/~ ...
```

With Administrative Security Enabled

When administrative security is enabled on the license server, the use of the HTTPS protocol on the local license server is strongly recommended to protect sensitive data being transmitted during administrative operations. (To use HTTPS, you must first enable it on your license server machine.)

The base URL for the local license server might look like the following.

```
https://11.11.1.111:1443/api/1.0/instances/~
```

The default port for a HTTPS URL is 443, unless one is supplied. The local license server defaults to 1443 to avoid privilege issues on Linux with port numbers less than 1024, but any appropriate port can be used.

To enforce the use of HTTPS when administrative security is enabled, set the `security.http.auth.enabled` license server policy to `false` in `producer-settings.xml`. See Administrative Security Policies on the License Server.

Option to Disable Certain Verifications on Server When Using HTTPS

If for some reason, you need to disable certificate checks and hostname verification on the license server when using HTTPS, you can do so using the `-noCertCheck` option in the FlexNet License Server Administrator command-line tool (`flexnetlsadmin`). See Summary of flexnetlsadmin Commands in the Using the FlexNet License Server Administrator Command-line Tool chapter.

Exceptions to the Base URL

The producer should inform you of any exceptions to the base URL.

“licenseServer_baseURL” as Base URL Designation

In this chapter (and throughout this book), the base URL in commands is represented by the variable *licenseServer_baseURL*.

Managing Administrative Security on a Local License Server or CLS Instance

FlexNet Embedded provides an administrative security facility on the license server to prevent unauthorized queries and operations on the server. When this security is enabled on your license server, anyone attempting to administer the license server will need to provide a set of authorization credentials before starting administrative operations.

The default license-server administrator account in place when you start the server gives you access to the same full range of administrative functionality that you would have without security enabled, with additional privileges to create and manage other enterprise user accounts. The accounts you create generally have limited administrative privileges, although you can create other license-server administrator accounts.

The administrative security feature is available on both a CLS instance and the local license server.

Note ▪ *A default producer account is also created when you start up the license server. This account is used to supply or delete checkout filters (these are selective license filters customizable by the producer), and change configuration values not editable by the default administrator account (for example, `lfs.syncTo.enabled`).*

The following sections provide an overview of managing administrative security on the license server:

Security Work Flow in the Enterprise

The following provides the basic work flow for managing license-server administrative security in the enterprise. This flow assumes that the producer has deployed your license server with security already enabled. It also assumes that you have access to a license-server administrator tool, such as the FlexNet License Server Administrator command-line tool, the FlexNet License Server Manager, or a custom administrator tool provided by the producer.

Table 2-12 ▪ Security Work Flow for Administering the License Server

Step	Who	Does what	How
1	Producer	Sets up the default license server administrator account and distributes both the default credentials (if needed) and the license server's base URL.	**For a CLS instance:** The default license-server administrator account is automatically created when the CLS instance is created. **For a local license server:** The default license-server administrator account generated by the producer.

Table 2-12 ▪ Security Work Flow for Administering the License Server (cont.)

Step	Who	Does what	How
2	Enterprise license-server administrator	Resets default administrator password	**For a CLS instance:** Uses **Action > Set Password** on **View Server** page in FlexNet Operations End-User Portal. Default credentials are not required.[1] **For a local license server:** Uses one of the following: ● `-authorize` command option (to authenticate default administrator credentials) with `-users -edit` options in FlexNet License Server Administrator command-line tool (`flexnetlsadmin`). Default credentials are required.[2] ● Appropriate facility in producer's custom administrator tool. Default credentials are required.[4]
3	Enterprise license-server administrator	Uses credentials to access license-server administrative functionality	Uses one of the following: ● `-authorize` command option (to authenticate administrator credentials) in FlexNet License Server Administrator command-line tool (`flexnetlsadmin`).[2] ● **Login** view in FlexNet License Server Manager (for local license server only).[3] ● Appropriate facility in producer's custom administrator tool.[4]
4	Enterprise license-server administrator	Optionally creates other enterprise user accounts to perform specific administrative tasks (and distributes credentials to those users)	Uses one of the following to create the accounts: ● `-authorize` command option (to authenticate administrator credentials) with `-users -create` options in the FlexNet License Server Administrator command-line tool (`flexnetlsadmin`).[2] ● Appropriate facility in producer's custom administrator tool.[4]

Table 2-12 ▪ Security Work Flow for Administering the License Server (cont.)

Step	Who	Does what	How
5	Enterprise user	Uses credentials to access the specific administrative functionality	Uses one of the following: ● `-authorize` command option (to authenticate user credentials) in the FlexNet License Server Administrator command-line tool (`flexnetlsadmin`).[2] ● **Login** view in FlexNet License Server Manager (for local license server only).[3] ● Appropriate facility in producer's custom administrator tool.[4]

1. To use the FlexNet Operations End-User Portal to set your license-server administrator password on a CLS instance, access the portal's help system for the **View Server** page.
2. To use the FlexNet License Server Administrator command-line tool (`flexnetlsadmin`) to reset your license-server administrator password, create and manage other user accounts, and administer the license server (as a license server administrator or enterprise user), see Using the Command-line Tool to Manage Administrative Security in the Using the FlexNet License Server Administrator Command-line Tool chapter.
3. To use the FlexNet License Server Manager to administer the license server (as a license server administrator or enterprise user), see "Providing Credentials on a Secured License Server" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).
4. To use a custom administrator tool provided by the producer, follow specific instructions from the producer.

Secured Functionality on the License Server

When administrative security is enabled on the license server, the following license server operations require credentials:

- Resetting your administrator password
- Creating and managing other enterprise user accounts
- Adding and deleting license reservations
- Setting or resetting license server policies, including administrative security policies
- Suspending and resuming the license server
- Manual initiating a binary exchange between the license server and the back office, such an online or offline activation or synchronization event

Additionally, "read" operations might also require credentials, depending on how administrative security is configured on the license server. Refer to the next section Policy Affecting "read" Security for more information.

For references to the appropriate documentation describing how you provide credentials when using a specific administrator tool, refer to the previous Security Work Flow in the Enterprise section.

Operations Exempt from Administrative Security

The following operations are always exempt from administrative security measures (that is, no credentials are ever needed to perform these operations):

- Binary exchanges between the license server and FlexNet Operations that are initiated by the license server, such capability-polling exchanges and synchronization at set intervals
- Binary exchanges between the license server and license-enabled clients

Policy Affecting "read" Security

When administrative security is enabled on the license server, it uses the `security.anonymous` license-server policy to determine whether users need to provide credentials simply to "read" information—such as current reservations, features, licenses, or status—on the license server. See Administrative Security Policies on the License Server for more information.

User Roles Defining Administrative Privileges

When administrative security is enabled, your default administrator account is assigned ROLE_ADMIN, ROLE_OFFLINE, ROLE_RESERVATIONS, ROLE_DROPCLIENT, and ROLE_READ, giving you full rights to administer the license server. Any user account that you create can be assigned one or more roles of these roles, including ROLE_ADMIN to create another administrator.

Keep in mind that any role can perform capability exchanges and synchronization operations as this functionality is exempt from security measures.

Table 2-13 ▪ User Roles Used to Define Administrative Privileges on the License Server

Role	Privileges
ROLE_READ	Privileges to perform "read" operations (for example, to query features, licenses, reservations, or server status). When no other role is assigned to a user account, ROLE_READ is assigned by default as the only role. Additionally, depending on the administration security configuration, either every account is automatically given "read" rights, or you are required to assign ROLE_READ explicitly to each account to give it "read" rights. See Policy Affecting "read" Security.
ROLE_RESERVATIONS	Privileges to add and delete reservations.
ROLE_DROPCLIENT	Privileges to delete client records on the license server.
ROLE_ADMIN	Administrator privileges to update license server policies (local license server only), create and manage other enterprise user accounts, and perform other administrative tasks, such as suspend or resume the license server.

Table 2-13 ▪ User Roles Used to Define Administrative Privileges on the License Server (cont.)

Role	Privileges
ROLE_OFFLINE	This role is useful when granting rights for handling offline endpoint activities to users who should not have full administrative privileges. Privileges to do the following: ● Perform “read” operations (equivalent to privileges for ROLE_READ). ● Upload a .bin file with licenses to the local license server for offline server updates. ● Download a confirmation file from the local license server, which enables the user to return licenses to the back office (FlexNet Operations).
ROLE_PRODUCER	Privileges given to a producer account (by convention, the account named "producer"). Required to supply or delete checkout filters (these are selective license filters customizable by the producer), and change configuration values not editable by the administrator account (for example, `lfs.syncTo.enabled`).

Administrative Security Policies on the License Server

As the license server administrator *of a local license server*, you can update the following policies that control administrative security:

- `security.enabled` to enable or disable administrative security on the license server. See Enabling Administrative Security on a Local License Server.

The following policies are in effect only when administrative security is enabled:

- `security.anonymous` to determine whether users need to provide credentials simply to “read” information—such as current reservations, features, licenses, or status—on the license server:
  - When `security.anonymous` is set to `true`, user accounts, including administrator accounts, are automatically granted “read” rights (ROLE_READ); no credentials are needed to perform “read” operations. This policy lessens your administrative burden to manage user accounts.
  - When this policy is set to `false`, a user account, including an administrator account, must be explicitly assigned ROLE_READ in order to perform “read” operations. (The exception occurs when no role is assigned to an account, in which case ROLE_READ is assigned as the only role by default.) Credentials are then required to perform any “read” operation. If an account is not authorized for ROLE_READ, no “read” access is given.
- `security.token.duration` to limit token validity. This policy sets a time limit on how long an authorization “token” is in effect before it expires, requiring a user to re-enter credentials. Note the following:
  - This policy is relevant to FlexNet License Server Manager and those custom administrator tools where credentials are entered once and then automatically applied to each subsequent operation requiring authorization. Once the token expires, the user must re-enter credentials.
  - The policy is not relevant for tools like the FlexNet License Server Administrator command-line tool, where users must manually re-enter credentials every time they perform an operation requiring authorization.

- `security.ip.whitelist` to grant to one or more secure machines (whose IP addresses you specify for this policy) unrestricted access to the license server administrative interface. This access is helpful in cases when you forget your credentials or when someone needs the convenience of accessing administrative functionality on the license server without having to provide credentials.

  Important ▪ *If you are configuring `security.ip.whitelist` and no gateway or reverse proxy server is present, consider disabling `X-Forwarded-For` (XFF) headers using the setting `disable-xforwarded-for` in `local-configuration.yaml` (see Edit "local-configuration.yaml" (Windows) and Edit "local-configuration.yaml" (Linux)).*

- `security.http.auth.enabled` to control whether HTTPS is required to access the license server to perform secured operations (that is, those requiring authorization):

  - When set to `true`, this policy allows the use of the HTTPS or HTTP protocol to perform secured operations.
  - When set to `false`, the policy enforces the use of the HTTPS protocol to perform secured operations. An error is generated when HTTP is used.

  This policy has no effect on those operations exempt from license-server security measures, as described in Operations Exempt from Administrative Security.

For more information about these policies, see the following:

- For details about the policies, refer Reference: License Server Policy Settings.
- For instructions to update the policies, see either the Using the FlexNet License Server Administrator Command-line Tool chapter or the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com), or the instructions provided by the producer.

Enabling Administrative Security on a Local License Server

If the producer deployed a local license server with administrative security disabled, you can enable it as long as the producer has provided you with administrator credentials. You can also disable security on a currently secured license server.

Task ***To enable administrative security on the local license server***

1. Obtain default administrator credentials from the producer if you do not already have them.
2. Start up the license server. When administrative security is disabled, you have full privileges to administer the license server (as though you have an account with ROLE_ADMIN, ROLE_OFFLINE, ROLE_RESERVATIONS, ROLE_DROPCLIENT, and ROLE_READ).
3. Use your license server administrator tool to change the `security.enabled` policy to `true`:
   - For the FlexNet License Server Administrator command line tool (`flexnetlsadmin`), see Managing License Server Policy Settings.
   - For the FlexNet License Server Manager, see "License Server Properties View" in the FlexNet License Server Manager Guide.
   - For the producer's custom administration tool, refer to the producer's instructions.

4. Once administrative security is enabled on the license server, reset your default administrator password, providing your default administrator credentials to authorize your access to this operation:
   - For the FlexNet License Server Administrator command-line tool, see Reset Your Administrator Password.
   - For the producer's custom administration tool, refer to the producer's instructions.

   This operation is currently not available in the FlexNet License Server Manager.
5. From this point on, provide your credentials to perform any secured operation:
   - For the FlexNet License Server Administrator command-line tool, see Use Credentials to Access Operations.
   - For the FlexNet License Server Manager, see "Providing Credentials on a Secured License Server" in the FlexNet License Server Manager Guide.
   - For the producer's custom administration tool, refer to the producer's instructions.

Task ***To disable administrative security on the local license server***

Using your credentials, access your administrator tool to change the `security.enabled` policy to `false`.

Credential Requirements

The following lists the requirements for the user name and password needed for each enterprise user account you create.

User Name

The user name (up to 64 characters) is case-sensitive but requires no special characters.

User Password

The password for a user account is case-sensitive and must meet the following criteria:

- At least 8 characters (with a maximum of 64 characters)
- At least one digit
- At least one upper-case character
- At least one special character. The following special characters are allowed:

  `@ # % * - _ = + [ ] : , . / ? ~`

  For a complete list of special characters, see Reference: Special Characters Allowed in User Password.
- No whitespace

HTTPS Recommended

When administrative security is enabled on the local license server, the use of HTTPS is strongly recommended to protect license server data being transmitted during administrative operations. (To use the HTTPS protocol, you must enable it on your license server machine.) For more information, see Base URL for the License Server.

Optionally, you can use the `security.http.auth.enabled` license server policy to enforce the use of HTTPS when administrative security is enabled. See Administrative Security Policies on the License Server for details.

Troubleshooting

Under high client concurrency, a local license server may experience delays or become unresponsive when large numbers of clients reconnect and continuously send heartbeat requests. This can occur when incomplete client connections remain open and consume server resources.

To help improve server stability and aid troubleshooting, the local license server provides the following configuration options:

- **Socket Read Timeout**—Automatically closes incomplete client connections.
- **Client IP Logging**—Enables tracking of incoming client IP addresses in the flexnetls.log file.

Socket Read Timeout

The socket read timeout feature automatically closes incomplete client connections that do not complete communication within a specified time period.

Table 2-14 ▪

Parameter	Description
readTimeoutEnabled	Enables or disables socket read timeout processing. Default: `false`
readTimeout	Specifies the timeout value, in milliseconds. Default: 15000 ms (15 seconds) when activated Configurable Range: 1,000 ms to 60,000 ms.

Client IP Logging

Client IP logging records the IP address of each connecting client in the `flexnetls.log` file. This information can be useful when troubleshooting connectivity issues or analyzing client traffic patterns.

Table 2-15 ▪

Parameter	Description
logClientIpEnabled	Enables or disables client IP address logging. Default: `true`

Configuration Guidelines

On Linux, the `logClientIpEnabled`, `readTimeoutEnabled`, and `readTimeout` parameters can be configured using either:

- Command-line options during installation
- The `local-configuration.yaml` file.

On Windows, these parameters can be configured only via `local-configuration.yaml`.

Configuration Combinations

- **Default Installation (read timeout disabled, client IP logging enabled).**
  - **Command line**:

    ```
    ./install-systemd.sh
    ```

  - **local-configuration.yaml**:

    ```
    readTimeoutEnabled: false
    logClientIpEnabled: true
    ```

- **Enable read timeout with default value (15 seconds), with client IP logging enabled (default):**
  - **Command line**:

    ```
    ./install-systemd.sh --enable-read-timeout
    ```

  - **local-configuration.yaml**:

    ```
    readTimeout: 15000
    readTimeoutEnabled: true
    logClientIpEnabled: true
    ```

- **Enable read timeout with a custom value (for example, 5 seconds), with client IP logging enabled (default):**
  - **Command line**:

    ```
    ./install-systemd.sh --enable-read-timeout --read-timeout 5000
    ```

  - **local-configuration.yaml**:

    ```
    readTimeout: 5000
    readTimeoutEnabled: true
    logClientIpEnabled: true
    ```

- **Disable client IP logging (read timeout disabled by default):**
  - **Command line**:

    ```
    ./install-systemd.sh --log-client-ip-enabled false
    ```

  - **local-configuration.yaml**:

    ```
    readTimeoutEnabled: false
    logClientIpEnabled: false
    ```

- **Enable custom read timeout and disable client IP logging:**
  - **Command line**:

    ```
    ./install-systemd.sh --enable-read-timeout --read-timeout 5000 --log-client-ip-enabled false
    ```

- **local-configuration.yaml**:

```
readTimeout: 15000
readTimeoutEnabled: true
logClientIpEnabled: false
```

Next Steps

Once the license server is installed, configured, and running, the next steps include:

- **Step 1: Provisioning licenses on the license server.** A purchased pool of product licenses needs to be activated on the license server so that it can then distribute these licenses as needed to client devices running the product.

  If license rights are mapped to the license server in the back office, these are automatically activated on the license server when the server starts or when capability polling occurs. However, you can also activate licenses using rights IDs obtained from the producer. To send capability requests containing specific rights IDs, follow the instructions provided by the producer, or do one of the following:

  - If you are using the FlexNet License Server Administrator command-line tool, see Activating License Rights on the Server in the chapter Using the FlexNet License Server Administrator Command-line Tool.
  - If you are using the FlexNet License Server Manager, refer to “Offline Server Updates View” in the FlexNet License Server Manager Guide to perform an offline activation.

  Important ▪ *If the license server has not been provisioned with any features, it responds to client requests with a 503 error with the error message "Server instance <instanceID> is not ready". This is expected behavior because the server is not ready in this state.*

- **Step 2: Administering the license server in general.** Depending on the administrator tool you are using, see the appropriate chapter or manual, respectively, for instructions on performing administrative tasks:

  - Using the FlexNet License Server Administrator Command-line Tool.
  - FlexNet License Server Manager Guide (available from the Revenera Product Documentation site, https://docs.revenera.com).

3

Using the FlexNet License Server Administrator Command-line Tool

This chapter describes how to use the FlexNet License Server Administrator command-line tool to manage the license server and its license distribution. The chapter includes following sections:

Before You Start

The FlexNet License Server Administrator command-line tool is run from the command line, using the `flexnetlsadmin` script or batch file (in Windows, `flexnetlsadmin.bat`). The general syntax for executing `flexnetlsadmin` commands is the following:

```
flexnetlsadmin -server licenseServer_baseURL [-authorize accountName accountPassword]
    -command_option [command_option_arguments] [-command_option...]
```

The general syntax for `flexnetlsadmin` commands uses the license server's base URL, indicated by *`licenseServer_baseURL`* in the command. For more information about this URL, see Base URL for the License Server in Getting Started.

A reference of all available `flexnetlsadmin` commands is available at the end of this chapter (see Summary of flexnetlsadmin Commands). You can also run `flexnetlsadmin` with the `-help` option to display brief descriptions of all the commands:

```
flexnetlsadmin -help
```

The following sections describe additional information you should know before using this tool:

- License Server URL Designation
- About the Example Commands

License Server URL Designation

The license server's base URL, as described in Base URL for the License Server in the Getting Started chapter, is required for each `flexnetlsadmin` command. To identify this URL in the `flexnetlsadmin` commands, do one the following:

- Use the "-server" Option
- Use an Environment Variable

Use the "-server" Option

To specify the license server URL explicitly in each `flexnetlsadmin` command, use the `-server` option. Insert it as the first option in the command.

The following excerpt from a sample command shows the `-server` option designating a base URL for a CLS instance:

```
flexnetlsadmin -server https://ABC.compliance.flexnetoperations.com/api/1.0/instances/
      XYZ10203040 ...
```

This excerpt from a sample command shows the `-server` option designating a base URL command for a local license server:

```
flexnetlsadmin -server http://11.11.1.111:7070/api/1.0/instances/~ ...
```

Use an Environment Variable

As an alternative to explicitly providing the license server's base URL every time you enter a `flexnetlsadmin` command, you can set the base URL as the environment variable `FLEXNETLS_BASEURL`. The following shows an example of the variable (set in this case for a local license server):

```
FLEXNETLS_BASEURL=http://11.11.1.111:7070/api/1.0/instances/~
```

See your Windows or Linux documentation for instructions on how to set this variable.

About the Example Commands

For sake of consistency, the example `flexnetlsadmin` commands used in this chapter follow these conventions:

- For simplicity, the commands use the Linux format of the `flexnetlsadmin` command (for example, they use `flexnetlsadmin`, not `flexnetlsadmin.bat`).
- The `-server` option used to identify the license server's base URL is included in each command (although you can set it as an environment variable, as described in License Server URL Designation, to avoid having to include it each time you issue a command). The example commands use the designation *licenseServer_baseURL* for the base URL, as described in Base URL for the License Server in Getting Started.

You will need to modify the example commands as needed for your environment.

Using the Command-line Tool to Manage Administrative Security

When administrative security is enabled on your license server, anyone attempting to administer the license server will need to provide a set of authorization credentials before starting administrative operations. When the license server is started, your default administrator account is assigned ROLE_ADMIN, ROLE_OFFLINE, ROLE_RESERVATIONS, ROLE_DROPCLIENT, and ROLE_READ. These roles give you a full range of privileges to administer the license server, including authorization to create and manage other enterprise user accounts that have limited administrative privileges. You can also create other license-server administrator accounts.

For an overview of the administrative security facility, including a description of the available roles, see Managing Administrative Security on a Local License Server or CLS Instance in the Getting Started chapter.

The following sections describe how to use the FlexNet License Server Administrator command-line tool to manage administrative security on the license server:

- Base URL for the Secured License Server
- Reset Your Administrator Password
- Determine Administrative Security Policies
- Create and Manage Other Enterprise User Accounts
- Use Credentials to Access Operations

Base URL for the Secured License Server

When administrative security is enabled on the license server, the use of the HTTPS protocol is strongly recommended to protect license server data being transmitted during administrative operations. For more information about the use of this protocol in the base URL for the license server (represented by *licenseServer_baseURL* in the example commands), see Base URL for the License Server.

Reset Your Administrator Password

The producer will provide the default credentials (for the default license-server administrator account) needed to authorize your use of administrative functionality when security is enabled on the license server. Once your license server starts up, your first task is to reset the default administrator password. Use the `-authorize` options (to provide your default credentials that authorize you for the task) with the `-users -edit` options (to change your password).

Task ***To reset your administrator password***

1. Enter a command similar to this:

   ```
   flexnetlsadmin -server licenseServer_baseURL -authorize defaultAdminName {defaultAdminPassword|
       -passwordConsoleInput} -users -edit admin {newAdminPassword!|-passwordConsoleInput}
   ```

   The command requires the following:

   - *defaultAdminName*—The case-sensitive name provided by the producer for the default administrator account. (“admin” is typically the default.)

     defaultAdminPassword—The case-sensitive password provided by the producer for the default administrator account. You can provide this password directly in the command; or, to avoid exposing the password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

     Note ▪ *On Linux platforms, if you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

   - *newAdminPassword!*—The new case-sensitive password that you want to define for the default administrator account. To avoid exposing the password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. This password must meet the criteria specified in Credential Requirements.

     Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

2. Once this password is reset, use your updated credentials with the `-authorize` option to access secured administrative functionality on the license server. See Use Credentials to Access Operations.

If you want to change your administrator name, create a new administrator account. See Creating Another Administrator Account in the next section, Create and Manage Other Enterprise User Accounts.

Determine Administrative Security Policies

For a local license server, you can set policies that configure administrative security. For more information about the security policies, see Administrative Security Policies on the License Server in the Getting Started chapter. For instructions on updating the policies, see Override Policy Settings in this chapter.

Create and Manage Other Enterprise User Accounts

As a license server administrator, you can use your credentials to create other user accounts—either for enterprise users or for another administrator. Refer to the following sections:

- Creating an Enterprise User Account
- Creating Another Administrator Account
- Edit a User or Administrator Account
- Deleting a User or Administrator Account
- Viewing All User Accounts

Creating an Enterprise User Account

To create an enterprise user account, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-users -create` options to create the account. (Creating a license-server administrator account is described in the next section, Creating Another Administrator Account.)

Task ***To create an enterprise user account***

1. Enter a command similar to this:

   ```
   flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
       -passwordConsoleInput} -users -create userName {userPassword|-passwordConsoleInput} role(s)
   ```

   The command requires the following:

   - *yourAdminName yourAdminPassword*—Your administrator credentials.

     Note that you can provide your administrator password directly in the command; or, to avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

     Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

   - *userName*—The case-sensitive name to identify the enterprise user's account.
   - *userPassword*—The case-sensitive password that you are defining for the enterprise user's account. This password must meet the criteria specified in Credential Requirements. To avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

     Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is*

to surround the whole password with single quotes, and backslash any single quote that is part of the password.

- *role(s)*—The one or more roles determining the type of administrative privileges for the account—either ROLE_OFFLINE, ROLE_READ, ROLE_RESERVATIONS, or ROLE_DROPCLIENT. If multiple roles are specified, insert a plus sign (+) between each role, using no spaces (for example, ROLE_READ+ROLE_RESERVATIONS). If no role is specified, ROLE_READ is automatically assigned. For more information about roles, see User Roles Defining Administrative Privileges in the Getting Started chapter.

2. View the currently available user accounts to verify that this account has been properly added. See Viewing All User Accounts.
3. Distribute the credentials to the enterprise user so that the user can access the appropriate administrative operations. Advise the user that both the user name and password are case-sensitive.

Creating Another Administrator Account

To create another license-server administrator account, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-users -create` options to create the account.

Task ***To create another administrator account for the license server***

1. Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName
    {yourAdminPassword|-passwordConsoleInput} -users -create newAdminName
    {newAdminPassword|-passwordConsoleInput} ROLE_ADMIN[+additional_roles]
```

The command requires the following:

- *yourAdminName yourAdminPassword*—Your administrator credentials.

  Note that you can provide your administrator password directly in the command; or, to avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

  Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

- *newAdminName*—The case-sensitive name with which you are identifying the new license-server administrator account.
- *newAdminPassword*—The case-sensitive password that you are defining for the new license-server administrator account. This password must meet the criteria specified in Credential Requirements. To avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

- `ROLE_ADMIN[+additional_roles]`—ROLE_ADMIN and optional other roles—such as ROLE_OFFLINE, ROLE_READ, ROLE_RESERVATIONS, or ROLE_DROPCLIENT—that define this administrator account. If multiple roles are specified, insert a plus sign (+) between each role, using no spaces (for example, `ROLE_ADMIN+ROLE_READ+ROLE_RESERVATIONS`). For more information about roles, see User Roles Defining Administrative Privileges in the Getting Started chapter.

2. View the currently available user accounts to verify that this account has been properly added. See Viewing All User Accounts.
3. Provide the new administrator with the credentials needed to perform administrative operations. Advise the administrator that both the user name and password are case-sensitive.

Edit a User or Administrator Account

To change the password or the roles for an account belonging to either an enterprise user or a license server administrator, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-users -edit` options to edit the account.

Task ***To update a user account***

1. Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -users -edit accountName
    {currentPassword|newPassword!|-passwordConsoleInput} newRole(s)!
```

The command requires the following:

- *yourAdminName yourAdminPassword*—Your administrator credentials.

  Note that you can provide your administrator password directly in the command; or, to avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

  Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

- *accountName*—The case-sensitive user name for the account.
- *currentPassword|newPassword!*—The current account password; or, if you are changing the password, the new password, which must meet the criteria specified in Credential Requirements. To avoid

exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

- *`newRole(s)!`*—If you modifying the account's roles, the new role or roles defining the type of administrative privileges for the account. If multiple roles are specified, insert a plus sign (+) between each role, using no spaces (for example, `ROLE_READ+ROLE_RESERVATIONS`). For more information about roles, see User Roles Defining Administrative Privileges in the Getting Started chapter.

  If no role is specified, the accounts current role assignment remains in effect.

2. If you changed the account's role assignment, view the current user accounts to verify that the account has been assigned the new roles. See Viewing All User Accounts.
3. Distribute new credentials (if updated) to the enterprise user so that the user can access the appropriate administrative operations.

Deleting a User or Administrator Account

To delete an account belonging to an enterprise user or an administrator, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-users -delete` options to delete the account.

Task ***To delete a user account***

1. Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -users -delete accountName
```

2. The command requires the following:

   - *`yourAdminName yourAdminPassword`*—Your administrator credentials.

     Note that you can provide your administrator password directly in the command; or, to avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

     Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

   - *`accountName`*—The case-sensitive user (or administrator) name for the account.

3. View the currently available user accounts to verify that the account has been deleted. See Viewing All User Accounts.

Viewing All User Accounts

To view all user accounts currently available on the license server, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-users` option.

Task ***To view all user accounts***

Enter a command similar to this, where *yourAdminName* and *yourAdminPassword* indicate your administrator credentials:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -users
```

Note that you can provide your administrator password directly in the command; or, to avoid exposing this password as a command-line argument, you can use the `-passwordConsoleInput` option, which then prompts you to enter your password as console input.

Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

The output shows each account by user name and lists its roles.

Use Credentials to Access Operations

When administrative security is enabled on the license server, the license server administrator and enterprise users must use their credentials with the `-authorize` option to perform operations for which they are authorized. For example, if an enterprise user account is assigned ROLE_RESERVATIONS, the user would provide credentials in a command typical of the following:

```
flexnetlsadmin -server licenseServer_baseURL -authorize accountName {accountPassword|
    -passwordConsoleInput} -reservations -delete...
```

The `-authorize` command requires these credentials:

- *accountName*—The case-sensitive name for the account attempting to perform the operation.
- *accountPassword*—The case-sensitive password for the account.

  Note that users can provide their account password directly in the command; or, to avoid exposing this password as a command-line argument, they can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input.

  Note ▪ *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

Attempts to Perform Non-Authorized Operations

If a user attempts to perform an operation that requires authorization, but has not provided credentials or is not authorized to perform the operation, an “access denied” message is displayed. The user must either enter the credentials or contact you about obtaining appropriate credentials.

General Server Maintenance

To perform general maintenance on your FlexNet Embedded local license server, use the FlexNet License Server Administrator command-line tool to perform the following:

- Obtain the License Server Status
- Show Version Information for the Administrator Command-line Tool
- Show General Configuration Information for the License Server
- Suspend or Resume the License Server

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Obtain the License Server Status

To confirm that the license server is running, you can use the `-status` command to perform a status check. This check tells you whether the server is active and lists version information. The output also displays the URLs for the back-office server and the back-up license server (if your environment is configured for license server failover).

Note ▪ *The server version information is useful should you need to report any incidents to Revenera Technical Support.*

Task ***To check the license server status***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -status
```

Status information is displayed for your license server:

```
Version              : 2020.01.0
Build Version        : 262400
Server               : https://localhost:7070/api/1.0/instances/~
State                : Ready
Backup Server        : Not configured
BackOffice Server    : http://localhost:8080/request
Records Pending Sync : 42
Last Sync            : 7d 4h38m28s
```

***Note** ▪ The time format for the "Last Sync" value is nYnMnD nHnMnS, where Y=years, M=months, D=days, H=hours, M=minutes, and S=seconds. If any leading elements are missing, they are assumed to have a value of zero. If a sync has never been performed, "sync to back office pending" is displayed.*

Show Version Information for the Administrator Command-line Tool

Use the `-version` command to display the version information for the currently installed FlexNet License Server Administrator command-line tool.

***Note** ▪ Version information for the License Server Administrator command-line tool is useful should you need to report any incidents to Revenera Technical Support.*

Task** **To display version information for the License Server Administrator command-line tool

Enter a command similar to this:

```
flexnetlsadmin -version
```

Version information for the License Server Administrator command-line tool is displayed:

```
Version                 2020.01.0
Build Version           262400
```

Show General Configuration Information for the License Server

Use the `-config -filter general` syntax to display the information about the license server's configuration, including the server's IP address, host name, hostid (binding ID), and port number and the software producer's name.

Task** **To display license server configuration information

Enter command syntax similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -config -filter general
```

The output lists current configuration properties for the license server. For each property, its corresponding producer-setting name is provided in parentheses. (For information about producer settings for your license server, see Reference: License Server Policy Settings.)

```
server.uuid                          Host UUID that uniquely identifies this server instance in the
                                     back office
server.instanceId                    Instance ID that uniquely identifies this server instance in the
                                     REST API
server.tenantId                      Tenant for which this server will serve licenses
server.enterpriseId                  Enterprise to which this server belongs
server.siteId                        Location of this server
server.trustedStorageDir             Default directory for the trusted storage                    ${base.dir}
server.logConfiguration              External override file for Logback configuration
server.accessLogPattern              Name format for request log                 access_yyyy_mm_dd.request.log
server.publisherDefinedHostId.policy Custom hostid for the license server                         DISABLED
server.extendedHostId.enabled        Extended hostids for the license server                      true
server.forceTSResetAllowed           Trusted storage can be reset when unsynchronized data exists false
server.backupMaintenance.interval    Maximum amount of time that the back-up server can serve
                                     licenses in failover                                         3d
server.syncCompatibility             Enable sync compatibility when migrating from FlexNet Embedded
                                     Server App                                                   false

The properties marked with * are the editable properties
```

Suspend or Resume the License Server

Use `-status -suspend` to put the license server in a temporary suspended state and then use `-status -resume` to remove the suspension.

In the suspended state, the license server remains running but does not accept client requests. For example, you might want to suspend the license server so that it can to complete other operations, such as synchronization or capability exchanges with the back office.

When you resume the license server, it starts accepting client requests again.

Task ***To suspend the license server***

Enter a command similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -status -suspend
```

The license temporarily stops accepting client requests until you issue the `-status -resume` command (described next).

Task ***To resume the license server***

Enter command syntax similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -status -resume
```

Activating License Rights on the Server

Once the FlexNet Embedded local license server is installed and running, it needs to obtain the pool of licenses from which it then serves licenses as needed to client devices in your enterprise. To obtain this license pool, you perform an *activation* process, in which a capability request, containing one or more license activation IDs and their counts, used to specify the desired licenses, is sent to the back-office server (FlexNet Operations). The back office then fulfills the request by sending a response, which the license server, in turn, processes to activate the licenses. After the initial activation, you can perform additional activations as needed.

Note ▪ *If the software producer has preregistered your license server with the back office, licenses are automatically activated when the server is first started.*

To perform an activation process, you need to obtain your activation IDs from the software producer. Each activation ID identifies a set of licenses to which you have rights. The software producer should provide you with a description of the licenses and counts defined for a given activation ID and the number of activation ID copies you are allowed to install. The traditional activation method is to use internet access to exchange information with the back office so that the license server is updated with the licenses without manual intervention. However, the FlexNet License Server Administrator command-line tool also supports an offline activation process, in which files are exchanged between the license server and back office by another means of communication.

Refer to the appropriate section:

- Online Activation
- Offline Activation

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Important ▪ *If the license server has not been provisioned with any features, it responds to client requests with a 503 error with the error message "Server instance <instanceID> is not ready". This is expected behavior because the server is not ready in this state.*

Online Activation

Online activation uses the internet to exchange capability information between your license server and the back office to activate licenses on the server. This is the traditional method for activating licenses.

Task ***To activate license rights on the license server***

1. Run a command similar to the following:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 2
   ```

   These options are used:

   - `-id` to specify the activation ID (in this example, `act1`) containing the license rights.

- `-count` to specify the number of copies (in this case, `2`) of the activation ID to be installed. If no count is provided, a count of 1 is assumed.

  Note ▪ *The activation count must greater than 0 and not lower than the current count on the server.*

  If you want to provide multiple activation IDs, repeat the `-activate` option:

  ```
  flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 2 -activate -id act2
      -count 1
  ```

2. To view the installed licenses, see Viewing Features Installed on the License Server.

Offline Activation

Traditional activation relies on having internet access to perform a capability exchange between the license server and back office. However, for security reasons, your license server might have limited or no external network connections. The following offline activation process allows you to generate a capability request (specifying your desired activation IDs and counts) as a binary file, which you then send to the back office by an agreed-upon means. The back office returns the capability response as a binary file that you then process on the license server to activate your licenses.

Task ***To perform an offline activation***

1. Run the following command to generate the capability request as a binary file:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 2 -o request.bin
   ```

   These options are used:

   - `-id` to specify the activation ID (in this example, `act1`) containing the license rights.
   - `-count` to specify the number of copies (in this case, `2`) of the activation ID to install. If no count is provided, one copy is assumed.

     Note ▪ *The activation count must greater than 0 and not lower than the current count on the server.*

   - `-o` to define the name of the output binary file to which to save the generated request (in this example, `request.bin`).

   If you want to provide multiple activation IDs, repeat the `-activate` option (as shown in the previous procedure).

2. Upload the capabilty-request binary file to the back office, and download the capability-response binary file generated by the back office.

3. Run the following command to process the response generated to install the licenses on the license server:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -load capabilityResponse.bin
   ```

   The `-load` option is used to specify the capability response file returned by the back office (in this example, `capabilityResponse.bin`).

4. To view the installed licenses, see Viewing Features Installed on the License Server.

Returning License Rights to the Server

You might need to return license rights when you add a license server or want to move licenses from one license server to another. The return process is identical to the activation process, the only difference being that the license count is reduced instead of increased. For more information about the activation process, see Activating License Rights on the Server.

The FlexNet License Server Administrator command-line tool supports both an online and an offline return process.

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Online Return

The online return process uses the internet to exchange capability information between your license server and the back office to return licenses to the back office. This is the traditional method for returning licenses.

The following procedure assumes that licenses are available on the license server.

Task ***To return license rights to the back office***

1. Run a command similar to the following:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0
   ```

   These options are used:

   - `-id` to specify the activation ID (in this example, `act1`) containing the license rights.
   - `-count` to specify the number of copies (in this case, `0`) of the activation ID that should be available on the license server after licenses have been returned. If you specify `0`, all licenses are returned to the back office.

   If you want to provide multiple activation IDs, use one of the following commands:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0 -activate -id act2
       -count 0
   ```

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0 -id act2 -count 0
   ```

2. To view the installed licenses, see Viewing Features Installed on the License Server.

Offline Return

You perform an offline return when your license server has limited or no external network connection. The process is similar to an offline activation and involves generating a capability request (specifying your desired activation IDs and counts) as a binary file, which you then send to the back office by an agreed-upon means. You upload the binary file to the back office, which generates a capability response as a binary file, which you then process on the license server to decrease your licenses.

The following procedure assumes that licenses are available on the license server.

Task ***To perform an offline return***

1. Run the following command to generate the capability-request binary file to decrease license rights:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0 -o request.bin
   ```

   These options are used:

   - `-id` to specify the activation ID (in this example, `act1`) containing the license rights.
   - `-count` to specify the number of copies (in this case, `0`) of the activation ID that should be available on the license server after licenses have been returned. If you specify `0`, all licenses are returned to the back office.
   - `-o` to define the name of the output binary file to which to save the generated request (in this example, `request.bin`).

   If you want to provide multiple activation IDs, add the additional IDs as shown in the following command:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0 -id act2 -count 0 -o
       request.bin
   ```

2. Upload the capability-request binary file to the back office, and download the capability-response binary file generated by the back office.

3. Run the following command to process the response generated to decrease the number of licenses on the license server:

   ```
   flexnetlsadmin -server licenseServer_baseURL -activate -load capabilityResponse.bin
   ```

   The `-load` option is used to specify the capability response file returned by the back office (in this example, `capabilityResponse.bin`).

   If the license server outputs error `<IP address:port> Process unsuccessful for capabilityResponse.bin`, this is due to FlexNet Operations requiring a confirmation capability request when reducing the number of copies of a license. The corresponding configuration setting is the Skip Confirmation check box. (For more information on this check box, including its location, consult the FlexNet Operations Producer Portal online help.) If this check box is **not** selected, the license count available on the license server is decreased in the back office, but has the status “Waiting for confirmation”. To confirm the decrease, perform the following additional steps:

   a. Run the following command to generate another capability-request binary file:

      ```
      flexnetlsadmin -server licenseServer_baseURL -activate -id act1 -count 0 -o request2.bin
      ```

   b. Upload the capabilty-request binary file to the back office.

4. To view the decremented license count, see Viewing Features Installed on the License Server.

Viewing Features Installed on the License Server

Use the `-features` command to obtain details about features currently installed on the license server.

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform this operation. For more information, see Use Credentials to Access Operations.

Task ***To view feature details***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -features
```

Feature information is displayed:

```
================================================================================================
Name            Version           Count           Type          Expiration        Start Date
================================================================================================

f1               1.0               10             CONCURRENT    permanent         2020-05-03
f2               1.0               10             CONCURRENT    permanent         2020-05-03

Total number of features : 2
```

Managing Named License Pools

FlexNet Embedded License Server includes named license pools as a mechanism to allocate licenses to specific client devices or users, to help ensure that these entities have access to the features they need. This functionality is available for the local license server or a Cloud Licensing Service (CLS) instance, hosted in the Revenera cloud.

Important ▪ *You can use either reservations or named license pools. Reservations and named license pools cannot coexist alongside each other. For more information, see Named License Pools vs. Reservations.*

Note ▪ *In previous releases of the FlexNet Embedded license server, named license pools used to be referred to as "partitions".*

For information about the setup and use of named license pools on the license server, see Allocating Licenses Using Named License Pools in the More About License Server Functionality chapter.

The following sections describe how to perform tasks to manage named license pools:

- Define a Model Definition
- Upload the Model Definition

- View the Model Definition
- Delete the Model Definition
- View License Pools

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Define a Model Definition

Named license pools are defined using a *model definition*. The model definition is configured and uploaded per license server instance or Cloud Licensing Service (CLS) instance. You can use the FlexNet License Server Administrator command-line tool to view or delete named license pools and license count allocations on the license server.

A model definition usually specifies the following:

- One or more named license pools
- Rules of access that allow access to licenses
- Rules of access that deny access to licenses

The model definition must be written according to the grammar described in Model Definition Grammar and Syntax—EBNF. The following shows a sample model definition:

```
model "example" {
    partitions {
        partition "engineering" {
           f1 1.0 2
        }
    }

    on hostid("F01898AD8DD3/ETHERNET", "5E00A4F17201/ETHERNET") {
        use "engineering"
        accept
    }

    on any() {
        use "default"
        accept
    }
}
```

This model definition creates the named license pool called `engineering`. It places 2 counts of feature f1, version 1.0 in the `engineering` license pool. The remainder of the counts will be placed in the default license pool. Client requests that include the ETHERNET hostid F01898AD8DD3 or 5E00A4F17201 are allowed access to the `engineering` license pool, ensuring that they have access to these licenses. Client requests from other hostids are only allowed access to counts in the default license pool.

For more information about the setup and use of named license pools and model definitions on the license server, see Allocating Licenses Using Named License Pools in the More About License Server Functionality chapter.

The appendix Model Definition Grammar for Named License Pools, section Use Case Examples and Their Model Definitions for Named License Pools, lists use cases that help you write your own model definition.

Upload the Model Definition

To add named license pools, create a model definition that defines named license pools and rules of access. Use the `-model -load <filename>` command to upload the model definition to the license server (requires administrator authorization).

When uploading a model definition, the following rules and limitations apply:

- After a model definition is uploaded, the local license server is locked, and attempts to upload another model while the lock is active are rejected.
- If the upload fails, the model definition is not stored in the database. Instead, the server uses the previous model definition.
- Only one model definition can be active for each local license server or CLS instance. The model definition must not exceed a file size of 900 KB.
- You cannot upload a model definition with the name `reservations` or `default`, because these names are reserved.

Important ▪ *If you were previously using reservations, you must delete all reservation groups. Otherwise, uploading a model definition will fail. For more information, see Named License Pools vs. Reservations.*

Task ***To upload the active model definition***

Enter a command similar to this, using the file you created as the input file (in this example, `definition.model`):

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -model -load definition.model
```

View the Model Definition

To view the model definition that is active for the license server, you can use the `-model` command (requires administrator authorization). This command outputs the current model definition, showing the named license pools and rules of access.

Task ***To view the active model definition***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -model
```

The model definition for your license server is displayed, similar to this:

```
model "example" {
    partitions {
```

```
        partition "engineering" {
            feature "f1" 1.0 5
        }
        partition "sales" {
            feature "f1" 1.0 5
        }
    }

    on dictionary("business-unit" : "engineering") {
        use "engineering"
        accept
    }

    on dictionary("business-unit" : "sales") {
        use "sales"
        accept
    }

    on any() {
        use "default"
        accept
    }
}
```

Delete the Model Definition

Use the `-model -delete` command to delete the model definition that is currently applied to the license server (requires administrator authorization). The default model definition is applied to the server.

Task ***To delete the active model definition***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -model -delete
```

View License Pools

Use the `-partitions` command to view all license pools (named and default) for the license server. The output lists information about every feature and every feature slice in each license pool (feature slices are portions of feature counts allocated to a named license pool).

See JSON Response Details for a description of the most relevant lines in the JSON response.

Task ***To view all license pools***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -partitions
```

The following shows example output (shortened for documentation purposes):


```
{
    "partitions": [
        {
            "id": 1,
            "name": "default",
            "lastModified": 1563445226020,
            "default": true,
            "activeFeatureSlices": [
                {
                    "id": 1,
                    "feature": {
                        "id": 1,
                        "type": "CONCURRENT",
                        "featureName": "f_con_1",
                        "featureVersion": "1.0",
                        "expiry": "permanent",
                        "featureCount": 100,
                        "used": 96,
                    },
                    "slice": 97,
                    "used": 95,
                    "computedCount": 97,
                    "requested": "97",
                    "uncounted": false
                    "status": "NORMAL",
                },
                {
                    "id": 2,
                    "feature": {
                        "id": 2,
                        "type": "CONCURRENT",
                        "featureName": "f_con_2",
                        "featureVersion": "1.0",
                        "expiry": "permanent",
                        "featureCount": 80,
                        "used": 40,
                    },
                    "slice": 50,
                    "used": 26,
                    "computedCount": 50,
                    "requested": "50",
                    "uncounted": false
                    "status": "NORMAL",
                },
            ]
        },
        {
            "id": 2,
            "name": "p1",
            "lastModified": 1564133869116,
            "default": false,
            "activeFeatureSlices": [
                {
                    "id": 3,
```

```
                    "feature": {
                     "id": 1,
                        "type": "CONCURRENT",
                        "featureName": "f_con_1",
                        "featureVersion": "1.0",
                        "expiry": "permanent",
                        "featureCount": 100,
                        "used": 96,
                    },
                    "slice": 3,
                    "used": 1,
                    "computedCount": 3,
                    "requested": "3",
                    "uncounted": false
                    "status": "NORMAL",
                }
            ]
        }
    ]
}
```


JSON Response Details

The following section explains the most relevant lines in the JSON response that is output when you use the `-partitions` command to view license pools.

Table 3-1 ▪ Elements in a JSON Response

Parent Element	Child Element	Description
`partitions`	`"id": n`	The ID is assigned to the license pool by the license server after the model definition has been uploaded.
	`"name": "name"`	The license pool name. The names `default` and `reservations` are reserved and cannot be used.
	`"lastModified": yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`	Indicates when the license pool was last changed (for example, the allocated feature count or rules of access for that license pool have changed). The date is specified using the ISO 8601 date/time notation, expressed in UTC (for example, `2019-10-22T11:16:31.000Z`).
	`"default": true|false`	Indicates whether the license pool is the default license pool or not. The default license pool cannot be deleted.

Table 3-1 ▪ Elements in a JSON Response

Parent Element	Child Element	Description
activeFeatureSlices	"id": *n*	Each feature slice is assigned an ID by the license server after the model definition has been uploaded.
	"slice": *n*	Total license count for the feature that has been allocated to the license pool. Typically, this would be the count specified in the model definition, assuming that sufficient counts are available on the license server. If fewer counts are available on the license server than are specified in the model definition, or if existing usage prevents allocation into the license pool after a new model definition has been uploaded, "slice" specifies the actual number of allocated licenses.
	"used": *n*	The number of feature counts that have been served from this slice.
	"computedCount": *n*	The number of feature counts the slice should have, based on the model definition and the number of feature counts the server has been provisioned with.
	"requested": "*value*"	Displays the same value as "computedCount". For the default license pool, the value is "0".
	"status": "*value*"	The status of the feature slice indicates how the slices of a server are used. Possible values: ● OVER_ALLOCATED—The slice has more feature counts than it should have, based on the model definition and the number of counts the server has been provisioned with ("slice" > "computedCount"). ● NORMAL—The slice has the number of feature counts that it should have, based on the model definition and the number of counts the server has been provisioned with ("slice" = "computedCount"). ● UNDER_ALLOCATED—The slice has fewer feature counts than it should have, based on the model definition and the number of counts the server has been provisioned with ("slice" < "computedCount").

Managing Reservations

Important ▪ *You can use either reservations or named license pools. Reservations and named license pools cannot coexist alongside each other. For more information, see Named License Pools vs. Reservations. For general information about named license pools, see Allocating Licenses Using Named License Pools.*

The FlexNet Embedded local license server supports the use of license reservations as a way to pre-allocate licenses to client devices or users to help ensure that these entities have access to the licenses they need.

The reservations you post to the license server are ordered in a hierarchy, enabling more granularity in managing them. For example, *reservation entries* are the actual feature reservations that you define for a *reservation,* which specifies the hostid for the specific client device or user to which the feature reservations apply. One or more reservations are assigned to a *reservation group*, which represents a more global entity to which the client devices and users identified by the reservations belong. This group might identify a department, a specific job role assigned to employees within the company, a physical location, or any other entity applicable to your company. This hierarchy enables you to view, add, or delete reservations at the group level, the reservation (hostid) level, or the reservation entry (feature) level.

For more information about the setup and use of reservations on the license server, see License Reservations in the More About License Server Functionality chapter.

The following sections describe how to perform tasks to manage reservations:

- Add Reservations
- View Reservations
- Back Up Reservation Information to a File
- Delete Reservations

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Add Reservations

Important ▪ *If you were previously using named license pools, you must delete the active model definition before you can manage reservations. See Delete the Model Definition.*

To add reservations, create a file whose contents define a new reservation group, which, in turn, contains definitions for one or more *reservations* (each with a hostid identifying a client device or a user). Each reservation definition contains one or more *reservation entries*, each entry defined by a feature and its count.

For a quick reference to example reservation-file contents, see the following section Sample License Reservation File. For details about reservation hierarchy, definitions, and behavior, see License Reservations in the More About License Server Functionality chapter.)

Note ▪ *If you want to add or edit reservations in an existing group, delete and then re-create the group, using an input file whose contents include the reservation definitions initially used to create the group, plus the edits. See Delete All Reservations in a Group for instructions on deleting a group.*

Task ***To add one or more reservations***

1. Run a command similar to the following:

   ```
   flexnetlsadmin -server licenseServer_baseURL -reservations -load resv-1.json
   ```

   The option `-load` points to the text file (in this example, `resv-1.json`) containing the reservation definitions.

2. Use the instructions in Confirm Reservations Posting to see the added reservations.

If the reservations were not added, see License Reservations in the More About License Server Functionality chapter for information on the types of errors or conditions that can cause reservations to be rejected.

Sample License Reservation File

The following shows the contents of an example reservation file that defines a reservation group, called **support**, that contains reservations for two hostids, **7A7A77AAA77A** and **Joe**. The file is written in the JSON format required for defining reservations.

Table 3-2 ▪ Sample Contents of License Reservations File

```
{ "name" :  "support", "reservations" :
  [
    { "hostId": { "value" : "7A7A77AAA77A", "type" : "ETHERNET" },
       "reservationEntries": [
         { "featureName": "f1", "featureVersion": "1.0", "featureCount": 1 },
         { "featureName": "f2", "featureVersion": "1.0", "featureCount": 1 }
         ]},
    { "hostId": { "value" : "Joe", "type" : "USER" },
       "reservationEntries": [
         { "featureName": "f1", "featureVersion": "1.0", "featureCount": 1 },
         { "featureName": "f2", "featureVersion": "1.0", "featureCount": 1 }
         ]}
]}
```

Confirm Reservations Posting

Use the following procedure to confirm that the reservations you just added were posted correctly to the license server.

Task ***To confirm proper posting of reservations***

1. Use the `-reservations` option to list the names and IDs of existing reservation groups posted to the license server. See Show a Summary of Reservation Groups for more information.

   Note the ID for the group you just added.

2. To drill down to view the details about the reservation group you just added, use the `-reservations -group` *`group_id`* syntax, where *`group_id`* is the group's ID (obtained in the previous step). The details include the reservations defined in the group, and the reservation entries added to each reservation. See Show Reservation Details by Group for more information.

The output from these options shows you that the reservation group that has been successfully posted. If the posting process has encountered any errors or determined that an insufficient feature count exists for any of the reservations, the entire reservation group is rejected. See License Reservations in the More About License Server Functionality chapter for details.

View Reservations

These procedures allow you to view information about the reservations posted to your license server:

- Show a Summary of Reservation Groups
- Show Reservation Details by Group

Show a Summary of Reservation Groups

Use the `-reservations` option to provide the list of existing reservation groups posted to the license server. This list provides the group IDs needed to manage reservations at the group level.

Task ***To show a summary of existing reservation groups***

Run a command similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -reservations
```

The following shows example output. Note that the `ID` column shows the specific ID used to manage a group.

```
================================================================================
ID          Name                                                Reservations
================================================================================
194         Acct2                                               56
225         Payroll                                             101
257         support                                             234
```

Show Reservation Details by Group

Use the `-reservations -group` *`groupID`* syntax to list the details for every reservation in the specified reservation group. The details list the reservation entries belonging to each reservation and whether the individual reservation entries are enabled or disabled (see Disabled Reservations for more information).

Task ***To show reservation details for a specific reservation group***

Run a command similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -reservations -group 257
```

The `-group` value is the ID of reservation group whose details you want to examine (in this example, `257`). You can obtain the group ID by using the `-reservations` option (as described in the previous section).

The following shows example output for the reservation group 257:

```
================================================================
Reservation ID=289, hostid=ETHERNET/7A7A77AAA77A
================================================================
Entry ID       State    Name                 Version    Count
================================================================
290            ENABLED  f2                   1.0        1
289            ENABLED  f1                   1.0        1
================================================================
Reservation ID=290, hostid=USER/Joe
================================================================
Entry ID       State    Name                 Version    Count
================================================================
291            ENABLED  f2                   1.0        1
292            ENABLED  f2                   1.0        1
```

In this output, note the `Reservation ID` value, which identifies the given *reservation*. (For example, the reservation ID for hostid `7A7A77AAA77A` is `289`; for hostid Joe, it is `290`.) You can use this ID to manage the reservation entries in a given reservation.

Disabled Reservations

The “disabled” state for a reservation entry means that the entry is currently not active due to an insufficient feature count. A new reservation entry is never added with a “disabled” status. However, a reservation entry might become “disabled” when the license rights on the license server change. To change the reservation entry status back to “enabled”, you might need to delete the reservation group and re-add it once sufficient feature counts on the server are available.

Back Up Reservation Information to a File

Use the `-o` option to write current reservation information to a file in JSON format. You can back up either the summary of current reservation groups or the reservation details for specific group.

Task ***To write a summary of all current reservation groups to a file***

Use a command similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -reservations -o reservations_summary.json
```

Task ***To write reservation details for a specified reservation group to a file***

Use a command similar to the following:

```
flexnetlsadmin -server licenseServer_baseURL -reservations -group 257
      -o reservations_289.json
```

Delete Reservations

Use the following commands to delete reservations at the group or reservation (hostid) level.

Delete All Reservations in a Group

Use the `-reservations -delete -group` *`group_id`* syntax to delete an entire reservation group. This action deletes all reservations in the specified group and removes the group itself from the reservations posted on the license server.

Task ***To delete a reservation group***

1. Run a command similar to the following:

   ```
   flexnetlsadmin -server licenseServer_baseURL -reservations -delete -group 194
   ```

   The `-group` value is the ID of group whose reservation details you want to delete (in this example, `194`).

2. Use the `-reservations` option to retrieve the list of existing reservation groups to confirm that the group has been deleted (see Show a Summary of Reservation Groups).

Delete All Reservation Entries in a Given Reservation

Use the `-delete -group` *`group_id`* `-reservation` *`reservation_id`* syntax to delete an entire reservation (which comprises all reservation entries in the given reservation for a specific hostid) within a given reservation group. The *`reservation_id`* value is the ID of the specific reservation you want to delete. (You can obtain the reservation ID by using the `-reservations` option, as described in the previous section Show Reservation Details by Group.)

Task ***To delete an entire reservation***

1. Run a command similar to the following

   ```
   flexnetlsadmin -server licenseServer_baseURL -reservations -delete -group 1 -reservation 290
   ```

   These options are used:

   - `-group` to specify the ID of the group (in this example, `1`) containing the reservation you want to delete.
   - `-reservation` to specify the reservation ID (in this example, `290`) for the reservation you are deleting.

2. Use `-reservations -group` *`group_id`* to retrieve reservation details for the specified group to confirm that the reservation has been deleted (see Show Reservation Details by Group).

Managing License Server Policy Settings

The producer provides you with a set of policy settings that control your license server's operations. As license server administrator, you can review these settings and override any setting that is editable. Refer to the following sections for more information:

- Review Policy Settings
- Write Editable Policy Settings to a File
- Override Policy Settings
- Reset Policy Settings

For a description of all the settings in this file and a list of the specific settings that you are allowed to edit, see Reference: License Server Policy Settings.

If administrative security is enabled on the license server, you might need to provide authorization credentials to perform operations described in this section. For more information, see Use Credentials to Access Operations.

Review Policy Settings

The License Server Administrator command-line tool provides the following views of the license server policy settings. You can examine these views individually or generate a compilation of all views.

Settings marked with an asterisk (*) are editable. See Override Policy Settings for instructions on editing properties.

- Policy Settings for Licensing Operations
- Synchronization Settings
- Administrative Security Policies
- Log Settings
- Settings for Polling the Back Office for License Updates
- Failover Settings
- All Settings

For an example of the output showing general license-server configuration settings (using the `-config -filter general` syntax), see the previous section Show General Configuration Information for the License Server.

To override any of the editable policy settings, see Override Policy Settings.

Policy Settings for Licensing Operations

Use the `-config -filter license-policy` options to retrieve the current policy settings that control licensing operations on the license server. The settings define behavior for license-borrowing and renewal, serving licenses in a virtual environment, and validating the license server's hostid.

Task ***To display policy settings for licensing***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -config –filter license-policy
```

The following shows sample output:

```
  licensing.responseLifetime           Lifetime of a served-license response on the client         1d
  licensing.borrowInterval             Borrow interval for served licenses, in seconds            1w
  licensing.renewInterval              Renew interval, as % of the borrow interval, for features
                                       that do not specify an override                            15
  licensing.allowVirtualClients        Virtual clients are allowed to obtain licenses             true
  licensing.allowVirtualServer         License server is allowed to run on a virtual host         true
  licensing.defaultBorrowGranularity   Borrow granularity to use for clients that do not specify
                                       one                                                        SECOND
  licensing.hostIdValidationInterval   Frequency with which the license server validates that
                                       its host ID has not changed                                5m
* licensing.backup.uri                 URI of the backup server in a failover configuration
* licensing.main.uri                   URI of the main server in a failover configuration
  licensing.disableVirtualMachineCheck Server should check if it is running on a virtual host     false
* licensing.clientExpiryTimer          Interval between client expiry sessions                    2s
* licensing.borrowIntervalMax          Maximum borrow interval allowed for any client, in
                                       seconds by default                                         NOT_CONFIGURED
  licensing.dropClientEnforcedDelay    A client drop is allowed                                   0m
* licensing.security.json.enabled      Security is applied to JSON licensing requests             true

The properties marked with * are the editable properties
```

Synchronization Settings

Use the `-config -filter sync` options to display the current settings that control the following:

- Automatic synchronization of metered-usage and license-distribution data to the back office
- Whether the ability to recover data from the back office is enabled

For more information about synchronization with the back office, see Online Synchronization to the Back Office and Synchronization From the Back Office in the More About License Server Functionality chapter.

Task ***To display synchronization settings***

Enter a command similar to this:

```
flexnetlsadmin licenseServer_baseURL -config –filter sync
```

The following shows sample output:

```
  lfs.syncTo.enabled      Synchronization to the back office is enabled                                    true
* lfs.syncTo.pagesize     Maximum number of client records to include in a synchronization message to the
                         back office                                                                     50
* lfs.syncTo.repeats      Amount of time between synchronization sessions with the back office            1d
* lfs.syncTo.retryCount   Number of times to retry synchronization attempts if a synchronization session
                         with the back office fails                                                      4
* lfs.syncTo.retryRepeats Amount of time between synchronization attempts, when synchronization with the
                         back office fails                                                               5m
* lfs.syncTo.delay        At license-server startup, the amount of time the server should wait before
                         initiating a synchronization session to the back office                         2s
  lfs.syncTo.includeAll   Historical license-distribution data for concurrent features is collected and
                         sent to the back office as part of the synchronization data                     true
* lfs.syncTo.rate         Rate limiting of sync messaging to LFS                                          20

The properties marked with * are the editable properties
```

Administrative Security Policies

Use the `-config -filter security` options to display whether administrative security is enabled or disabled on the license server.

Task ***To display the security setting***

Enter a command similar to this:

```
flexnetlsadmin licenseServer_baseURL -config -filter security
```

The following shows sample output:

```
* security.enabled           Security is applied to REST endpoints                                  true
* security.token.duration    REST Security tokens expire after this interval                        1d
* security.http.auth.enabled Allow use of plain HTTP for secure REST endpoints                      true
* security.ip.whitelist      IP addresses that do not require REST Security
* security.jwt.cookies       If true, HTTP-Only secure cookies will be accepted as a token source   false
* security.anonymous         Set to permit anonymous read (GET) access                              false

The properties marked with * are the editable properties
```

Log Settings

Use the `-config -filter log` options to display the current settings that control the type of messages to log for your license server and the location of the log files.

Task ***To display log settings***

Enter a command similar to this:

```
flexnetlsadmin licenseServer_baseURL –config -filter log
```

The following shows sample output:

```
  logging.directory Directory to which the license server writes the log    C:\Users\johndoe\flexnetls\demo\logs
* logging.threshold Lowest level of log-message granularity to record,fatal, error, warn, or info       LICENSING

The properties marked with * are the editable properties
```

Settings for Polling the Back Office for License Updates

Use the `-config -filter capability` options to display the current settings that control the license server's polling the back office periodically to update license rights on the server. (The polling process involves sending a capability request to the back office to determine whether license updates exist.)

Task ***To display settings used to control the polling process***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL –config -filter capability
```

The following shows sample output:

```
  lfs.capability.enabled        Capability-request polling is enabled                                    true
* lfs.capability.repeats        Amount of time between capability-request polls                          1d
* lfs.capability.retryCount     Number of capability-polling attempts allowed, if polling fails         3
* lfs.capability.retryRepeats   Amount of time between capability-polling attempts, if polling fails    30s

The properties marked with * are the editable properties
```

Failover Settings

Use the `-config -filter failover` options to retrieve settings that you have defined on the back-up and main server when license server failover is configured.

For more information about license-server failover configuration, see License Server Failover in the More About License Server Functionality chapter.

Task ***To display failover settings***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -config –filter failover
```

The following shows sample output:

```
* fne.syncTo.enabled      To enable server to server synchronization                                    false
* fne.syncTo.mainUri      URI of the main server in a failover configuration
  fne.syncTo.repeats      Amount of time between initiating synchronization sessions with the main server   5m
* fne.syncTo.pagesize     Maximum number of client records to include in a synchronization message to the
                          backup server                                                                 100
* fne.syncTo.retryCount   When a synchronization from the main server fails, the number of times to retry
                          synchronization                                                               1
* fne.syncTo.retryRepeats Amount of time between synchronization attempts when, synchronization from the
                          main server fails                                                             1m

The properties marked with * are the editable properties
```

All Settings

Use `-config` or `-config -filter none` to view a compilation of all settings—general license-server identification, licensing, synchronization, administrative security, logging, and capability polling. The compilation also includes failover settings defined on the main or back-up license server if license server failover is configured.

Task ***To display all settings***

Enter a command similar to either of these:

```
flexnetlsadmin -server licenseServer_baseURL –config
```

or

```
flexnetlsadmin -server licenseServer_baseURL –config -filter none
```

Write Editable Policy Settings to a File

You can write the current editable policy settings for the local license server to a file in JSON format. This feature provides not only a backup of the settings for reference but also the basis for an input file you can use to apply multiple overrides to the settings (see the next section Override Policy Settings). You can write all editable settings or a specific category of editable settings to a file.

Task ***To write all editable policy settings to a file***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL –config -filter none -o allsettings.json
```

To write all editable settings, the value for the `-filter` option must be `none`; the value for the `-o` option is the name of the file to which to write the current settings (in this example, `allsettings.json`).

Task ***To write a category of editable policy settings to a file***

Enter a command using the desired `-filter` category. For example, to write policy settings related to synchronization with the back office, use the `sync` category:

```
flexnetlsadmin -server licenseServer_baseURL –config -filter capability -o capability_settings.json
```

The file contents look similar to this:

```
{"lfs.capability.repeats":"1d"}
{"lfs.capability.retryCount":"3"}
{"lfs.syncTo.retryRepeats":"30s"}
```

See Summary of flexnetlsadmin Commands for a description of the `-filter` categories.

Override Policy Settings

You can override editable policy settings for the local license server as needed for your site. The License Server Administrator command-line tool allows you to override individual settings from the command line or apply multiple overrides through an input file that defines the overrides.

For a quick reference of editable settings, run use the `flexnetlsadmin` command with `-config` to review which settings are marked with an asterisk, indicating that they are editable (see Review Policy Settings.)

Refer to Reference: License Server Policy Settings for a description of all policy settings.

Override Individual Settings

Use the following command to override individual policy settings.

Task ***To override settings directly from the command line***

1. Enter a command using this basic syntax, where *`settingName`* is name of the setting you are updating, and *`settingValue`* is its new value:

   ```
   flexnetlsadmin -server licenseServer_baseURL –config -set settingName=settingValue
   ```

   For example, to override a single policy setting (in this example, `lfs.capability.repeats`), enter a command similar to this:

   ```
   flexnetlsadmin -server licenseServer_baseURL –config -set lfs.capability.repeats=2d
   ```

   The following shows other syntax requirements:

   - To override more than one setting, enter a command similar to this, using commas to separate the settings:

     ```
     flexnetlsadmin -server licenseServer_baseURL –config -set lfs.capability.repeats=2d,
         lfs.capability.retryCount=1
     ```

   - To override a setting that contains multiple values (such as might be the case with `security.ip.whitelist`), use commas to separate values, and then enclose the entire set of values in

square brackets. Additionally, on Windows, you must also enclose the entire bracketed component in double quotes, as shown:

```
flexnetlsadmin -server licenseServer_baseURL -config -set
    security.ip.whitelist="[127.0.0.1,192.0.0.1]"
```

On Linux, you can omit the double quotes around the bracketed component:

```
flexnetlsadmin -server flexnetlsadmin -server licenseServer_baseURL -config -set
    security.ip.whitelist=[127.0.0.1,192.0.0.1]
```

- To override more than one setting when one of the settings includes multiple values (such as might be the case for `security.ip.whitelist`), enter a command similar to this (shown for Windows):

  ```
  flexnetlsadmin -server licenseServer_baseURL -config -set
      lfs.capability.repeats=2d,security.ip.whitelist="[127.0.0.1,192.0.0.1]"
  ```

  As in the previous example, on Linux you can omit the double quotes around the bracketed component.

2. If necessary, stop and restart the license server to put the overrides into effect.

From an Input File

The following procedure defines how to apply an input file to override multiple policy settings.

Task ***To update multiple producer settings using an input file***

1. Use one of the procedures described in Write Editable Policy Settings to a File to generate a JSON file containing the current editable settings for your license server.
2. Open the file in a text editor, change the values for those settings you want to override, and then save the file.

   For example, to change the `lfs.capability.retryRepeats` value from 30 seconds to 20 seconds, locate the setting in the text file and change its value:

   ```
   lfs.capability.retryRepeats=20s
   ```

3. Enter a command similar to the following, using the JSON file you updated (in this example, `settings.json`) as the input file:

   ```
   flexnetlsadmin -server licenseServer_baseURL -config -load settings.json
   ```

Reset Policy Settings

You can reset one or more policy settings back to the original default values set by the producer.

For example, suppose that the original value for `lfs.capability.repeats`, as set by the producer for your enterprise, was `1d`. At some point, you used `-config -set` to override the default with the new value `2d`. If you later wanted set the policy back to the producer's default (that is, `1d`), you would use the procedure described here.

Task ***To reset individual policy settings***

Enter a command using this syntax, where *settingName* is the name of the policy setting you are resetting:

```
flexnetlsadmin -server licenseServer_baseURL -config -reset settingName...
```

For example, to reset a single policy (in this example, `lfs.capability.repeats`), enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL –config -reset lfs.capability.repeats
```

To reset more than one setting, enter a command similar to this, using commas to separate the settings:

```
flexnetlsadmin -server licenseServer_baseURL –config -reset lfs.capability.repeats,
    lfs.capability.retryCount
```

Monitoring License Distribution to Clients

The FlexNet License Server Administrator command-line tool provides options to monitor the license server's current license distribution at a summary and detailed level:

- View a Summary of Features and Active Clients
- View License Details

View a Summary of Features and Active Clients

Use the following command to display the total of different features and active clients on the license server.

Task ***To display a summary of licenses and active clients***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -licenses
```

The following shows sample output:

```
License Server Address : 11.11.1.111:7070
Number of features     : 2
Number of clients      : 1
```

View License Details

Use the following command to display these license details:

- Each feature currently installed on the license server and its current used and available counts
- Each client device that currently has features checked out and the checked-out count for each feature
- A summary that includes the total counts of features not being used (available) and being used, along with the number of uncounted features

Task ***To display license details***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -licenses -verbose
```

The following shows sample output:

```
================================================================================
Feature Name    Version     Count    Used/Available     Expiration       Start Date
================================================================================
f1              1.0         6        4/2                permanent        2020-01-14
f2              1.0         3        2/1                permanent        2020-02-05


================================================================================
Client                              Alias               Feature / Used Counts
================================================================================

STRING/client-14                     DeviceABC           f2/2
STRING/client-1                      DeviceXYZ           f1/4

Total of all feature counts:        : 9
Number of installed features        : 2
Number of "uncounted" features      : 0
Total of used feature counts        : 6
================================================================================
```

Note ▪ *In this display, the total number of licenses per feature or for all features is the sum of the available and used counts. For example, if feature f1 shows 4 counts used and 2 counts available, the total number of f1 licenses installed on the server is 6.*

Add `-json` to display output in JSON format instead of the default table layout:

Task ***To display license details in JSON format***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -licenses -verbose -json
```

The following shows sample output:

```
[ {
    "id" : 3,
    "type" : "CONCURRENT",
    "featureName" : "f1",
    "featureVersion" : "1.0",
    "featureId" : "a704e059-671e-4378-8492-f6c4ce05cc9c",
    "featureCount" : "2",
    "overdraftCount" : 0,
    "expiry" : "permanent",
    "startDate" : "",
    "used" : 0,
```

```
    "vendorString" : "%%ROLE:ANY,REGION:EMEA%%",
    "notice" : "integration-helper",
    "featureKind" : "NORMAL_FEATURE",
    "vendor" : "fnedemo",
    "meteredUndoInterval" : 0,
    "meteredReusable" : false,
    "receivedTime" : "2020-02-05T10:36:50.000Z",
    "selectorsDictionary" : {
      "ROLE" : "ANY",
      "REGION" : "EMEA"
    },
    "concurrent" : true,
    "uncounted" : false,
    "uncappedOverdraft" : false,
    "reserved" : 0,
    "metered" : false
}, {
    "id" : 4,
    "type" : "CONCURRENT",
    "featureName" : "f2",
    "featureVersion" : "1.0",
    "featureId" : "2b92a4a8-3296-4594-8a49-8dba420cafbe",
    "featureCount" : "2",
    "overdraftCount" : 0,
    "expiry" : "permanent",
    "startDate" : "",
    "used" : 0,
    "vendorString" : "%%ROLE:ANY,REGION:EMEA%%",
    "notice" : "integration-helper",
    "featureKind" : "NORMAL_FEATURE",
    "vendor" : "fnedemo",
    "meteredUndoInterval" : 0,
    "meteredReusable" : false,
    "receivedTime" : "2020-02-05T10:36:50.000Z",
    "selectorsDictionary" : {
      "ROLE" : "ANY",
      "REGION" : "EMEA"
    },
    "concurrent" : true,
    "uncounted" : false,
    "uncappedOverdraft" : false,
    "reserved" : 0,
    "metered" : false
} ]
```


Viewing Current Binding Status

If the producer has enabled a special binding-break detection feature for the license server, you can check the server's current binding status as reported by this feature.

Task ***To view the current binding status of the license server***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -binding
```

If the producer has not enabled the binding-detection feature, the following is displayed:

```
{
   "bindingPolicy" : "disabled"
}
```

If the binding-detection feature is enabled, the results show the current policy for binding breaks on the license server and the server's current binding status, as shown in this example output:

```
Binding Policy :86400 seconds
Binding Status :soft break
```

The next sections provide more information about this output.

Binding Policy

The `Binding Policy` property shows the current policy that the producer has set for binding breaks for the license server. This policy uses one of the following values to define the action taken should the binding-break detection feature perceive a break:

- **hard**—An immediate hard binding break is imposed—that is, the license server can no longer serve licenses.
- **soft**—A soft binding break will be in effect, allowing the license server to continue to serve licenses despite the break.
- ***n* seconds**—A soft binding break with a grace period will go into effect, allowing the license server to continue to serve licenses until the grace period expires. (The *n* `seconds` value shows the length of the grace period.) When the expiration occurs, a hard binding break goes into immediate effect, and licenses can no longer be served.

Binding Status

The `Binding Status` property shows the current binding status on the license server:

- **ok**—No binding break is detected.
- **hard break**—A binding break is detected, and the license server is no longer able to serve licenses. If a grace period is defined, this status goes into effect when the grace period expires.
- **soft break**—A binding break is detected, but the license server continues to serve licenses. If a grace period is defined, the server continues to serve licenses until the grace period expires, at which time the status changes to hard.

Repair a Binding Break

To repair the binding break, contact the producer for the correct repair procedure. You might be required to send a capability request to the back office to initiate the repair process.

Summary of flexnetlsadmin Commands

The following describes the command options and associated arguments used by the FlexNet License Server Administrator command-line tool.

Usage

The following shows general command usage for the license server administrator and an enterprise user.

For a License Server Administrator

The following is command usage for a license server administrator performing general administrative tasks:

```
flexnetlsadmin -server licenseServer_baseURL [-authorize adminName {adminPassword|
    -passwordConsoleInput}] -command_option [command_option_arguments] [-command_option...]
```

The following is the command usage for a license server administrator managing user accounts when administrative security is enabled:

```
flexnetlsadmin -server licenseServer_baseURL -authorize adminName {adminPassword|
    -passwordConsoleInput} -users [-create|edit|delete]
```

For an Enterprise User

The command usage for an enterprise user is the following:

```
flexnetlsadmin -server licenseServer_baseURL [-authorize userName {userPassword|
    -passwordConsoleInput}] -command_option [command_option_arguments] [-command_option...]
```

Additional Information

For additional assistance, see the following:

- For the license server's base URL (*licenseServer_baseURL*) required for each command, see License Server URL Designation.
- If administrative security is enabled on the license server, you are required to provide authorization credentials to perform certain `flexnetlsadmin` operations. For more information about providing these credentials, see Use Credentials to Access Operations.

Command Summary

The following table summarizes `flexnetlsadmin` command options and arguments.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options

Option	Arguments	Description
-help	(no argument)	Lists the syntax of all `flexnetlsadmin` commands.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-server	(no argument)	Sets the license server base URL required in every command as the first option. For a local license server, use the following URL: `flexnetlsadmin.bat -server http://`*`licenseServerURL`*`/api/1.0/instances/`*`instanceID`* For a CLS instance, use the URL that the producer provides, similar to this: `https://`*`siteID`*`.compliance.flexnetoperations.com/api/1.0/instances/`*`instanceId`* Alternatively, set this value as the environment user variable `FLEXNETLS_BASEURL` on your machine so that you no longer have to include this option in the commands. See License Server URL Designation for more information.  ***Note ▪*** *Use of the HTTPS protocol on the local license is strongly recommended when administrative security is enabled on the license server.*

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *adminName* {*adminPassword* | -passwordConsoleInput}	The `-authorize` command option and its related options and arguments are in effect only when administrative security is enabled on the license server. Position the `-authorize` option after the `-server` option and before any other options. Administrators can choose to enter their password directly in the command or, for security reasons, use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. The remaining options and arguments for `-authorize` are described next. See Using the Command-line Tool to Manage Administrative Security for details. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*	
	`[-users]`	Shows all current user accounts on the license server.
	`[-users -create` *`userName`* `{`*`userPassword`*`| -passwordConsoleInput} [`*`roles`*`]]`	Creates a new user account: • The *`userName`* and *`userPassword`* must meet the criteria specified in Credential Requirements. • To avoid exposing the password as a command-line argument, you can use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. • Roles are optional. If no role is specified, ROLE_READ is assigned by default. Valid roles include ROLE_READ, ROLE_RESERVATIONS, ROLE_ADMIN, ROLE_OFFLINE, ROLE_DROP_CLIENT, and ROLE_PRODUCER. Multiple roles can be combined using a plus sign (+) without any spaces in between (for example, `ROLE_READ+ROLE_RESERVATIONS`). See User Roles Defining Administrative Privileges for details. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *adminName* {*adminPassword*| -passwordConsoleInput} (cont.)	[-users -edit *userName* {*password*|*newPassword*| -passwordConsoleInput} [*newRoles*]]	Updates the password or role or both for the user account identified by *userName*. ● The password is not optional. Enter either the user's current password or, if changing the password, the new password (which must meet the criteria listed in Credential Requirements). ● To avoid exposing the password as a command-line argument, you can use the -passwordConsoleInput option, which then issues a prompt to enter the password as console input. ● Assignment of new roles is optional. If one or more new roles are specified, they replace all current roles. If no roles are specified, the user's current role assignment remains in effect. For information about roles, see User Roles Defining Administrative Privileges.  ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*
	[-users -delete *userName*]	Deletes the user account specified by *userName*
	[*any options listed below*]	Authorizes your access, as needed, to operations specified by any of the options listed below.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-authorize *userName* {*userPassword*| -passwordConsoleInput}	[*specific options listed below*]	Authorizes an enterprise user's access to the specific operations to which that user has privileges. Users can choose to enter their password (as defined for their account) directly in the command or, for security reasons, use the `-passwordConsoleInput` option, which then issues a prompt to enter the password as console input. ***Note ▪*** *(Linux only) If you supply a password directly on the command line you may need to escape some characters with a backslash in order to prevent the shell from interpreting them. The safest approach is to surround the whole password with single quotes, and backslash any single quote that is part of the password.*
-status	[-suspend | -resume]	Shows information about the license server, including its status (as active, inactive, or suspended), its build and version, the back-office URL, and other information. The `-suspend` option temporarily suspends the license server (that is, the license server is still running but does not accept capability requests from client devices). The `-resume` option ends the license server's suspended state so that the server can proceed with normal operations.
-hostid	(no argument)	Displays the license server's hostid value used to fulfill capability requests against a back-office server. If the server has multiple hostid values, the list contains the available hardware Ethernet addresses and dongle IDs. If virtual hosts are supported, the VM UUID will also be listed.
	-selected	Returns the current "selected" hostid—that is, the hostid currently used by the license server.
	-setactive *hostIdValue hostIdType*	Designates a new "selected" hostid. The new hostid must be one of the hostids returned by the `-hostid` command.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-activate	`-id` *`activation-id`* `[-count` *`value`*`]`	Activates licenses on the server by installing the specified number of copies of license rights (identified by the activation ID) from the back office. You can specify the `-activate` option multiple times to activate license rights from different activation IDs.
	`-activate -id` *`activation_id`* `-count` *`value`* `-O` *`filename`*	(Part 1 of an offline activation) Generates a capability request containing the activation ID and saves the request as a binary file. To generate the request with multiple activation IDs, repeat the `-activate` option for each ID.
	`-load` *`filename`*	(Part 2 of an offline activation) Processes the offline capability-response binary file from the back office to install licenses on the license server.
-licenses	`[-verbose]`	Retrieves summary information or details (with `-verbose`) about current license distribution to client devices.
-features	(no argument)	Shows details about the features in the current license pool on the server.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-config	`[-filter none | license-policy | sync | security | log | general | capability | failover]`	Lists all license-server policy settings. Use the `-filter` argument to list settings for only specific categories. The categories include: • `none`: All categories • `license-policy`: Policies for license distribution and usage • `sync`: Policies for synchronization with back office • `security`: The policy enabling administrative security on the license server • `log`: Logging parameters • `general`: Attributes identifying the license server • `capability`: Policies for polling the back-office for license updates • `failover`: License server failover You can specify multiple categories for the `-filter` option. For example, `flexnatlsadmin -config -filter sync log` shows synchronization and logging settings. (Use `-config` or `-config -filter none` to list all settings.) Those settings that you can edit are listed with an asterisk.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-config (cont.)	`-set` *`settingName=settingValue,settingName=settingValue...`*	Overwrites the current value for the specified policy setting. You can provide multiple value pairs separated by commas. For additional syntax formats, see Override Policy Settings. ***Note ▪*** *To determine which settings are editable, use "-config" to list the settings; editable settings are marked with an asterisk. Additionally, when you use "-o" to write settings to a file, only editable settings are written. Another way to apply edits to multiple settings is to use "-filter category -o <filename>" to write the editable settings to file, edit the settings as needed, and then use "-load <filename>" to apply the edits.*
	`-reset` *`settingName,settingName...`*	Resets one or more policy settings back to their original default values set by the producer. Separate multiple setting names by commas. For additional syntax formats, see Override Policy Settings.
	`-o` *`filename`*	Writes the current, editable policy settings (except those is the `general` category) to the specified file in JSON format. This argument can be used with the `-filter none` option to write all editable settings to file or with the `-filter` *`category`* option (except for the `general` category) to limit the scope of editable settings written to the file.
	`-load` *`filename`*	Applies the edits to policy settings from the specified file to the existing policy settings. The contents of the file you are loading must be in JSON format. In a typical scenario, first run `-filter` *`category`* `-o` *`filename`* to write current editable settings to a file, edit the values as needed, and then apply the edits using the `-load` *`filename`* option.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-model	(no argument)	Displays the model definition that is currently active, showing the license pools and rules of access.
	`-load` *`filename`*	Uploads the model definition to the license server. The contents of the file you are loading must be written according to the grammar in Model Definition Grammar and Syntax—EBNF. Only one model definition can be active for each license server instance. You cannot upload a model definition with the name `reservations` or `default`, because these names are reserved.  ***Important ▪*** *If you were previously using reservations, you must delete all reservation groups. Otherwise, uploading a model definition will fail.*
	`-delete`	Deletes the model definition that is currently applied to the license server. The default model definition is automatically applied to the license server.
-partitions	(no argument)	Displays all license pools (named and default) for the license server. The output lists information about every feature and every feature slice in each license pool (feature slices are portions of feature counts allocated to a license pool).
-reservations	`[-group` *`group_id`*`] [-o` *`filename`*`]`	Provides a summary of the reservation groups, including each group's ID, currently available to the license server. The `-group` *`group_id`* option shows details—at the reservation (hostid) and reservation-entry (feature) levels—for the specified reservation group. The `-o` *`filename`* option writes the reservation information in JSON format (for all groups or for the specified group ID) to a file.
	`-genjson` *`filename`*	Generates a template file for reservations in JSON format.

Table 3-3 ▪ FlexNet License Server Administrator Command-line Tool Options (cont.)

Option	Arguments	Description
-reservations (cont.)	`-load` *`filename`*	Creates a reservation group and its reservation entries via a file containing the reservation definitions in JSON format.  ***Note ▪*** *To make changes to a reservation group, delete the group and recreate it with the changes to the reservation definitions.*  ***Important ▪*** *If you were previously using named license pools, you must delete the active model definition before creating reservations.*
	`-delete -group` *`group_id`*	Deletes the entire reservation group with all its reservations and reservation entries.
	`-delete -group` *`group_id`* `-reservation` *`reservation_id`*	Deletes the specified reservation within the specified reservation group.
-uploadPublicKey *clientPublicKeyFileName*	(no argument)	Uploads a file containing a client-side RSA public key (2048-bit DER-encoded) in octet-stream format to the license server. The producer will provide details about obtaining and uploading this key should such a key be required. License-server administrator credentials are required to use this option.
-binding	(no argument)	Shows both the current policy for a binding-break detection facility (enabled by the producer) and the current binding status as detected by this facility (`ok`, `hard break`, or `soft break`). If this facility is not enabled, the output indicates as such.
-version	(no argument)	Shows the version and build information for the FlexNet License Server Administrator command-line tool.
-json	(no argument)	Displays output in JSON format instead of the default table layout.  ***Note ▪*** *You can query the JSON output of* `flexnetlsadmin` *with an external tool like jq. For information about jq, see https://stedolan.github.io/jq/manual/v1.6/.*

4

More About License Server Functionality

This chapter provides background information about certain functionality that can be enabled for your FlexNet Embedded license server and that requires you to perform additional set-up steps. Note that the functionality described in this chapter applies mainly to the local license server, except for functionality involving license reservations, which applies to both the local license server and a CLS (Cloud Licensing Service) instance. The following topics are discussed:

General Information

The following provides some general information about using license server functionality

About "licenseServer_baseURL" Used in Commands

Some the commands referenced in this chapter use the license server's base URL, indicated by *licenseServer_baseURL* in the command. For more information about this URL, see Base URL for the License Server in the Getting Started chapter.

Authorization Credentials for Administrative Security

If administrative security is enabled for the license server, you might need to provide authorization credentials to perform certain functions described in this chapter. For more information about administrative security and setting up authorization credentials, see Managing Administrative Security on a Local License Server or CLS Instance in the Getting Started chapter.

License Borrowing

The borrow interval can be used to specify an early expiration date for a feature. In other words, the borrow interval is the maximum amount of time that a client can borrow a feature from the license server. Once the borrow interval expires, the feature is no longer available for acquisition on the client. The client must send another capability request to the license server to borrow the feature again. By setting a borrow interval, a license administrator or producer can control the use of licenses more efficiently and ensure that unused licenses are returned to the license pool so that they are available to users who need them.

A borrow interval can be set at different levels:

- **In the back office**—By default, the producer specifies the borrow interval at feature-level. This borrow interval is also referred to in this documentation as the *feature borrow interval*.

  ***Note** ▪ All current versions of FlexNet Operations require the feature borrow interval to be set.*

- **In producer-settings.xml**—The property `licensing.borrowInterval` specifies the default borrow interval. The default interval is one week (`1w`). This value is only editable by the producer. This borrow interval is also referred to in this documentation as the *server borrow interval*.

- **In a client request**—The end user can define a borrow interval that is included in capability requests that a client sends to the license server. This borrow interval is also referred to in this documentation as the *client borrow interval*.

- **In a configuration parameter**—The license administrator can specify the configuration parameter `licensing.borrowIntervalMax` to restrict the borrow period of the clients and make more efficient use of their licenses where clients are significantly shorter lived than the borrow interval set at any of the levels mentioned in the preceding bullet points. This borrow interval is also referred to in this documentation as the *admin borrow interval*.

  ***Note** ▪ The configuration parameter `licensing.borrowIntervalMax` cannot be used for metered features.*

A feature's current borrow expiration can never exceed the final expiration time for that feature. Should the borrow expiration be greater than the feature's final expiration, the borrow interval is shortened to the final expiration time (including grace period, where applicable). In addition, a borrow-interval granularity is applied to the effective borrow interval. For more information, see Borrow Granularity.

Determining the Effective Borrow Interval

The following bullet points help you determine the borrow interval that will apply to a feature:

- **If the feature borrow interval has been set in the back office**, the effective borrow interval is the lowest of the following values:
  - feature borrow interval (set in the back office)
  - client borrow interval (set in a client capability request)
  - admin borrow interval (set using the configuration parameter `licensing.borrowIntervalMax`)
- **If the feature borrow interval has not been set in the back office**, the effective borrow interval is the lowest of the following values:
  - server borrow interval (defined in `producer-settings.xml` by the property `licensing.borrowInterval`)
  - client borrow interval (set in a client capability request)
  - admin borrow interval (set using the configuration parameter `licensing.borrowIntervalMax`)

The following diagram illustrates the behavior:

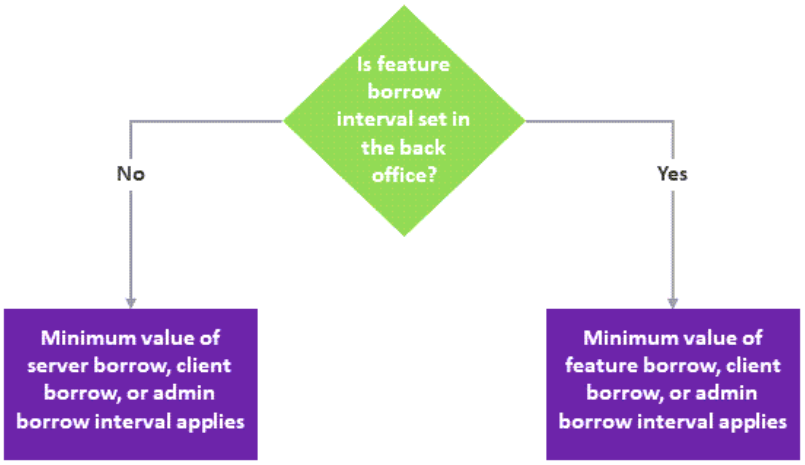


Figure 4-1: Borrow interval precedence

The expiration date that is calculated through the borrow interval can never exceed the feature expiry date (including grace period, where applicable).

Setting the Admin Borrow Interval

Users require the ROLE_ADMIN role to set the admin borrow interval. The configuration parameter `licensing.borrowIntervalMax` can be set in the following ways:

- Using the FlexNet License Server Administrator command-line tool, `flexnetlsadmin`
- Using the FlexNet License Server Manager

When setting the admin borrow interval, note the following:

- Take into account that the borrow-interval granularity (if set) can affect the effective borrow interval. For more information, see Borrow Granularity.
- Consult with your software producer to ensure that their software's license renewal mechanism can adapt to borrow intervals shortened by the `licensing.borrowIntervalMax` parameter.
- Do not set an excessively short admin borrow period which could result in the license server being overloaded by client requests.

Setting the Admin Borrow Interval Using flexnetlsadmin

To set the admin borrow interval in `flexnetlsadmin`, use the `-authorize` option (to provide your credentials that authorize you for the task) with the `-config -set` option.

Task ***To set the admin borrow interval in flexnetlsadmin***

Enter a command similar to this:

```
flexnetlsadmin -server licenseServer_baseURL -authorize yourAdminName {yourAdminPassword|
    -passwordConsoleInput} -config -set licensing.borrowIntervalMax value
```

where *value* is the borrow interval in seconds.

Setting the Admin Borrow Interval Using the FlexNet License Server Manager

You set the admin borrow interval in the FlexNet License Server Manager user interface.

Task ***To set the admin borrow interval using the FlexNet License Server Manager***

1. In the FlexNet License Server Manager, click **License Server Setup** and select **Properties**.
2. In the **License Generation** section, locate the **Borrow Interval Maximum** setting and enter the required value.

For more information about the License Server Manager, see the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

Borrow Granularity

A borrow-interval granularity is applied to the effective borrow interval. The granularity is the time unit (day, hour, minute, or second) by which the license server rounds up the borrow interval.

For example, if the borrow interval is 60 seconds, and the borrow granularity is `day`, then a license issued at 5:05:01 PM expires at 11:59:59 PM—which is the borrow interval (5:06:01 PM) rounded to the end of the nearest day. Likewise, if granularity is `minute`, expiration is at 5:06:59 PM. If the granularity is `second`, expiration is 5:06:01 PM.

By default, the granularity is set by the producer on the license server, and the default is `second`. In addition, a capability request can specify a granularity, which will override the granularity set on the license server.

Renew Interval

The renew interval can be used to specify how often—if ever—the client may attempt to recontact the local license server. Successful contact extends the expiration based on the effective borrow interval (in other words, the timer for the effective borrow interval is restarted).

The producer can set the renew interval at feature level in the back office, or at server level using the property `licensing.renewInterval` in `producer-settings.xml`. The renew interval is set as a percentage of the effective borrow interval. The default value is **15**. For information on how the effective borrow interval is calculated, see Determining the Effective Borrow Interval.

Important ▪ *This specification by itself does not lead to enforcement. The client-side APIs must extract this value from the license server capability response and take appropriate action.*

Example

The following example illustrates how the renew interval might be used:

Settings: Effective borrow interval = `604800s` (1 week); renew interval = `15`

Suppose the device initially obtains its license on January 1. The 7-day effective borrow interval results in a license that is valid until January 8. The 15% renew interval indicates that the device should attempt to recontact the server after 90720 seconds, that is, on January 2 (15% of 604800 seconds = 90720 seconds = approx. 1 day). If successful, the license is now valid until January 9. If unsuccessful, the license remains valid until the end of the original borrow period, which is still January 8.

Allocating Licenses Using Named License Pools

Important ▪ *You can use either reservations or named license pools for license allocation. Reservations and named license pools cannot coexist alongside each other. If currently no license reservations are defined for your license server, review the information in Comparison of Named License Pools and Reservations to decide which approach is best for you. In general, reservations are conceptually similar to named license pools, but named license pools offer significantly more flexibility in controlling a server's license estate. For more information, see Named License Pools vs. Reservations.*

Note ▪ *In previous releases of the FlexNet Embedded license server, named license pools used to be referred to as "partitions".*

In a generic setup, each license server has a license pool which contains all the licenses available to its client devices. The license server distributes licenses from this license pool on a first-come-first-served basis to client devices requesting them. This license pool is also referred to as the *default license pool*.

However, in addition to the server's default license pool you can create *named license pools*, which divide your license estate into groups of licenses. By defining rules of access, you can allocate licenses to a group of client devices or users to help ensure that these entities have access to the features they need.

Thus, a license server serves licenses from its default license pool and any named license pools that you create. Each named license pool contains license counts for one or more features, which can be allocated to client devices or users. It is important to note that in a named license pool, licenses are not assigned to individual client devices or users. Instead, counts allocated to a named license pool can be accessed by a number of devices or users that meet certain criteria, such as, for example, be member of a particular business unit or have a certain hostid. Allocated license counts can be checked out by any device or user that fulfills the specified criteria.

As the license server administrator, you can choose to allocate the entire pool of licenses, a portion of the licenses, or none of the licenses. Any license that is not allocated to a named license pool remains in the default license pool and is available to any device on a first-come-first-served basis.

The following sections provide information about named license pools on the FlexNet Embedded license server:

Named License Pool Terminology

The following table provides brief definitions for the terminology used around the named license pool functionality:

Table 4-1 ▪ Named License Pool Terminology

Name	Definition
License pool	When no named license pools have been defined, “license pool” means the license estate from which the license server serves licenses to its client devices. In setups where named license pools have been defined, the term “license pool” is used to refer to a nondescript license pool, that is, it can refer to a *default license pool* and/or *named license pool*.
Default license pool	Every server has a default license pool which cannot be deleted. If no *named license pools* have been defined, the *default license pool* holds all licenses on the server. If one or more named license pools have been defined, the default license pool holds any licenses that have not been allocated to a named license pool.

Table 4-1 ▪ Named License Pool Terminology

Name	Definition
Named license pool	A pool that holds any licenses that have been allocated to it by rules of access. A named license pool is separate from the default license pool.
Model definition	The model definition specifies the license pools and the rules of access that define how licenses are allocated to license pools (named or default).
Rules of access	Rules of access define the criteria that allow or block certain client devices or users from obtaining licenses from specified license pools (named or default).
Conditions	Criteria used in rules of access that allocate licenses to license pools (named or default).
Feature slice	Feature slices are portions of feature counts allocated to a named license pool. A slice can contain only counts from a single feature. Each slice has a unique ID.
Under-allocated	Status of a feature slice in a named license pool if it has fewer counts than specified by the active model definition. For an example, see Counts in Use in section When a New Model Definition Is Uploaded.
Over-allocated	Status of a feature slice in a named license pool if it has more counts than specified by the active model definition. For an example, see Counts in Use in section When a New Model Definition Is Uploaded.

Introduction to the Model Definition

Named license pools are defined using a *model definition*. The model definition is configured and uploaded per local license server instance or Cloud Licensing Service (CLS) instance. You can use the FlexNet License Server Administrator command-line tool to view or delete named license pools and license count allocations on the license server.

A model definition specifies the following:

- **License pools**—Named license pools are optional. If a model definition does not include any named license pools, all counts are placed in the default license pool, and rules of access can only allow or deny access to the default license pool.

  If a model definition contains named license pools, each named license pool will hold one or more features. Within the license pools (named or default), counts are arranged in *feature slices* based on their feature ID, meaning each slice contains only counts from a single feature (which has a unique feature ID). For each feature, the license pool specifies the name of the feature, its version, and its feature count. The feature count can either be specified as a number or as a percentage of the total of the available feature count.

- **Rules of access allowing access to licenses**—Rules allow client devices or users that fulfill certain conditions access to counts in specified license pools (named or default), in a given order. For example, you could specify that only clients of a specific host type can receive licenses from a certain named license pool.

- **Rules of access denying access to licenses**—You can define criteria that block certain client devices or users from obtaining licenses. This means that no licenses will be allocated to entities meeting those criteria, even if sufficient licenses are available on that license server.

After the model definition has been uploaded to the license server, the definition is saved in the database. Named license pools and rules of access defined in the model definition persist even if the license server is restarted.

Note ▪ *The size of the model definition must not exceed 900 KB.*

Refer to the following section, Model Definition Components, for information about how to create named license pools and building rules of access.

For detailed information about the format and grammar of the model definition, see Model Definition Grammar for Named License Pools. This section also contains some sample definitions that help you write your own definition.

For information about managing (uploading, deleting, and viewing) the model definition, see Managing Named License Pools.

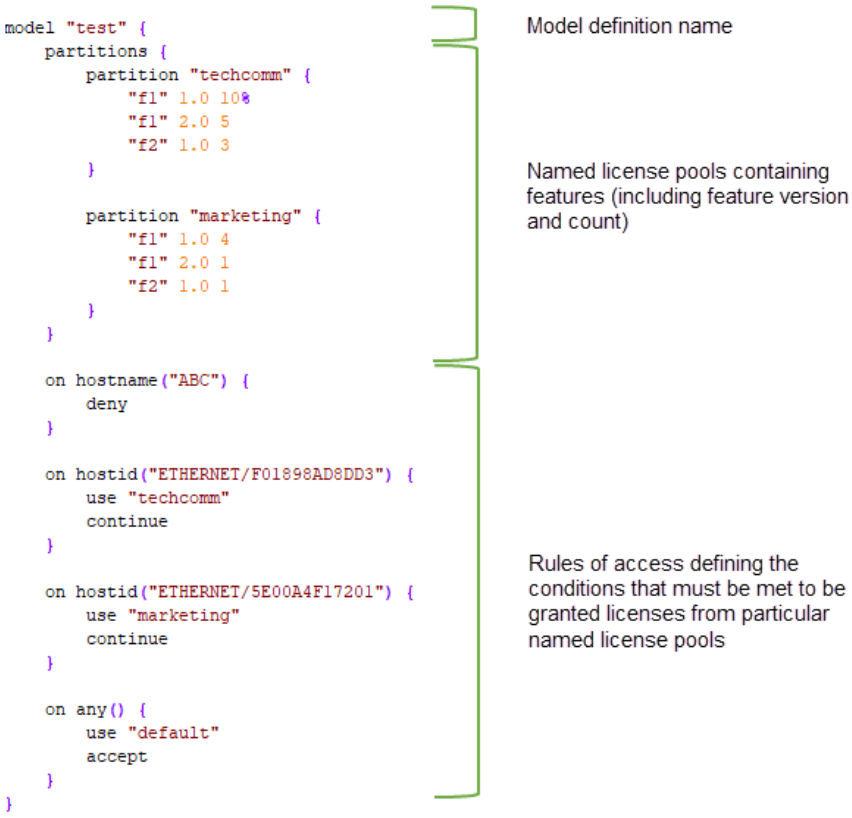
Model Definition Components

A model definition typically contains the following entities:

- Model
- Named License Pools
- Rules of Access and Conditions

A full description of the grammar and valid values can be found in the appendix, Model Definition Grammar for Named License Pools. This appendix also includes a number of use cases and sample model definitions that help you write your own model definition.

The following example shows a simple model definition. The individual components are described in the following sections.



Model

The top line of the model definition specifies the model name. The model name cannot be “default” or “reservations”, because these names are reserved.

Example

```
model "test"
```

Named License Pools

You can define one or more named license pools under the optional `partitions` element. If you do not specify a named license pool, all feature counts are placed in the default license pool. (The naming of the element stems from previous releases of the FlexNet Embedded license server, where named license pools were referred to as “partitions”.)

Each named license pool is introduced by a `partition` element, followed by the name of the license pool. You then specify one or more features that the named license pool should contain. For each feature, you need to specify the feature name, feature version, and feature count. The feature count can be expressed as a number or as a percentage of the overall feature count that is available on the license server.

Example

```
partitions {
    partition "p1" {
```

```
        "f1" 1.0 50
        "f1" 2.0 25
        "f2" 1.0 100
    }
}
```

Limitations

In a model definition, the features defined in each `partition` element must be unique. In other words, a named license pool with the following features is not allowed:

```
partition "p1" {
    "f1" 1.0 50
    "f1" 1.0 25
}
```

However, when the license server creates a named license pool, it may create two slices of a feature that has the same name and version. This can occur only if the features have different feature IDs, for example, because they differ in their expiry dates, as shown in this example:

- Feature slice 1: `fid1 f1 1.0 10 permanent`
- Feature slice 2: `fid2 f1 1.0 10 2028-12-31`

Further Information

The following section, Rules of Access and Conditions, describes the syntax and condition types used in rules of access.

For information about how the license server allocates feature counts to license pools, see Server Behavior When Distributing Feature Counts to Named License Pools.

Rules of Access and Conditions

Rules of access define criteria that allow or block certain client devices or users from obtaining licenses from specified license pools. This section covers the following topics:

- Syntax for Rules of Access—Explains the syntax for building rules, covering operators and their precedence and common keywords.
- Condition Types—Introduces the conditions you can use to allow or deny access to feature counts.
- Advanced Keywords and Directives—Covers how to use advanced keywords to build more sophisticated rules of access.

For detailed information about the format and grammar of the model definition, see Model Definition Grammar for Named License Pools. This section also contains some sample definitions that help you write your own definition.

Syntax for Rules of Access

Rules of access are defined using the following simplified syntax:

```
on <condition> {
    [use "<license-pool-name>"]
    [accept|deny|continue]
}
```

where <condition> specifies one or more conditions that must be matched.

Example

If the condition dictionary("business-unit" : "engineering") evaluates to true—that is, the capability request's vendor dictionary contains the business-unit:engineering key-value pair—access to the named license pool engineering is granted:

```
on dictionary("business-unit" : "engineering") {
    use "engineering"
    accept
}
```

The following subsections provide more information about configuring rules of access:

- Conditions Syntax—Describes how to configure a condition in simple terms.
- AND, OR, and NOT Operators—Describes how to combine conditions within a rule of access.
- Basic Keywords and Their Actions—Explains the keywords accept, deny, and continue.
- Default Behavior—Describes how feature counts are served when a feature request does not meet any of the conditions in the model definition.

Conditions Syntax

This section describes the syntax for defining conditions in simple terms. For detailed information, see Model Definition Grammar and Syntax—EBNF.

The following shows a simplified conditions syntax:

```
[NOT]<condition_type>(<parameter_list>) [AND|OR [NOT] <condition_type>(<parameter_list>)]
```

where <condition_type> is replaced with a condition type (hostid, hostname, hosttype, dictionary, userIdentifier, or any), followed by a value or value/type pair that qualifies the condition.

Optionally, additional conditions—<condition_type>(<parameter_list>)—can be chained on using an AND, OR, or optional NOT operator.

AND, OR, and NOT Operators

The syntax supports the AND, OR, and NOT operators to combine conditions within a rule of access to create a more complex rule. The operators have synonyms that are familiar to C and Java programmers:

Table 4-2 ▪

Operator	Synonym
AND	&&
OR	||
NOT	!

The syntax for combining conditions within a rule of access using these operators is illustrated in detail in section Model Definition Grammar and Syntax—EBNF. You may also find it helpful to review the provided sample rules of access.

Precedence Considerations

The AND and OR operators have left-to-right associativity; NOT has right-to-left associativity. NOT has a higher precedence than AND and OR.

Parentheses are not supported. An expression like this:

```
on hosttype("tv") and not hostname("xyz") || hostid("h1")
```

is interpreted as :

```
OR(
    AND(hosttype("tv"), NOT(hostname("xyz"))),
    hostid("h1")
)
```

Sample Rules Of Access

The following sample rules of access in the appendix Model Definition Grammar for Named License Pools illustrate the use of operators:

- Use Case: Exclusive Use of Feature Counts for Business Unit With Exception of Specific Clients
- Use Case: Exclusive Use of Feature Counts for Business Unit and Specified Clients from other Business Units
- Use Case: Assigning Features Based on Combined hosttype and hostname Properties

Basic Keywords and Their Actions

The following table explains the keywords `accept`, `continue`, and `deny` and their actions. For additional keywords and directives, see Advanced Keywords and Directives.

Table 4-3 ▪ Basic Keywords

Keyword	Action
`accept`	If the conditions of the rule of access are met and the rule ends with `accept`, the feature request is granted. No further rules of access are evaluated.
`continue`	The `continue` keyword causes rules of access on subsequent lines to be evaluated, allowing multiple rules of access that match a single request. The `continue` keyword therefore allows clients to accumulate counts from more than one license pool. For a use case example, see Use Case: Accumulating Counts from Multiple Named License Pools (“Continue” Action). This is the default behavior if no other keywords are present. The lack of an `accept` or `deny` action also means that the `on any` clause (see Default Behavior, below) will be in scope to be matched.

Table 4-3 ▪ Basic Keywords

Keyword	Action
deny	If the conditions of the rule of access are met and the rule ends with **deny**, the feature request is refused. No counts are acquired and any previously held counts are returned to the server. No further rules of access are evaluated. If the conditions of the rule of access are not met and the rule ends with **deny**, the next rule is evaluated.

Default Behavior

When a feature request does not satisfy any of the rules of access in the model definition, an `on any()` condition is expected that either denies or allows access to a license pool. If no such `on any()` condition is provided, the default is that the feature will be served from the default license pool (if features are available). This behavior is equivalent to the following code:

```
on any() {
    use "default"
    accept
}
```

The examples in this book generally do not explicitly show this final `on any()` condition.

Condition Types

This section describes the following condition types:

- Hostid Condition
- Hostname (Device Name) Condition
- Hosttype (Device Type) Condition
- Vendor Dictionary Condition
- User Identifier Condition

These condition types can be combined using AND, OR, and NOT, to form more complex rules of access. For more information, see AND, OR, and NOT Operators.

Note that in this section, the syntax describing each condition has been simplified. For a detailed syntax description, refer to Model Definition Grammar and Syntax—EBNF.

Hostid Condition

When a client requests a license from the license server, the client includes the unique hostid in the request to which the license server binds the licenses sent in the response. The hostid condition enables you to allocate licenses from a specified license pool to one or more client devices that are identified by a hostid.

The hostid is specified as a *value/type* pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.

Syntax

```
hostid_condition = 'hostid', '(', parameter_list, ')' ;
```

Example

The following hostid condition allows any feature requests with hostid `h1` or `h2` to use features from the named license pool `p1`.

```
on hostid("h1", "h2") {
    use "p1"
    accept
}
```

Tip ▪ *For examples that demonstrate the use of the hostid condition, see the following use case sections:*

- *Use Case: Simple Allow List*
- *Use Case: Simple Block List*
- *Use Case: Exclusive Use of Feature Counts for Business Unit With Exception of Specific Clients*
- *Use Case: Exclusive Use of Feature Counts for Business Unit and Specified Clients from other Business Units*

Hostname (Device Name) Condition

Your software producer can set a host name on a client device. (This host name is also sometimes referred to as a device name.) A host name is a human-readable “alias”—in contrast to the hostid—which can optionally be included in a capability request. For information about host names that might be available, contact your software producer.

Syntax

```
hostname_condition = 'hostname',  '(', parameter_list, ')' ;
```

Example

The following host name condition allows any feature requests with host name `ABC` to use features from the named license pool `p2`.

```
on hostname("ABC") {
    use "p2"
    accept
}
```

Note that the host name is case sensitive.

Tip ▪ *For examples that demonstrate the use of the hostname condition, see the following use case sections:*

- *Use Case: Assigning Features Based on Combined hosttype and hostname Properties*
- *Use Case: Using Regular Expression to Allocate License Counts*
- *Use Case: Letting Server Specify Counts*

Hosttype (Device Type) Condition

Your software producer can set a host type on a client device. (This host type is also sometimes referred to as a device type.) A host type can optionally be included in a capability request. For information about host types that might be available, contact your software producer.

Syntax

```
hosttype_condition = 'hosttype', '(' parameter_list, ')' ;
```

Example

The following host type condition allows any feature requests with host type `device` to use features from the named license pool `p3`.

```
on hosttype("device") {
    use "p3"
    accept
}
```

Note that the host type is case sensitive.

***Tip** ▪ For examples that demonstrate the use of the hosttype condition, see the following use case sections:*

- *Use Case: Assigning Features Based on Combined hosttype and hostname Properties*
- *Use Case: Device-specific Handling—Sharing Feature Counts Based On Hosttype*

Vendor Dictionary Condition

The vendor dictionary enables the software producer to send custom data in a capability request (in addition to the FlexNet Embedded–specific data) to the license server and vice-versa. Basically, the vendor dictionary provides a means to send information back and forth between the client and server for any producer-defined purposes, as needed; FlexNet Embedded does not interpret this data.

Vendor dictionary data is stored as key–value pairs. The key name is always a string, while a value can be a string or a 32-bit integer value. (In certain cases, the value can also be an array of a string and/or 32-bit integer.) Keys are unique in a dictionary and hence allow direct access to the value associated with them.

For information about whether the vendor dictionary supports arrays and the data that might be available, contact your software producer.

Syntax

```
vendor_dictionary_condition = 'dictionary', '(', keyword_parameter_list, ')' ;
```

Example

The following vendor dictionary condition allows any feature requests from the `engineering` business unit (as defined in the vendor dictionary) to use features from the named license pool `p4`.

```
on dictionary("business-unit" : "engineering") {
    use "p4"
    accept
}
```

Tip ▪ *For examples that demonstrate the use of the vendor dictionary condition, see the following use case sections:*

- *Use Case: Sharing Counts Between Business Units*
- *Use Case: Assigning Extra Counts To Business Unit*
- *Use Case: Exclusive Use of Feature Counts for Business Unit*
- *Use Case: Exclusive Use of Feature Counts for Business Unit With Exception of Specific Clients*
- *Use Case: Exclusive Use of Feature Counts for Business Unit and Specified Clients from other Business Units*
- *Use Case: Assigning Features Based on Vendor String Property*
- *Use Case: Using Regular Expression to Allocate License Counts*
- *Use Case: Making Feature Counts Available to Multiple Business Units*
- *Use Case: Accumulating Counts from Multiple Named License Pools (“Continue” Action)*

User Identifier Condition

The user identifier condition is useful when access to software features is granted based on individual user identities. To track a unique user across multiple device, every access request must include a user identifier in the `userIdentifier` field.

The benefit of the user identifier condition is that it allows license administrators to create more granular license allocation policies by applying rules at the user level, in combination with existing conditions such as host identifiers and vendor dictionary metadata.

Syntax

```
user_identifier_condition = 'userIdentifier', '(', parameter_list, ')' ;
```

Examples

The following user identifier condition allows any feature requests with user identifier `alice`, `james`, or `umi` to use features from the named license pool p5.

```
on userIdentifier("alice", "james", "umi"){
    use "p5"
    accept
}
```

Note ▪ *User identifiers are treated as case insensitive. For example, the access rule on `userIdentifier("user1")` is the same as rule on `userIdentifier("User1")`, they both grant access to `user1`.*

Tip ▪ *For examples that demonstrate the use of the user identifier condition, see the following use case sections:*

- *Use Case: Restrict Access to a License Pool for a Specific User*
- *Use Case: Allow Specified Users Access to Specified License Pool*

Best Practices and Limitations

When using user-based conditions for named license pools, license administrators should follow these best practices to ensure predictable behavior and optimal performance.

- License administrators must not create a large number of license pools on a per-user basis. Creating a separate license pool for each user does not scale and can lead to excessive rule complexity and performance degradation. For an example of users sharing licenses from the same pool, see Use Case: Allow Specified Users Access to Specified License Pool.
- License administrators should limit the number of license pools per Cloud Licensing Service (CLS) instance or local license server to a maximum of 1200.
- Individual rule updates per user or per model are likely to exceed the effective limits of the **/rules** endpoint, which can negatively impact system performance. To protect system stability, after each rule update, the local license server enforces a lock and does not accept another rule update for 120 seconds.

Advanced Keywords and Directives

The following table introduces advanced keywords and directives and their usage. For a description of the keywords `accept`, `continue`, and `deny`, see Basic Keywords and Their Actions.

Table 4-4 ▪ Keywords and Directives for Rules of Access

Keyword or Directive	Description	Refer to Section
`remainder`	Allocates any remaining license counts to a specified license pool. Useful in scenarios where integer rounding could cause left-over counts to remain in the default license pool when trying to configure license pools summing up to a count of 100%.	Allocating Remaining License Counts
`vendor string matches`	Allocates feature counts to license pools based on variables specified in the vendor string.	Allocating Counts Based on Vendor String Property
`max`	Limits the number of feature counts that a single user or device can consume.	Limiting Checkout of Feature Counts per User or Device
`without requested features`	Lets the license server specify the features that should be served to a client when a capability request lists no desired features.	Letting Server Specify Counts
[regular expressions]	Match patterns to reduce the number of license pools and conditions. Syntax: `~/REGEX/`	Using Regular Expressions

Allocating Remaining License Counts

The `remainder` keyword can be used to allocate any remaining license counts to a specified named license pool. This keyword is particularly useful in preventing the situation where integer rounding would cause left-over counts to remain in the default license pool when trying to configure named license pools summing up to a count of 100%.

Using the `remainder` keyword is equivalent to specifying `100%`, which allocates all available license counts. Any license pool later in the model definition than the one specifying the `remainder` keyword (or `100%`) in the model definition (including the default license pool) receives zero counts for that feature. For an example demonstrating its use, see the Use Case: Named License Pool Receiving Entire Remaining Feature Count.

Example

```
model "exampleModel" {
    partitions {
        partition "p1" {
            "f1" 1.0 33%
        }

        partition "p2" {
            "f1" 1.0 33%
        }

        partition "p3" {
            "f1" 1.0 remainder
        }
    }
}
```

Allocating Counts Based on Vendor String Property

The directive `vendor string matches` enables license administrators to allocate feature counts to license pools based on variables specified in the vendor string. After use, feature counts are returned to their original license pool.

To use the directive, the producer must define a vendor string in the license model when they create a line item in the back office. The vendor string can hold arbitrary producer-defined license data. Data can range from feature selectors to pre-defined substitution variables.

The directive `vendor string matches` is located in the `partitions` section of the model definition. This directive is evaluated when the license server loads the model definition.

Syntax Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            f1 1.0 75% vendor string matches "ProductName:Premium"
            f2 1.0 75% vendor string matches "ProductName:Premium"
        }
        partition "sales" {
            f1 1.0 25% vendor string matches "ProductName:Basic"
            f2 1.0 25% vendor string matches "ProductName:Basic"
        }
    }
```

```
}
```

For a use case example, see Use Case: Assigning Features Based on Vendor String Property.

Limiting Checkout of Feature Counts per User or Device

Use the `max` keyword to limit the number of feature counts that a single user or device can consume.

`max` is used in the `partitions` section of the model definition. The following shows an example model definition which limits the consumption of features f1 and f2 to 10 and 1 counts, respectively:

Syntax Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "f1" 1.0 100 max 10
            "f2" 1.0 100 max 10
        }
        partition "sales" {
            "f1" 1.0 5 max 1
            "f2" 1.0 5 max 1
        }
    }

    on dictionary("business-unit" : "engineering") {
        use "engineering"
        accept
    }

    on dictionary("business-unit" : "sales") {
        use "sales"
        accept
    }
}
```

If `max` is set to 0, access to that feature from the license pool is denied.

If a client requests a feature count that is greater than the value of `max`, the request is denied with the error `FEATURE_COUNT_INSUFFICIENT`.

To allow a partial checkout, set the “partial” attribute to true. In that scenario, a client is granted access to the requested number of features up to the value of `max`, even if a feature count exceeding `max` is requested.

Letting Server Specify Counts

There might be scenarios when you want the license server to specify the features that should be served to a client. The server cannot override a capability request containing desired features coming from the client. However, if a capability request lists no desired features, the `without requested features` directive lets the server allocate feature counts.

You can use the `without requested features` directive to do the following:

- **Allocate specific feature counts.** To achieve this, list the feature counts that should be served. The following example shows the syntax:

  ```
  on hostname("myhost") {
  ```

```
        use "p1", "default"
        without requested features {
            "f1" 1.0 1
        }
        accept
    }
```

For a use case example, see Use Case: Letting Server Specify Counts.

- **Allocate the entire feature count from a license pool.** To achieve this, combine `without requested features` with the `all from "license_pool_name"` instruction. Replace *`license_pool_name`* with the name of the license pool whose entire list of features, as specified in the model definition, should be allocated. Note that if the specified license pool does not have sufficient counts to satisfy the request, counts are allocated from subsequent license pools listed in the rules of access (provided that the client has access to those license pools). The following example shows the syntax:

```
on hostname("myhost") {
    use "p1", "default"
    without requested features {
        all from "p1"
    }
    accept
}
```

For a use case example, see Scenario: Server-specified Counts.

Using Regular Expressions

Rules of access can include regular expressions to match patterns in conditions. This means that the number of license pools and conditions can be greatly reduced, simplifying management of license pools.

Regular expressions can be included in conditions using the following syntax: `~/REGEX/`

For a list of supported expressions, see https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/regex/Pattern.html.

Full vs Partial Equality

The regular expression is generally matched against the whole value of the property on which the condition is based (that is, hostid, hostname, hosttype, or vendor dictionary string). It is not necessary to add the ^ and $ meta characters to signify the beginning and end of the expression.

Example

The following expression...

```
on hostname("~/[A-Z]{2}[0-9]{3}/") {
    ...
}
```

... matches the hostname `AB123`, but not `ABCD1234`.

However, when a regular expression is used in combination with the `vendor string matches` directive, the expression is used as a “contains” match.

Example

The following expression...

```
… vendor string matches "~/%%ProductName:[A-Z]{3}[0-9]{3}%%/
```

... matches `%%ProductName:AAA111%%` found anywhere inside the vendor string.

For a use case example, see Use Case: Using Regular Expression to Allocate License Counts.

Validating Model Definitions for Named License Pools

The license server validates all model definitions. Invalid model configurations are rejected to prevent ambiguous license assignments.

Examples for Rejected Model Definitions

The following examples illustrate model definitions that will be rejected.

- **Duplicate Feature Entries (Same Name and Version)**

  If a model contains multiple entries for the same feature name and version, regardless of the assigned counts, it will be considered invalid.

  ```
  partition "p1" {
      "f1" 1.0 10
      "f1" 1.0 5
  }
  ```

- **Duplicate Feature Entries with Matching Vendor Strings**

  If the feature name, version, and vendor string all match across multiple entries, the model will be rejected.

  ```
  partition "p1" {
      "f1" 1.0 10 vendor string matches "abc"
      "f1" 1.0 20 vendor string matches "abc"
  }
  ```

Error Message for Rejected Model Definitions

In case of a rejected model, an error message similar to the following is displayed:

```
{
    "key": "glsErr.invalidModel",
    "message": "Invalid model posted, Duplicate feature entries found : [feature=ss_f1, version=1.0,
    vendorString=abc]",
    "arguments": [
        "Duplicate feature entries found : [feature=ss_f1, version=1.0, vendorString=abc]"
    ]
}
```

Server Behavior When Distributing Feature Counts to Named License Pools

This section describes the distribution of feature counts to named license pools for these situations:

- When a New Model Definition Is Uploaded
- When the Feature Count Changes on the License Server

- When Clients Return or Renew Counts to the Server

When a New Model Definition Is Uploaded

When the model definition is uploaded to the license server, feature counts are placed into license pools. Counts are allocated to named license pools in the order that is specified in the model definition, starting with the topmost named license pool.

The following rules of access apply when allocating feature counts of the same name, but with different versions and expiry dates:

1. Features that match the version specified in the model definition are allocated first. Features with longer expiry dates are allocated before features with a shorter expiry date.
2. If no features are available that match the version specified in the model definition, features of the next higher version are allocated. Features with longer expiry dates are allocated before features with a shorter expiry date. (Features of a lower version are not allocated.)

Features With Future Start Date

Features holding a start date in the future will not be allocated to named license pools but will remain in the default license pool. A feature with a future start date will stay in the default license pool even beyond its start date until the model definition is reapplied to the license server.

Grouping of Features in License Pools

Within named license pools, counts are arranged in feature slices based on their feature ID, meaning each slice contains only counts from a single feature (which has a unique feature ID). Named license pools that have the exact feature counts, including the exact feature versions, as specified in the model definition are considered *fully satisfied*.

When the available number of counts has been distributed, any named license pool for which insufficient counts are available will remain empty or will only be partially filled.

Counts in Use

When the model definition on the license server changes, any counts that are currently in use by clients are recorded as used against the named license pool the counts originate from. If a named license pool is deleted, it stays active until all used counts have been returned to it. Only after that is the named license pool removed. This enables the license administrator to maintain an overview of all used feature counts.

As a result of this behavior, license pools can be under-allocated or over-allocated.

The following example illustrates how under- or over-allocation can occur:

1. The model definition allocates **8** counts each of features **f1** and **f2** to the named license pool **P1**.
2. Users check out **6** counts of feature **f2**:

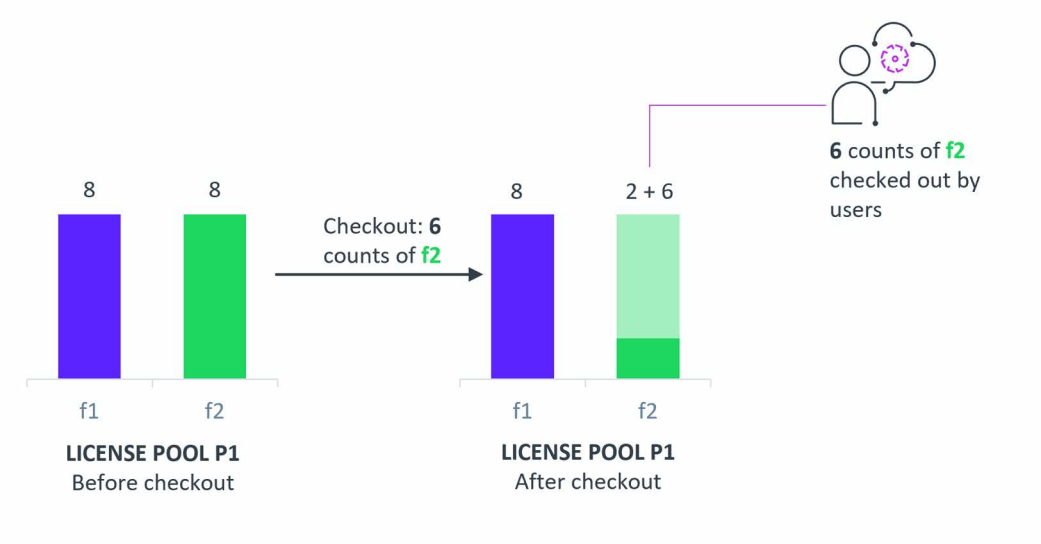


Figure 4-2: Diagram showing the license count allocation of f1 and f2 to named license pool P1, and users subsequently checking out counts of f2.

3. An updated model definition is posted, which moves all 8 counts of f2 to another license pool **P2**.
4. The following diagram shows the allocation as specified by the updated model, and the actual allocation:

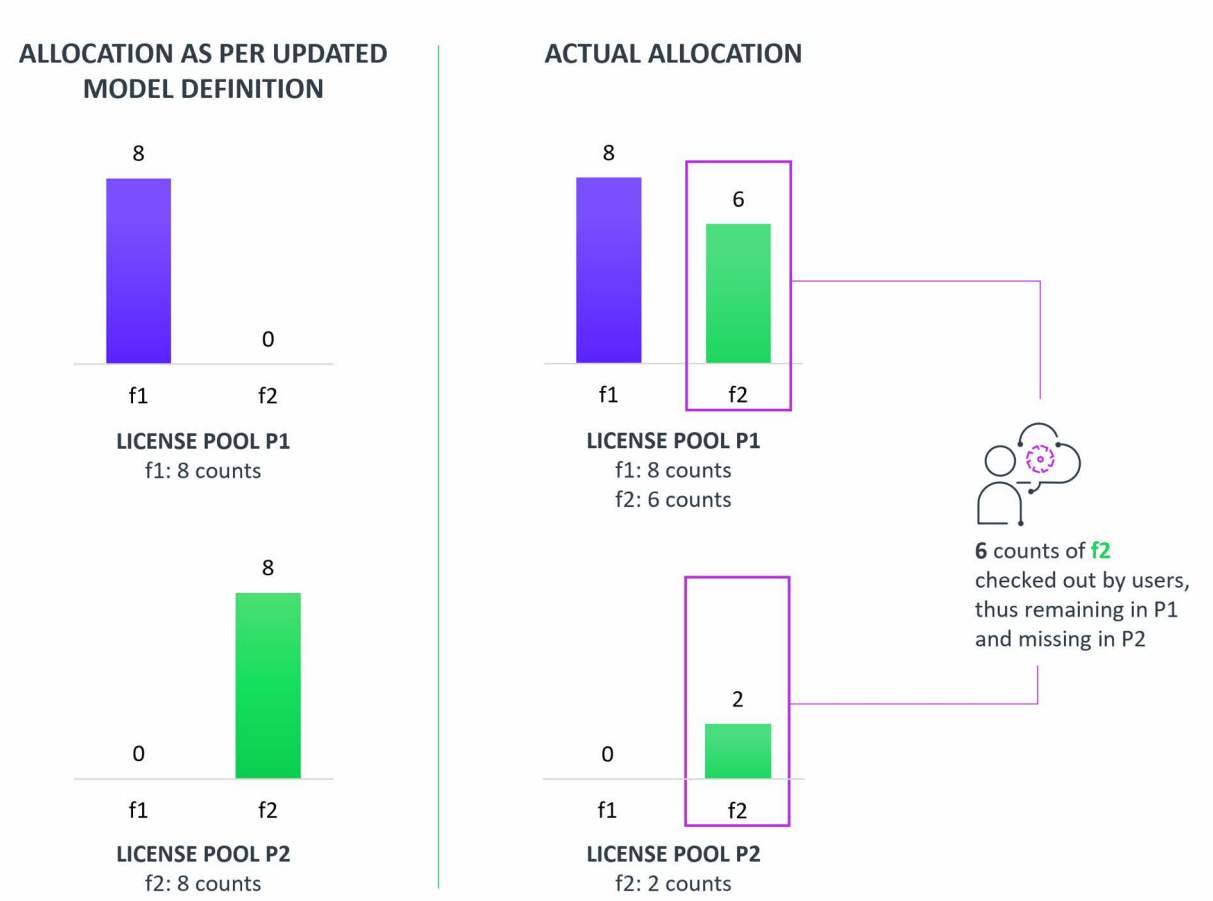


Figure 4-3: Diagram showing the license count allocation after the model update while counts were in use.

Note that the actual allocation of counts differs from that specified in the model, due to 6 counts being in use when the model was updated:

- **P1** should have 0 counts of f2, but retains the 6 counts that are in use. The feature slice of f1 in **P1** is therefore OVER-ALLOCATED.
- **P2** should have 8 counts of f2, but receives only 2 counts. The feature slice of f1 in **P1** is therefore UNDER-ALLOCATED.

When the 6 used counts in P1 expire or are returned, the license pools are recomputed according to the new model. All 6 counts are moved from P1 to P2 as per the model definition, and the feature slices for f1 in P1 and P2 return to a NORMAL status.

Optimization Logic For Model Definition Changes

When the model definition changes, any named license pools that are already fully satisfied are skipped when feature counts are placed into named license pools. As a result, it is possible that existing features in fully satisfied named license pools are not replaced with newly available features that have a longer expiry date. Instead, these newly available features are placed into lower-ranked named license pools (provided that these are not fully satisfied) or the default license pool.

To ensure that higher-ranking named license pools receive features with longer expiry dates, it is recommended that you delete and re-upload the model definition. This forces a redistribution of feature counts across all license pools, regardless of the current allocation of feature counts.

License Server Lock During Model Updates

When a new model definition is uploaded, the server is locked until the model repost is completed. When it is locked, the server will not accept additional model reposts or licensing requests (preview requests, access requests or signed access requests from the Cloud Monetization APIs, or capability requests). A local license server will return error 500 or 503, and a CLS instance will return error 429 or 503. In both cases, wait until the server is unlocked before retrying.

When the Feature Count Changes on the License Server

When feature counts are added, reduced or expire on the license server, the server reapplies the model definition. The server redistributes the counts across license pools as described in When a New Model Definition Is Uploaded.

When Clients Return or Renew Counts to the Server

When there have been no changes to the model definition or to the licenses held by the server, licenses returned by clients are returned to the original slice from which they were acquired.

However, in scenarios where clients were holding usage prior to a model change or a change to the licenses held by the server, counts returned by clients may need to be redistributed to reflect the new state of the server.

The following sections describe more about feature count redistribution:

- Basic Redistribution Rules
- Order of Redistributing Counts

In these sections, *slices in deficit* refers to slices that have fewer counts than they should have, according to the model definition and the number of counts the server has been provisioned with (`"slice" < "computedCount"`).

Basic Redistribution Rules

When clients had counts in use prior to a model change or prior to a change of feature counts on the license server, the following basic redistribution rules apply:

- A client cannot renew counts that it could not have acquired, based on the current model definition that is in force on the license server.
- Counts can only be reallocated to slices with a matching feature ID.

Order of Redistributing Counts

The server redistributes returned counts in the following order:

1. The server allocates feature counts to reduce or eliminate overage before redistributing counts.
2. Counts are redistributed to slices in deficit that are accessible to the client, in the order in which license pools that are accessible to the client are specified in the current model definition.
3. Counts are redistributed to slices in deficit, irrespective of whether these are accessible to the client. The server redistributes counts to license pools in the order in which they are specified in the current model definition.
4. Returned counts are only returned to overdraft after all slices in deficit have been adjusted to the counts they should have according to the model definition. Overdraft counts are only ever available from the default license pool.
5. Any remaining counts are returned to the default license pool.

Server Behavior when Assigning Features to Clients

When the license server receives a capability request from a client, it tries to serve the feature counts matching the requested version from the first named license pool that is configured in the model definition. If the request cannot be fulfilled from the first named license pool, or can only be partially fulfilled, the server will try to serve any remaining counts to the client from further feature slices and license pools in a specific hierarchy. The following rules apply, in order, when fulfilling capability requests:

1. Regular counts take priority over overdraft counts (overdraft counts are only available from the default license pool).
2. Counts are served from accessible named license pools in the order license pools are specified in the model definition (starting from the top).
3. Counts that match the requested version are served before counts of a higher version (lower versions are not accepted).
4. Regular licenses are prioritized over licenses that are in a grace period. This means that it will only serve grace license if there are no non-grace licenses available.
5. Counts with the shorter expiry date are served before counts with a later expiry date.

The following diagrams illustrate this hierarchy. Figure 4-4 shows the order in which feature counts are assigned from license pools.

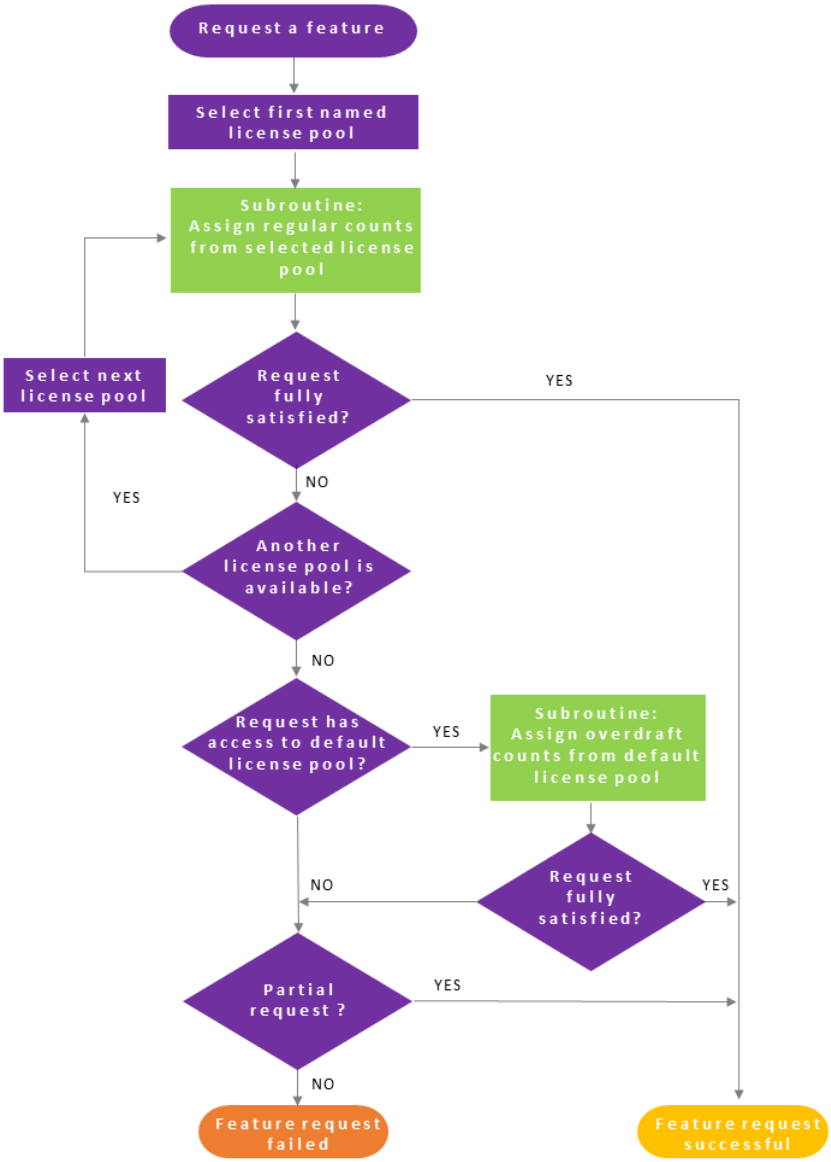


Figure 4-4: Flow of feature allocation

Figure 4-4 includes a subroutine (highlighted in green) that explains how counts are assigned if the request is served from more than one slice. This subroutine—see Figure 4-5—also comes into play when regular (non-overdraft) counts are exhausted, but the request has access to the default license pool which includes overdraft counts:

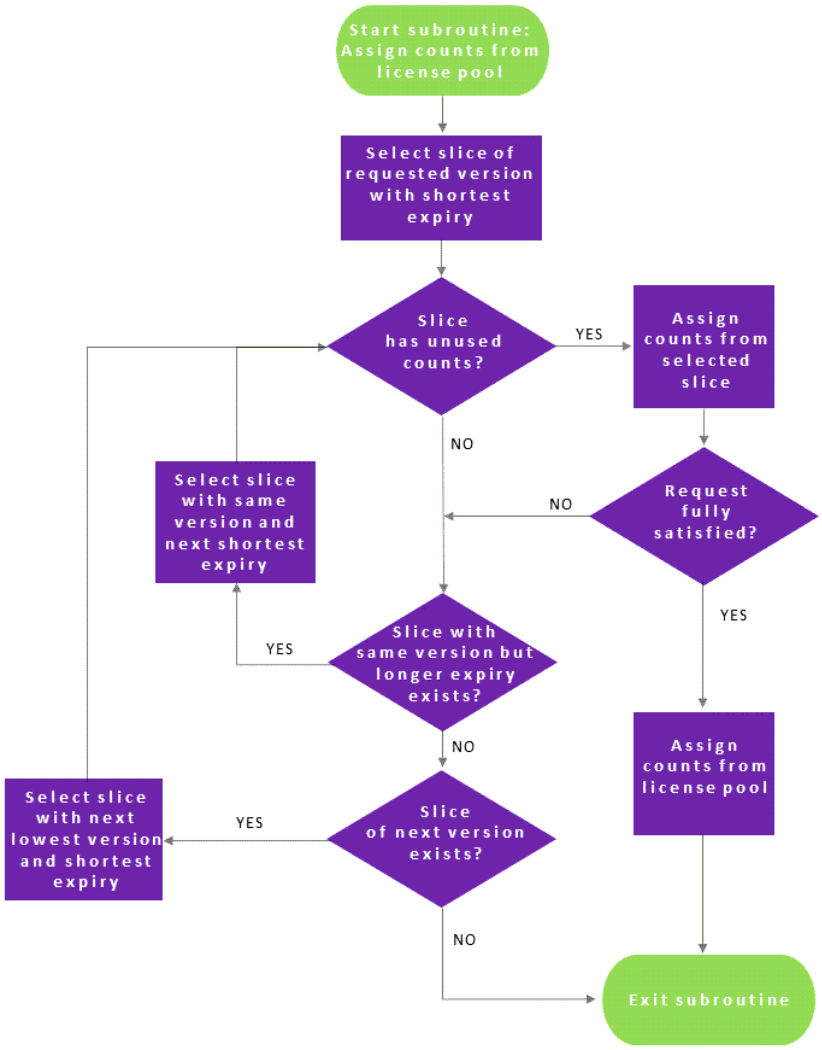


Figure 4-5: Flow of feature allocation in the subroutine

Named License Pools vs. Reservations

Named license pools are conceptually similar to reservations but offer significantly more flexibility in controlling your server's license estate. If currently no license reservations are defined for your license server, Revenera strongly recommends to use the Named License Pools functionality to allocate licenses.

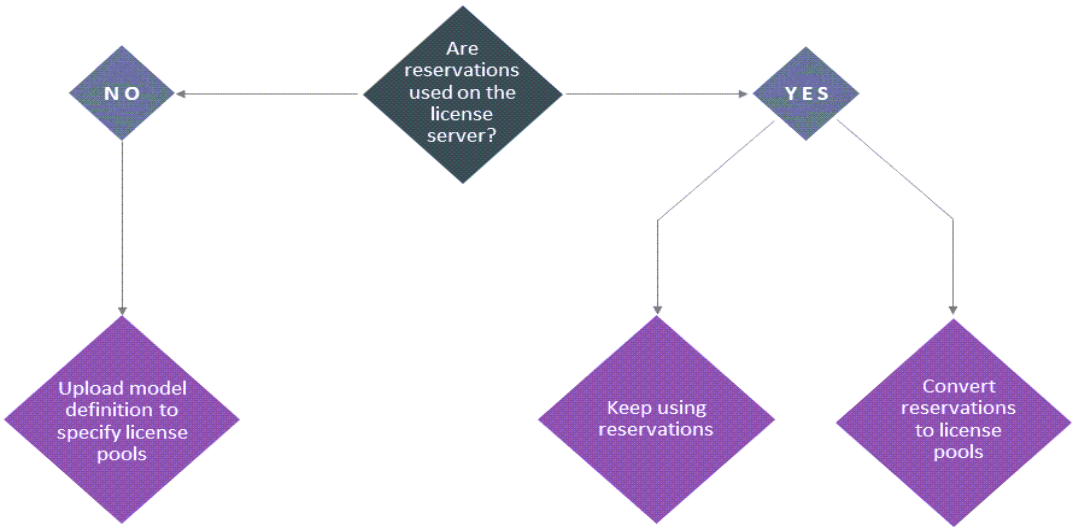
Read this section if one or more of the following statements apply to you:

- You want to know the pros and cons of using named license pools or reservations. See Comparison of Named License Pools and Reservations.
- You are currently using reservations and would like to find out how named license pools affect the behavior of reservations. See Impact of "Named License Pools" Functionality on Using Reservations.
- You have previously been using reservations and would like migrate to named license pools. See Transitioning from Reservations to Named License Pools.

- You have migrated to named license pools but want to use reservations again. See Returning to Reservations after Transitioning to Named License Pools.

If you have not been using reservations before and are now starting to use named license pools, you can skip this section.

The following diagram illustrates your options:



Comparison of Named License Pools and Reservations

If reservation groups or entries change frequently, it is recommended to use reservations. Otherwise, it is recommended to use named license pools.

The following table lists the main differences between named license pools and reservations.

Table 4-5 ▪ Named License Pools vs. Reservations

Action	Named License Pools	Reservations
Counts are allocated based on...	• hostid • host name (device name) • host type (device type) • vendor dictionary data	• hostid
If the license server has been provisioned with less counts than are posted in a reservation or are specified in the model definition, respectively,...	...the license server allows access to feature counts based on the rules of access in the model definition until all available counts are in use by entitled clients.	...the license server rejects the entire reservation group.
Users' and devices' access to feature counts can be...	...allowed or denied.	...allowed.

Table 4-5 ▪ Named License Pools vs. Reservations

Action	Named License Pools	Reservations
Criteria that allow or block certain client devices or users from obtaining licenses can be configured using...	...an extensible model language.	...a file in JSON format or by adding reservation entries using the FlexNet License Server Manager.

Impact of "Named License Pools" Functionality on Using Reservations

The introduction of the Named License Pools functionality has no impact on the reservations functionality. You can continue configuring reservations using the FlexNet License Server Administrator command-line tool or the FlexNet License Server Manager, and the license server will distribute licenses according to the reservations that have been configured.

When reservations are configured, attempts to upload a model definition will fail.

Transitioning from Reservations to Named License Pools

If you decide to move from using reservations to using named license pools, you need to perform the following steps:

1. Delete any reservation groups.
2. Create a new model definition which defines rules of access and named license pools (see Define a Model Definition and Upload the Model Definition).

The new model definition takes effect immediately after upload.

Before starting the migration, you might want to use the FlexNet License Server Administrator command-line tool (`flexnetlsadmin`) to view information about the reservations currently active on your license server. See Managing Reservations in Using the FlexNet License Server Administrator Command-line Tool.

A Note on Reserved Counts in Use

If any reserved license counts are in use by clients while transitioning from reservations to named license pools, such used counts are recorded as `used` against the default license pool. They remain in the default license pool until the client renews or returns these counts, after which point the license counts are allocated to their named license pool according to the model definition.

Returning to Reservations after Transitioning to Named License Pools

To return to using reservations, you must delete the active model definition. After that, you can configure reservations (using the FlexNet License Server Administrator command-line tool or the FlexNet License Server Manager).

For information about deleting the active model definition, see Delete the Model Definition.

Limitations of Named License Pools

Note the following about named license pools:

- Named license pools support only concurrent features, but not metered features. Metered features will always remain in the default license pool. This matches the behavior of reservations, because metered features cannot be reserved.
- Using feature selectors in combination with named license pools may produce unexpected results. A feature selector is a key-value structure included as part of the feature's definition created in the back office.
- The license server failover configuration does not support named license pools. During a failover period, feature counts will be distributed only from the default license pool.

License Reservations

Important ▪ *You can use either reservations or named license pools. Reservations and named license pools cannot coexist alongside each other. For more information, see Named License Pools vs. Reservations.For general information about named license pools, see Allocating Licenses Using Named License Pools.*

If currently no license reservations are defined for a license server, review the information in Comparison of Named License Pools and Reservations to decide which approach is best for you. In general, reservations are conceptually similar to named license pools, but named license pools offer significantly more flexibility in controlling your server's license pool.

Important ▪ *Reservations can only be used if the policy setting* ***licensing.allowDuplicateClients*** *is set to false (the default).*

The license reservation feature available with the FlexNet Embedded license server (the local license server or a CLS instance) enables you to control the distribution of the server's license pool within your enterprise.

With no reservations defined, the license server distributes licenses on a first-come-first-served basis to client devices requesting them. However, with the use of reservations, the server pre-allocates specific license counts to certain client devices or users to help ensure that these entities have access to the licenses they need. As the license server administrator, you can choose to pre-allocate the entire pool of licenses, a portion of the licenses, or none of the licenses. Any license not reserved remains in the shared pool of licenses and is available to any device on a first-come-first-served basis.

The following sections provide background information about reservations on the FlexNet Embedded license server:

- Overview of Reservation Types
- Reservation Hierarchy
- Managing Reservations
- Disabled Reservations
- License Allocation on the Server

Overview of Reservation Types

The FlexNet Embedded license server supports two types of license reservations—device-based and user-based.

- A device-based reservation is assigned to the unique hostid identifying the client device on which the software producer's product is running.
- A user-based reservation is assigned to a unique user ID (also called a *hostid*). A user with reserved counts for a feature can access these counts from any client device. Thus, user-based reservations help to ensure that users who always need access to certain licenses can obtain them from any client device. (For example, users might need to access their licensed application from their laptop, tablet, or phone.)

For information about how the license server processes reservations to satisfy the feature count specified in a capability request, see Processing the Capability Request When Reservations Are Used.

Note ▪ *A user-based reservation reserves feature counts exclusively for a given user, who can obtain partial counts of the reservation across multiple client devices. However, these feature counts are ultimately served to and returned from—and thus bound to—the client device from which they were requested.*

Reservation Setup

When setting up a license reservation on the license server, you must provide a hostid to identify the entity—client device or user—to which the reservation belongs. Obtain from the producer the specific hostid types (for example, `ETHERNET`, `STRING`, `USER`, or other) that you can use to define a device-based or user-based reservation. See Managing Reservations for more information about setting up reservations.

Reservation Hierarchy

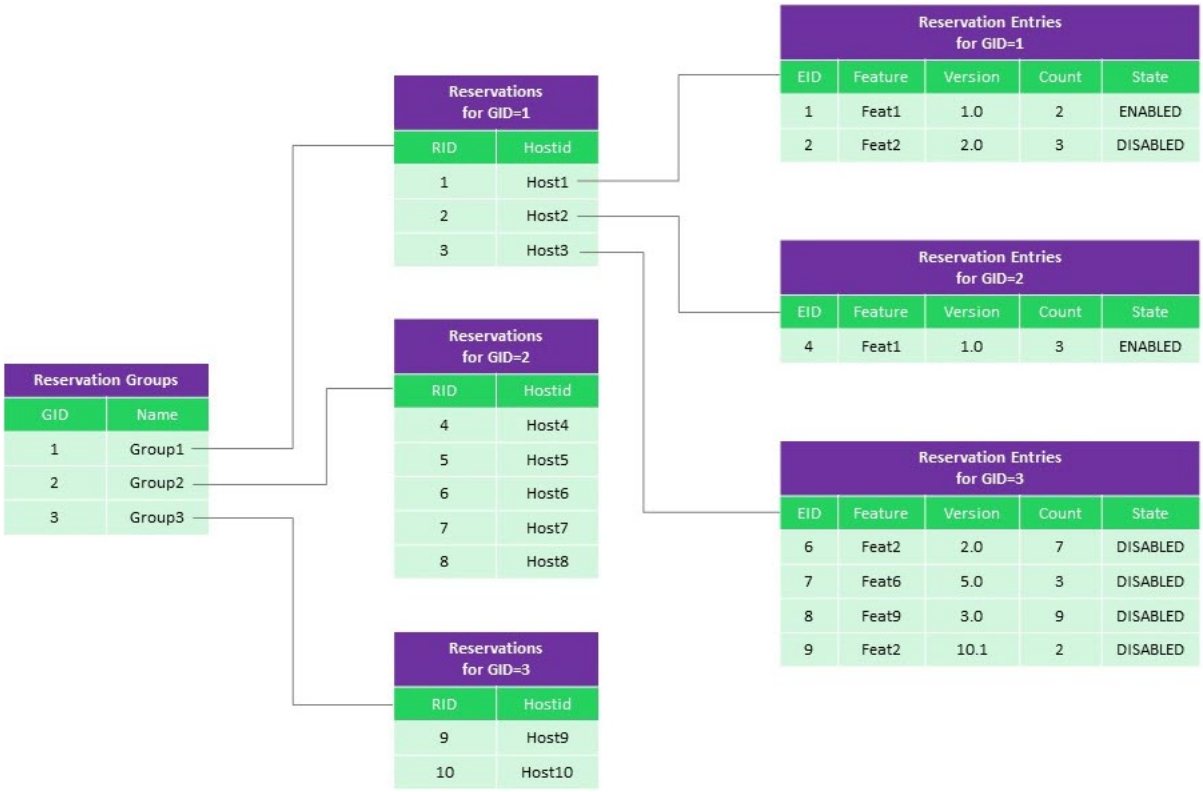
The reservations you post to the license server are ordered in a hierarchy, enabling more granularity in managing them. For example, *reservation entries* are the actual feature reservations that you define for a *reservation*, which is identified by the hostid for the specific client device or user to which the feature reservations apply. One or more reservations are assigned to a *reservation group*, which represents an entity to which the client devices and users identified by the reservations belong. This entity might be a department, a job role assigned to specific employees within the company, a physical location, or any other entity applicable to your company. This hierarchy enables you to view, add, or delete reservations at the group level, the reservation (hostid) level, or the reservation entry (feature) level.

When reservations are added to the license server, the following IDs are generated for the various hierarchal components that define a reservation:

- A *group ID* (GID) for a reservation group
- A *reservation ID* (RID) for the reservation, based on the hostid identifying a specific client device or user within the reservation group
- A *reservation-entry ID* (EID) for each reservation entry defined in a specific reservation

The following diagram illustrates the reservation hierarchy. For example, reading from right to left, trace the reservation entry **EID 1** for feature **Feat1**:

- **EID 1** belongs to the reservation **RID 1**, defined for the hostid **Host1**.
- **RID 1** is one of three reservations belonging to the reservation group **GID 1**, also named **Group1**.



Managing Reservations

Both the FlexNet License Server Administrator command-line tool and the FlexNet License Server Manager provide mechanisms for managing reservations on the license server:

- The License Server Administrator command-line tool adds or updates reservations by loading an input file (that you provide) containing the reservation definitions, including updates. The tools provides commands that delete reservations at the reservation group and reservation levels. For more information, see Using the FlexNet License Server Administrator Command-line Tool.
- FlexNet License Server Manager allows you to create and post individual reservations one at a time or import reservations from a file. All reservations are managed from a single internally created reservation group. For more information, see the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

The next sections describe what happens “behind the scenes” when reservations are managed on the license server:

- Definitions in JSON Format
- Add a New Reservation Group

- Add or Delete Reservations in an Existing Group

Important ▪ *If you were previously using named license pools, you must delete the active model definition before creating reservations. See Delete the Model Definition.*

Definitions in JSON Format

When you provide an input file that contains reservation definitions, the file contents must be in JSON format. The reservation entries for a given reservation (identified by a hostid) use the following format:

```
{ "name" :  "groupName", "reservations" :

  [
    { "hostId": { "value" : "hostid", "type" : "hostidType" },
      "reservationEntries": [
        { "featureName": "name", "featureVersion": "version", "featureCount": count },
        { "featureName": "name", "featureVersion": "version", "featureCount": count } ]},
```

Then “name” field is optional. If omitted, the license server simply generates the group ID.

Add a New Reservation Group

Important ▪ *If you were previously using named license pools, you must delete the active model definition before creating reservations. See Delete the Model Definition.*

The following shows an example of reservation definitions (in JSON format) that might be posted to define a new reservation group—in this case, a group named **support**, containing reservation entries for two reservations, one for hostid **7A7A77AAA77A** and one for hostid **Joe**:

Table 4-6 ▪ Contents of Sample License Reservations File

```
{ "name" :  "support", "reservations" :
  [
    { "hostId": { "value" : "7A7A77AAA77A", "type" : "ETHERNET" },
      "reservationEntries": [
         { "featureName": "f1", "featureVersion": "1.0", "featureCount": 1 },
         { "featureName": "f2", "featureVersion": "1.0", "featureCount": 2 } ]},
    { "hostId": { "value" : "Joe", "type" : "USER" },
      "reservationEntries": [
         { "featureName": "f3", "featureVersion": "1.0", "featureCount": 3 }]}
]}
```

When this reservation information is processed without any errors, the license server does the following:

- Creates the reservation group **support** if it does not yet exist. If it does exist, an attempt to add another group named **support** will fail.
- Adds the two new reservations—one for hostid **7A7A77AAA77A** (RID **193**) and one for hostid **JOE** (RID **196**)—to the reservation group **support**.

Note ▪ *A given hostid can be used only once not only within a given reservation group but within the entire reservation posting. An attempt to add a reservation group that includes a hostid already existing in another group will fail.*

- Creates two reservation entries for hostid **7A7A77AAA77A**—a reservation for **1** count of feature **f1** and a reservation for **2** counts of feature **f2**.
- Creates one reservation entry for **3** counts of feature **f3** for hostid **JOE**.

Errors that Cause Rejection of New Reservation Groups

When a new reservation group is processed during the posting phase, the *entire* group is rejected if any of following errors exist.

- A group name is repeated within the reservation posting. A group name must be unique within the *entire* reservation posting.
- A hostid is used in more than one reservation group. A given hostid can be used only once, not only within a given group, but within the entire reservation posting.
- A hostid is missing. A group reservation cannot be created without at least one reservation, defined by a hostid and its reservation entries. Additionally, reservation entries not associated with a reservation cause the entire reservation group to be rejected.
- A reservation has no reservation entries associated with it.
- A reservation entry is invalid. For example, the entry might be missing a feature name, version, or reserved count, or the specified reserved count is less than or equal to zero.
- The feature or feature version specified in a reservation entry is not found in the license pool.
- The feature specified in a reservation entry is expired or has not yet started.
- The feature specified in a reservation entry is a metered feature.
- The license server has an insufficient count of a given feature to fulfill the *total* requested count for that feature across all reservation entries within the entire group. For more explanation, see the later section License Allocation on the Server.

Add or Delete Reservations in an Existing Group

Currently, the FlexNet License Server Administrator command-line tool lets you add or update reservations at the reservation-group level only. Therefore, if you want to add reservations to an existing group or reservation entries to an existing reservation, you must delete the entire reservation group and then re-create it with the changes. The input file used to recreate the group should contain the reservation group, reservations, and reservation entries used to create the group initially, plus the updates.

Likewise, the License Server Administrator command-line tool provides commands that let you delete reservation groups or reservations only. To remove a reservation entry from a specific reservation, you must drop the entire reservation group and recreate it without the specific reservation entry.

On the other hand, the FlexNet License Server Manager deletes the current reservation group and creates a new one every time a reservation list is submitted to the license server (that is, every time you add, edit, or delete reservations from the **Reservations** view).

Disabled Reservations

The “disabled” state for a reservation entry means that the entry is currently not active due to an insufficient feature count. A new reservation entry is never added with a “disabled” status. However, when license rights on a the license server change, reservation entries can become disabled due to an insufficient feature count on the license server. Should sufficient counts on the license server become available again, you can delete and recreate the reservation group to re-enable the reservation entries.

License Allocation on the Server

When reservations are posted for a new group, the license server determines the total feature counts required by all the reservations within the group. If the totals cannot be satisfied by the available feature counts on the license server, the entire new group is rejected. Additionally, the license server re-allocates reservations when license counts on the server change. See the following sections:

- When Adding a Reservation Group
- When Feature Counts Change on the Server

When Adding a Reservation Group

In the attempt to add a new reservation group, the license server determines the total feature counts required by all the reservation entries within the new group. If the totals cannot be satisfied by counts not reserved to other reservation groups on the license server, the entire new group is rejected.

Example Allocations When Adding a New Reservation Group

The examples described in this section use the following reservations to pre-allocate the licenses in the server’s license pool. All these reservations are being created in the single new reservation group **payroll**.

Table 4-7 ▪ Sample Allocations When Adding a Reservation Group

```
{ "name" :  "payroll", "reservations" :
  [
    { "hostId": { "value" : "111A11AAA11A", "type" : "ETHERNET" },
      "reservationEntries": [
         { "featureName": "f1", "featureVersion": "1.0", "featureCount": 1 },
         { "featureName": "f3", "featureVersion": "1.0", "featureCount": 2 } ]},
    { "hostId": { "value" : "222A22AAA22A", "type" : "ETHERNET" },
      "reservationEntries": [
         { "featureName": "f3", "featureVersion": "1.0", "featureCount": 1 } ]},
    { "hostId": { "value" : "Joe", "type" : "USER" },
      "reservationEntries": [
         { "featureName": "f3", "featureVersion": "1.0", "featureCount": 3 }]}
]}
```

Example 1: Sufficient Feature Counts

In this example, a license server is allocated 1 count of f1 and 8 counts of feature f3 from the back office. When processing the new reservation group, the license server does the following:

- Determines that a total of 1 count of f1 is required (for device 111A11AAA11A).
- Determines that a total of 6 counts of f3 is required—**2** counts for device 111A11AAA11A, **1** count for device 222A22AAA22A, and **3** counts for user Joe.
- Reserves **6** counts of f3 and **1** count of f1.
- Keeps **2** unreserved counts of f3 available for any request on a first-come-first-served basis.

Example 2: Insufficient Feature Counts

If a license server is allocated 1 count of f1 and 4 counts of f3, it does the following when the reservations are posted:

- Determines that a total of 1 count of f1 is required (for device 111A11AAA11A).
- Determines that a total of 6 counts of f3 are required—**2** counts for device 111A11AAA11A, **1** count for device 222A22AAA22A, and **3** counts for user Joe.
- Rejects the entire reservation group since only 4 counts of f3 are available.

Example 3: Features Not in the License Pool

If a license server is allocated no (0) counts of f1 but is allocated 8 counts of f3, it rejects the entire reservation group since f1 is not available in the license pool.

When Feature Counts Change on the Server

When features counts are reduced or expire on the license server, the server automatically disables certain reservation entries to adjust to the new feature count. The server uses no particular order as it processes the reservation groups, reservations, or reservation entries to determine which reservation entries to disable.

Processing the Capability Request When Reservations Are Used

The following sections describe how the license server processes capability requests to fulfill licenses when reservations are available:

- Basic Process for Granting Licenses
- Example Scenarios of the Basic Process for Granting Licenses

Basic Process for Granting Licenses

When reservations have been added and the license server receives a capability request from a client device, the server uses the following basic process in its attempt to grant the license count requested.

Note ▪ *The process described here assumes that the capability request is defined with no special capability request options that can control aspects of the license server's behavior in determining which licenses are sent in the capability response. Consult the software producer for information about any such options that client code includes in the request and their effect on the basic fulfillment process.*

1. Validates that the capability request contains the appropriate content for processing.
2. Returns all licenses currently served to the client device. (Reserved licenses remain reserved. Shared licenses are returned to the pool.)
3. Searches for reservations assigned to the client device and to the user sending the request from that client.
4. Determines whether desired features are specified in the request.

   If no desired features are requested, the server grants the client device whatever licenses have been reserved for it through device-based reservations and through user-based reservations. If no reservations exist, no licenses are granted.

   If a desired feature is specified, the server determines whether the specified feature is available through a reservation. If not, it determines whether any shared counts are still available. In summary, the server does the following:

   a. Checks for the device-based reserved count first.
   b. If device-based reserved count is not sufficient, checks the user-based reserved count for an additional count.
   c. If the sum of device-based and user-based reserved counts is not sufficient, checks the available shared counts to fulfill the remaining requested count.
   d. If enough reserved and shared counts are available, grants the licenses.

Example Scenarios of the Basic Process for Granting Licenses

The following scenarios illustrate instances of the basic process (described in the previous section) that the license server uses to grant licenses when reservations are involved. The sections describing the scenarios in include the following:

- Feature Allocation on the License Server
- Starting State for Each Scenario
- Scenario 1—No Device or User Reservations Available
- Scenario 2—Device Reservations But No User Reservations Available
- Scenario 3—User Reservations But No Device Reservations Available
- Scenario 4—Device and User Reservations Available

For the sake of simplicity, the feature version is omitted in these scenarios. Anytime feature F1 is mentioned, assume that its version is 1.0.

Feature Allocation on the License Server

The license server has 10 counts of feature F1:

- 3 reserved counts for device D1
- 2 reserved counts for user U1
- 5 shared counts available

Starting State for Each Scenario

The starting state for each scenario is a follows:

- Device D1 has been served 4 counts (all 3 reserved counts, 1 shared count).
- Device D2 has been served 3 counts (3 shared counts).
- User U1 currently has no served counts.
- 1 shared count remains available.

Scenario 1—No Device or User Reservations Available

When user U7 (who has no reservations) sends a capability request from device D2 (which also has no reservations), the 3 counts of F1 currently served to D2 are returned to the shared counts. (Four shared counts are now available.)

The following describes possible scenarios of what happens next:

- If the request has no desired features, no counts are served since neither D2 nor U7 has reserved counts.
- If the request specifies 4 counts of F1, the remaining 4 available shared counts are served.
- If the request specifies 5 counts of F1, no counts of F1 are sent in the response since only 4 shared counts are available.

Scenario 2—Device Reservations But No User Reservations Available

When user U7 (who has no reservations) sends a capability request from device D1, the 4 counts of F1 currently served to D1 (3 reserved counts for D1 and 1 shared count) are returned. (Two shared counts are now available.)

The following describes possible scenarios of what happens next:

- If the request has no desired features, the 3 counts of F1 reserved for D1 are served.
- If the request specifies 4 counts of F1, the 3 counts reserved for D1 plus 1 shared count are served.
- If the request specifies 6 counts of F1, the 3 counts reserved for D1 and the remaining 2 shared counts are insufficient to fulfill the request, so no counts of F1 are sent in the response.

Scenario 3—User Reservations But No Device Reservations Available

If user U1 sends a capability request from device D2 (which has no reservations), the 3 counts of F1 currently served to D2 are returned to shared counts. (Four shared counts are now available.)

The following describes possible scenarios of what happens next:

- If the request has no desired features, the 2 counts of F1 reserved for U1 are served.

- If the request specifies 4 counts of F1, the 2 counts reserved for U1 plus 2 shared counts are served.
- If the request specifies 7 counts of F1, the two counts reserved for U1 and the remaining 4 shared counts are not sufficient to fulfill the request, so no counts of F1 are sent in the response.

Scenario 4—Device and User Reservations Available

If user U1 sends a capability request from device D1, the 4 counts of F1 currently served to D1 (3 reserved and 1 shared) are returned. (Two shared counts are now available.)

The following describes possible scenarios of what happens next:

- If the request has no desired features, all reserved features—3 counts of F1 for D1 and 2 counts for U1—are served.
- If the requests specifies 4 counts of F1, the 3 counts reserved for D1 and 1 of the 2 counts reserved for U1 are served.
- If the request specifies 8 counts of F1, the 3 counts reserved for D1, the 2 counts reserved for U1, and the 2 remaining shared counts are not sufficient to fulfill the request, so no counts are sent in the response.

Reservation Limitations

Note the following about reservations:

- A license server administrator can use either reservations or named license pools on a license server. Reservations and named license pools cannot coexist alongside each other. For more information, see Named License Pools vs. Reservations.For general information about named license pools, see Allocating Licenses Using Named License Pools.
- Reservations support only concurrent features; metered features cannot be reserved.
- The license server failover configuration does not support reservations.

Online Synchronization to the Back Office

When synchronization to the back office is enabled, the FlexNet Embedded local license server automatically sends a synchronization message to the back office on a periodic basis through an active internet connection with the back office. The type of data submitted in the synchronization message depends on the value specified for the policy setting `lfs.syncTo.IncludeAll` (see Data Synchronized During Online Synchronization for more information). When the back office receives and processes the synchronization message, it sends back a synchronization acknowledgment. The synchronization acknowledgment contains status information and any error information regarding the processing. Additionally, the server log records the event, along with any related errors or warnings.

After a successful synchronization session, information about device clients that have been served licenses is accessible to the back office.

Note ▪ *This synchronization process described here is "online"—that is, it uses direct internet access as a means of transmitting data to and from the back office. As an alternative to this type of data transmission, the FlexNet*

Embedded license server supports an offline synchronization process that does not require the server to have direct internet access. See Offline Synchronization to the Back Office.

The following sections describe synchronization to the back office:

- Enablement
- Configuring the Synchronization

Enablement

Your license server's ability to synchronize to the back office is enabled by the software producer. If you need to change your license server's current policy about synchronization, contact the software producer.

Data Synchronized During Online Synchronization

Your software producer determines the type of data that is collected during online synchronization using the policy setting `lfs.syncTo.includeAll`. If you need to change your license server's current policy about the type of data that is synchronized, contact the software producer.

- `lfs.syncTo.includeAll=false`: This mode collects only the current state for each active client device at the point of synchronization and uploads this data to the back office.
- `lfs.syncTo.includeAll=true`:This mode collects all historical client actions (license distribution and metered usage on client devices, based on the server's deduction records) in the synchronization history and uploads this data to the back office as part of the synchronization.

Note that `lfs.syncTo.enabled` (not editable) must be set to `true` for online synchronization.

Configuring the Synchronization

Using the FlexNet License Server Administrator command-line tool or FlexNet License Server Manager, whichever tool is available to you, you can edit certain producer-defined settings that control synchronization:

- If using the FlexNet License Server Administrator command-line tool, see Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- If using the FlexNet License Server Manager, see "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

You can override the default for any of the editable `lfs.syncTo...` settings to control the synchronization process.

For a description of these settings and a list of those you can edit, see the appendix Reference: License Server Policy Settings.

Offline Synchronization to the Back Office

The traditional process of synchronizing license data from the FlexNet Embedded local license server to the back office relies on the license server having direct internet access to transmit data to the back office. For security reasons, such as data privacy, local license servers might have limited or no external network connections, making it difficult to synchronize data about concurrent-license distribution and metered usage on the server to

the back office. The offline synchronization feature supports the ability to download this data from the license server on to another device that has an external network connection and then transmitting the data to the back office.

The following sections describe offline synchronization to the back office:

- Configuring Synchronization Page Size
- Synchronization Tools
- Offline Synchronization Process

Configuring Synchronization Page Size

Offline synchronization uses the producer-defined `lfs.syncTo.pagesize` setting to control the maximum number of transaction records allowed in a single synchronization message sent to the back office. If the amount of accumulated data is larger than the number of records allowed in a single synchronization message, multiple messages are sent during the synchronization session until all data is communicated to the back office.

You can review and edit the current value for this setting by using the license server's FlexNet License Server Administrator command-line tool or FlexNet License Server Manager, whichever tool is available to you:

- If using the FlexNet License Server Administrator command-line tool, see Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- If using the FlexNet License Server Manager, see "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

For more information about the `lfs.syncTo.pagesize` setting itself, see Reference: License Server Policy Settings.

Synchronization Tools

The license server provides these two tools to perform the offline synchronization operations.

- "serverofflinesynctool.bat" (or "serverofflinesynctool.sh")
- "backofficeofflinesynctool.bat" (or "backofficeofflinesynctool.sh")

"serverofflinesynctool.bat" (or "serverofflinesynctool.sh")

This utility downloads the client records to be synced to the back office to a temporary storage location on (or accessible by) the device that has direct internet access to the back office. It is also used to upload and process the synchronization acknowledgment sent by the back office, purge old downloaded records, and force a download of all synchronization records when necessary.

If administrative security is enabled for the license server, the utility command (shown in the steps that follow) must include an `-authorize` option to specify the credentials needed to run the utility, as shown:

```
serverofflinesynctool.bat -authorize name password -url http://licenseServerHostName:port/api/1.0/
    ...
```

For more information about administrative security and setting up authorization credentials, see Managing Administrative Security on a Local License Server or CLS Instance in the Getting Started chapter.

"backofficeofflinesynctool.bat" (or "backofficeofflinesynctool.sh")

This utility uploads the client records to be synchronized from the temporary location to the back office. It also receives and saves the synchronization acknowledgment received from back office.

***Note** ▪ As an alternative to the backofficeofflinesynctool, you can also synchronize the records to the back office using the **Upload synchronization history** option in FlexNet Operations. For more information, see Uploading Synchronization History in the FlexNet Operations User Guide (available on the Revenera Product Documentation site).*

Offline Synchronization Process

Perform the offline synchronization process as described here.

Step 1: Download Records

The first step is to download the transaction records to be synchronized from the license server using the `serverofflinesynctool.bat` (in Windows) or `serverofflinesynctool.sh` (in Linux) tool:

```
serverofflinesynctool.bat -url http://licenseServerHostName:port/api/1.0/sync_message/offline
    -generate path
```

where:

- *`licenseServerHostName:port`* is the server name and port for the license server URL from where the records are being downloaded (default port is 7070)
- *`path`* is the path on the local machine where the records will be temporarily stored (for example, **`offline_sync`**)

The following shows an example command:

```
serverofflinesynctool.bat -url http://localhost:7070/api/1.0/sync_message/offline
    -generate offline_sync
```

Additionally, if administrative security is enabled, you must include your authorization credentials in the command. See "serverofflinesynctool.bat" (or "serverofflinesynctool.sh") for details.

This command can be run on a system with an external network connection or on the hosting license server. If you run the command on the hosting license server, the files containing the data to be synchronized need to be copied to a machine with external network connectivity.

Once the download completes, a message stating the number of transaction records downloaded is displayed:

```
OfflineSync utility started.
Sync completed for 3 device records.
```

A file with a name such as `20190112T144552.fnesync` will be created in the path you specified (for example, **`offline_sync\20190112T144552.fnesync`**)

If there are no new records to download, the message displays the following:

```
OfflineSync utility started.
No new data is available.
```

Step 2: Synchronize to the Back Office

Note ▪ *As an alternative to the backofficeofflinesynctool, you can also synchronize the records to the back office using the* ***Upload synchronization history*** *option in FlexNet Operations. For more information, see Uploading Synchronization History in the FlexNet Operations User Guide (available on the Revenera Product Documentation site).*

Next, use the `backofficeofflinesynctool.bat` (in Windows) or `backofficeofflinesynctool.sh` (in Linux) tool to synchronize the records to the back office:

```
backofficeofflinesynctool.bat -url https://siteID.flexnetoperations.com/flexnet/
    deviceservices -out filename path
```

The following describes the parameters used in the command:

- *siteID* is the specific site ID—supplied by Revenera—for the back-office URL.
- *filename* is the name and path of the file to which the synchronization acknowledgment will be written to (for example, **offline_sync\syncack.bin**).
- *path* is the path on the local machine where the transaction records to be synchronized were stored by the `serverofflinesync` tool (for example, **offline_sync**). To specify a specific file, provide the path and file name.

The following shows an example command:

```
backofficeofflinesynctool.bat -url https://flex1234-uat.flexnetoperations.com/flexnet/
    deviceservices -out offline_sync\syncack.bin offline_sync
```

A synchronization acknowledgment message is returned:

```
Successfully sent sync data and received a sync acknowledgment.
```

The synchronization acknowledgment received from the back office is written to the output file (default is `syncack.bin` in the current directory).

```
MessageType="Server sync acknowledgment"
MessageTIme="Jan 13, 2019 10:53:46 AM"
LastSyncTime="Jan 13, 2019 10:52:45 AM"
SourceIDType=String
SourceID=BACK_OFFICE
SourceIds=BACK_OFFICE
TargetID=A088B436F208
TargetIDType=Ethernet
```

Step 3: Update the Synchronization Time

The synchronization acknowledgment needs to be processed on the license server to update the last time of synchronization so that it knows that the data has been synchronized to the back office. When executing the tool, use the same *path* value used to download and synchronize the data to the back office so that the tool can remove the old synchronized data file after it processes the response. (Additionally, if administrative security is enabled, you must include your authorization credentials in the command, as described in "serverofflinesynctool.bat" (or "serverofflinesynctool.sh").)

```
serverofflinesynctool.bat -url http://licenseServerHostName:port/api/1.0/sync_ack/offline
    -process filename path
```

The following describes the parameters used in the command:

- *licenseServerHostName:port* is the server name and port for the license server URL.
- *filename* is the path and name of the acknowledgment file (for example, **offline_sync\syncack.bin**) on the local machine.
- *path* is the path on the local machine where the transaction records to be synchronized were stored by the serverofflinesync tool (for example, **offline_sync**).

The following shows an example command:

```
serverofflinesynctool.bat -url http://localhost:7070/api/1.0/sync_ack/offline -process
    offline_sync\syncack.bin offline_sync
```

The server responds with the following message:

```
OfflineSync utility started.
Purging file 20140613T105312.fnesync
```

Force a Download of All Synchronization Records

The -force option can be used with the serverofflinesync tool if you encounter an error similar to the following:

```
"Mismatched or out of order time stamp in sync message": "Expected sync time Jan 24, 2019 11:04:09
AM, got Jan 24, 2019 11:05:00 AM")
```

Checking the synchronized records in the back office reveals that those records processed after a mismatch are not synchronized. To correct the problem, you must force a download of all transaction records to account for any records missed during synchronization. The -force option allows you to start the record collection and download from the last successful synchronization time.

Task ***To force a download of all records since the last time of synchronization***

1. Process the synchronization acknowledgment on the license server (as described in Step 3: Update the Synchronization Time). This will update the records up to the point of the mismatch. Only those records that were synchronized to the back office are deleted.
2. Recover the records missing from synchronization. However, if you try to download the records again, the server thinks it has already downloaded all the records and responds with No new data is available. To remedy the problem, use the -force option with serverofflinesynctool. (If administrative security is enabled, you must also include your authorization credentials in the command, as described in "serverofflinesynctool.bat" (or "serverofflinesynctool.sh"))

   ```
   serverofflinesynctool.bat -url http://licenseServerHostName:port/api/1.0/sync_message/offline
       -generate path -force
   ```

   This will dump *all* the records since the lastSyncTime specified in the last synchronization acknowledgment.
3. Run the backofficeofflinesynctool to upload the data to the back office, or use the **Upload synchronization history** option in FlexNet Operations (see Uploading Synchronization History in the FlexNet Operations User Guide).
4. Once the data has been successfully synchronized, process the acknowledgment on the server to update the synchronization date and delete the synchronized data.

Status of Synchronization to the Back Office

You can periodically check your synchronization status to view the last time synchronization was run and the number of transaction records that need to be synchronized. This information is especially helpful when you are using offline synchronization because you can determine whether a synchronization is due based on back-office policies or general maintenance needs. For example, you might be late in reporting metered usage to the back office, causing you to be out of compliance with the producer; or you might have a large volume of client transaction records that need to be synchronized, increasing disk space requirements on the server machine.

Task ***To check your synchronization status***

Using the FlexNet License Server Administrator command-line tool, run the `-status` command, as shown in this example:

```
flexnetlsadmin.bat -server licenseServer_baseURL -status
```

(See the Using the FlexNet License Server Administrator Command-line Tool chapter for details about the `-status` command.)

The output includes the `RecordsPendingSync` and `Time elapsed since last sync` items to provide a synchronization status. (In this case, two records are pending synchronization, and the last synchronization occurred 23 hours previously.)

```
Copyright (c) 2015-2017 Flexera. All Rights Reserved.

(version) Version            : 2017.02
(buildVersion) Build Version : 198983
...

RecordsPendingSync : 2

Time elapsed since last sync: 23 hours
```

The `RecordsPendingSync` item might indicate that you have exceeded a metered-usage count or an unsynchronized-record limit. The `Time elapsed since last sync` item (the lowest unit of time for which is “hours”) might indicate that you have missed a scheduled synchronization.

Synchronization From the Back Office

Another type of synchronization is synchronization of license data *from* the back office to the FlexNet Embedded local license server. Synchronization from the back office enables a server’s license-distribution and metered-usage state to be restored from the back office after catastrophic server failure.

The following sections describe synchronization from the back office:

- Enablement
- Synchronization Process
- Viewing Restored Data
- Reviewing Synchronization Settings

Enablement

Your license server's ability to synchronize from the back office is enabled by the software producer. If you need to change your license server's current policy about synchronization, contact the software producer.

Synchronization Process

Once the license server is up and running, if trusted storage is not present, the server first sends a capability request to the back office to obtain its pool of licenses. The server then sends a *synchronization request* message to the back office to obtain the server's license-distribution and metered-usage data required to restore the deduction records on the server. (If synchronization from the back office is enabled, the server log contains an entry stating as such.)

Once the synchronization from the back office is complete, the license server can start responding to capability requests from client devices. Moreover, server's transaction records will be reflected in the back office with updated timestamps the next time synchronization to the back office is performed. These updated records indicate a successful restoration of the server's license-distribution and metered-usage state.

Viewing Restored Data

When the synchronization from the back office is complete, use your license-server administrator tool to review the restored license pool and usage data.

- If using the FlexNet License Server Administrator command-line tool, see Viewing Features Installed on the License Server and Monitoring License Distribution to Clients in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- If using the FlexNet License Server Manager, see "Devices View" and "Features View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

Reviewing Synchronization Settings

Using the FlexNet License Server Administrator command-line tool or FlexNet License Server Manager, whichever tool is available to you, you can review the producer-defined settings that control synchronization from the back office:

- If using the FlexNet License Server Administrator command-line tool, see Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- If using the FlexNet License Server Manager, see "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

For a description of these settings, see Reference: License Server Policy Settings. For the restoration of the server's license pool, see the capability-polling settings (beginning with `lfs.capability...`). For the synchronization from the back office, see the regular synchronization setting `lfs.syncFrom.enabled`.

License Server Failover

The FlexNet Embedded local license server includes support for *license server failover*. Failover involves two local license servers—the *main license server* and the *back-up license server*—that work together to ensure that licenses in the enterprise remain available for serving to client devices should the main license server fail.

The two license servers in a failover configuration must be of the same version.

In a failover configuration, the main FlexNet Embedded local license server periodically sends synchronization messages to the back-up license server and the back-up license server uses this information to mirror the state of the main server.

If a client device determines that the main license server has failed—by a failure to respond to the client's capability request, for example—it can communicate with the back-up license server, which then takes over responsibility for serving the pool of licenses until the main license server is restored, or until the configured maintenance interval (as defined by the software producer) has elapsed.

The back-up license server does not synchronize to the back office or to the main license server. Therefore, data about clients served by the back-up license server during a failover period is not stored in the back office. However, once the main license server has come back online, license-distribution data collected from that point on is once again synchronized to the back office.

The following sections describe license server failover:

- Configuring Server Failover
- Editing Failover Configuration Settings
- (Optional) Automatic Registration of the Failover Pair
- Additional Failover Considerations

Configuring Server Failover

Failover configuration is actually set up on the back-up license server. Follow these procedures to complete the configuration.

Step 1: Install and Configure the License Servers

Install both the main and back-up license servers. As part of the installation process, configure both servers with basically the same local settings, with some allowable differences. For example, you can specify different `PORT` values for the license servers. Additionally, if specifying the `BACKUP_SERVER_HOSTID` setting, do so only on the main license server (see (Optional) Automatic Registration of the Failover Pair).

For more information about installing a license server and editing its local settings file—`flexnetls.settings` on Windows or `/etc/default/flexnetls-`*`producer_name`* on Linux, see Getting Started.

Step 2: Register the Main and Backup License Servers

Using the FlexNet Operations End-User Portal, register the main license server with the back-up license server as a failover pair on the back-office server. (This registration is performed by specifying the hostid of the back-up license server on the portal's **Create Server** page for the main license server.) The hostid for each the main and the back-up license server must be of the same type.

***Note** ▪ It may be necessary to enable support for server failover in FlexNet Operations.*

For more information about registering the failover pair from the **Create Server** page for the main license, see the online help system for the FlexNet Operations End-User Portal.

This step can be replaced with an automatic registration of the failover pair through the capability request sent from the main license server. See (Optional) Automatic Registration of the Failover Pair for details.

Step 3: Start Up the License Servers

Start up the license servers. Any license rights already mapped to the failover pair through the main license server are automatically activated on the both license servers. If you need to activate licenses manually, see Step 6: Activate Licenses on the Servers.

You can now use the administrator tool provided with the license server to update policy settings, as described next in Steps 7 and 8.

Step 4: Enable Failover Support on the Back-Up Server

Update policy settings on the back-up license server, using the provided license-server administrator tool, such as the FlexNet License Server Administrator command-line tool or the FlexNet License Server Manager:

1. Set the following license server policy settings on the back-up license server:
   - `fne.syncTo.mainUri` - Set to the URI of the main license server (in the format `http://`*`licenseServerHostName`*`:`*`port`*`/fne/bin/capability`).
   - `fne.syncTo.enabled` - Set to *`true`*.
2. (Optional) Set other failover-related policy settings, such as other `fne.syncTo...` settings or the `licensing.backup.uri` and the `licensing.main.uri` settings. (Use the format `http://`*`licenseServerHostName`*`:`*`port`*`/fne/bin/capability` for URI settings.)

See Editing Failover Configuration Settings for information about editing these license-server policies.

(Optional) Step 5: Edit Producer Settings on the Main Server

Configure one or both of the following license server policy settings on the main license server to include these URIs as reference information sent in capability responses to client devices. Use the format `http://`*`licenseServerHostName`*`:`*`port`*`/fne/bin/capability` for either URI.

- `licensing.backup.uri`
- `licensing.main.uri`

See Editing Failover Configuration Settings for information about editing these license-server policies.

Step 6: Activate Licenses on the Servers

If no license rights were activated on the license servers at startup, perform these steps to provision the servers with licenses:

1. Start up both license servers.
2. Send one or more rights IDs in a capability request to activate identical license rights on both license servers via the current (main) license server. For information about activating license rights, see the appropriate information:
   - If using the FlexNet License Server Administrator command-line tool, see Activating License Rights on the Server in the Using the FlexNet License Server Administrator Command-line Tool chapter.
   - If using the FlexNet License Server Manager, see "Offline Server Updates View" in the FlexNet License Server Manager Guide chapter (available on the Revenera Product Documentation site, https://docs.revenera.com).

Step 7: Verify Failover Roles

Verify the failover role of each license server after the servers receive their license rights. (The roles are determined by the back office and included in the capability response.)

Verification can be performed by using the FlexNet License Server Administrator command-line tool (for example, `flexnetlsadmin -server licenseServer_baseURL -status`). Additionally, the primary server log will state the failover role of the given license server.

Editing Failover Configuration Settings

Using the FlexNet License Server Administrator command-line tool or FlexNet License Server Manager, whichever tool is available to you, you can edit the policy settings used to configure the main and back-up servers, as described in the previous section:

- If using the FlexNet License Server Administrator command-line tool, see Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- If using the FlexNet License Server Manager, see "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com).

For a description of these settings, see the appendix Reference: License Server Policy Settings.

(Optional) Automatic Registration of the Failover Pair

As described in Step 2: Register the Main and Backup License Servers, the license server administrator must register the main and back-up license servers as a failover pair in FlexNet Operations. One way to perform this registration is to do so manually through the FlexNet Operations End-User Portal.

Another way to perform this registration is to use the capability request sent to FlexNet Operations by the main license server at startup as a means of conveying the back-up license server's hostid to the back-office server. FlexNet Operations then uses the hostid information in the capability request to automatically register both license servers as a failover pair, thus avoiding the extra step of having to access the End-User Portal to perform the registration.

Task ***To set up this automatic registration process***

1. Shut down both the main and back-up license servers.
2. On the main license server, update the local license-server settings to include the `BACKUP_SERVER_HOSTID` setting. This value identifies the hostid of the back-up license server. It must be the same hostid type (such as "ETHERNET") as the type of hostid used by the main (current) license server. See Getting Started for details about updating local settings.
3. Start up the main license server first.

   The main license server sends a capability request containing the back-up license server's hostid to FlexNet Operations, where both servers are then registered as a failover pair.
4. Start up the back-up license server once the main license server has successfully communicated with FlexNet Operations. Both servers can now obtain license rights either through a rights ID sent in a capability request or through capability polling if licenses rights are subsequently mapped in FlexNet Operations to the registered failover pair.

   Note ▪ *You must wait until the main license server has successfully started and communicated with FlexNet Operations before starting the back-up license server. Otherwise, FlexNet Operations might assign the back-up license server to the "main server" role, generating an error when FlexNet Operations then attempts to register the actual main license server.*
5. Configure the license servers as described in Step 4: Enable Failover Support on the Back-Up Server and (Optional) Step 5: Edit Producer Settings on the Main Server.

You can view details for the main and back-up server by searching for the main license server (from the **Search Servers** page) in the FlexNet Operations End-User Portal. The **View Server** page for the main license server shows both license servers registered as a failover pair.

Additional Failover Considerations

The following lists additional considerations when enabling failover configuration:

- Since the back-up license server does not synchronize to the back office or the main license server, data about clients served by the back-up server during a failover period is permanently missing in the back office. Once the main server has come back online, the new license-distribution data can be synchronized to the back office.
- Metered-usage data is currently not supported in a failover scenario.
- The failover process might be incompatible with the type of FlexNet Embedded licensing with which your purchased product is enabled and secured. Contact the producer for details.
- Failover functionality is dependent on the clocks of the main license server and back-up license server being accurate, synchronized with each other, and in the same time zone. If any of these requirements are not met, unpredictable behavior can occur.
- The failover configuration does not support reservations.

- The failover configuration does not support named license pools. During a failover period, feature counts will only be distributed from the default license pool.

Outgoing HTTPS

The FlexNet Embedded local license server is capable of HTTPS communication with FlexNet Operations—that is, *outgoing* HTTPS.

For a secure connection, a truststore file containing root and intermediate certificates is required.

- If the back-office server certificate is signed by a known certificate authority (as is the case for Revenera-hosted back offices), then little or no configuration is needed on the server. See Default Server Configuration.
- If the producer has provided their own truststore, or has changed the password found in the Java `cacerts` file, you need to perform some configuration steps. See Server Configuration When Another Certificate Is Used for instructions. Typically, a producer provides their own truststore file when the back office is an "on-premises" version of FlexNet Operations and the server certificate has been signed by an unknown (private) certificate authority.

Default Server Configuration

In most scenarios, HTTPS will work "out of the box". A secure connection to the back office requires access to the appropriate root and intermediate certificates for connection validation. The default behavior is to use the truststore shipped with the Java runtime environment (JRE). This is the "cacerts" file found in the `lib/security` directory of the JRE installation. The file is encrypted, and is installed with a password of "changeit".

No additional configuration should be needed for Revenera-hosted FlexNet Operations instances and instances where FlexNet Operations is hosted on AWS.

If any SSL-related errors appear in the server log, refer to the following section, Server Configuration When Another Certificate Is Used.

Server Configuration When Another Certificate Is Used

You need to override the default server configuration if either of these conditions are true:

- The producer has changed the password for `cacerts`.
- You need to connect to an "on-premises" installation of FlexNet Operations that is using a private certificate authority whose root certificates are not found in the `cacerts` file.

As always, consult the producer if you need to determine the name and location of the truststore file.

There are different ways to override the default server configuration, depending on whether you are newly installing the local license server or whether you have an older installation.

New Installations

Update the `local-configuration.yaml` file to override the default server configuration. On Linux systems, this file is generated in the `/opt/flexnetls/producer` directory during installation. On Windows, it is located in the same directory as `flexnetls.jar`.

You might need to change the password or the path to the truststore file, or both.

Edit the following settings in `local-configuration.yaml` so that they look similar to this:

```
https-out:
  # Set to true to enable
  enabled: true
  # Path to truststore containing server certificate.
  truststore-path: path-to-your-truststore
  # Truststore password. You can obfuscate this with java -jar flexnetls.jar -password your-
  password-here
  truststore-password: your-password-here
  # Switch off if you're having host validation problems (not recommended)
  host-verify: true
  # Set to true if you're using self-signed certificates (not recommended)
  self-signed: false
```

The password is not required to be stored in plain text; you can obfuscate it first. The following shows an example session to obfuscate the password:

```
$ java -jar flexnetls.jar --password=abracadabra
abracadabra => OBF:1ri71v1r1v2n1ri71shq1ri71shs1ri71v1r1v2n1ri7
```

Be sure to set the `truststore-password` value to the entire obfuscated string, including the `OBF:` prefix.

For detailed instructions about editing the `local-configuration.yaml` file, see Edit “local-configuration.yaml” (Windows) and Edit “local-configuration.yaml” (Linux).

Existing Installations

If you are using an older version of the local license server, which provides access to the truststore file using a “client” configuration file, you can do either of the following:

- Update the existing “client” configuration file to use the new values. See Update the “client” Configuration File.
- Transfer the HTTPS values from the “client” configuration file to `local-configuration.yaml`. See Transferring HTTPS Values from “client” Configuration File to YAML File.

Update the “client” Configuration File

Task ***To update the “client” configuration file***

1. Update the following values in the existing “client” configuration file:

   ```
   truststore.path=path-to-client-certificate
   truststore.password=password
   ```

2. If required, use the appropriate step to tell the license server where the “client” configuration file is located:

- In Windows, edit the `flexnetls.settings` file to include this option to specify the path to the configuration file:

  ```
  HTTPS_CLIENT_CONFIG=path-to-configuration-file
  ```

  In Linux, edit following variable in the `/etc/default/flexnetls-producer_name` file, replacing the comment with the path to the configuration file:

  ```
  HTTPS_CLIENT_FILE=    #empty, can replace by -https-client-configuration path
  ```

See Editing the Local Settings Post-Installation in the Getting Started chapter for complete instructions on editing this local settings file.

Transferring HTTPS Values from "client" Configuration File to YAML File

Instead of using the "client" configuration file, you can use the `https-out` setting in `local-configuration.yaml`. The following table shows how to transfer the values from the "client" configuration file to `local-configuration.yaml`. Note that the settings `truststore-path` and `truststore-password` in `local-configuration.yaml` use a hyphen instead of a dot.

Table 4-8 ▪

"client" Configuration File Content	local-configuration.yaml File Content
`truststore.path=path-to-client-certificate` `truststore.password=password`	`https-out:` `  # Set to true to enable` `  enabled: true` `  # Path to truststore containing server certificate.` `  truststore-path: path-to-your-truststore` `  # Truststore password. You can obfuscate this with java -jar flexnetls.jar -password your-password-here` `  truststore-password: your-password-here` `  # Switch off if you're having host validation problems (not recommended)` `  host-verify: true` `  # Set to true if you're using self-signed certificates (not recommended)` `  self-signed: false`

Incoming HTTPS

The following sections help you set up the license server for incoming HTTPS and configure the license server to use a self-signed certificate:

- Configure HTTPS Connection for Incoming License Server Communication
- Self-Signed Certificate for the Local License Server

Configure HTTPS Connection for Incoming License Server Communication

In most cases, communication between client devices and the FlexNet Embedded local license server need not be secure. However, in deployments where Internet security is required, the local license server can be configured to support exchanges with the client devices over a secure HTTPS connection.

Best practice is to offload this HTTPS protocol work to a separate device that specializes in this type of processing. However, if this practice is not feasible or desirable, perform these steps to enable the license server to support incoming HTTPS connections.

Step 1: Obtain Certificate

You will require a “server” certificate from a certificate authority (CA) such as Verisign. This certificate must be in Java keystore format (not a PEM text file) and secured by a password. Your certificate provider will have instructions on how to request it in this format (or convert to it.)

Step 2: Enable Access to “server” Certificate

Access to the “server” certificate can be provided in different ways, depending on whether you are newly installing the local license server or whether you have an existing installation.

New Installations

For new installations of the local license server, specify the location of the keystore file holding the “server” certificate in the `local-configuration.yaml` file using the `https-in` parameter. On Linux systems, the `local-configuration.yaml` file is generated in the `/opt/flexnetls/producer` directory during installation. On Windows, it is located in the same directory as `flexnetls.jar`.

You might need to change the password or the path to the truststore file, or both.

Edit the following settings in `local-configuration.yaml` so that they look similar to this:

```
https-in:
  # Set to true to enable
  enabled: true
  # HTTPS listening port
  port: 1443
  # Path to keystore
  keystore-path: path-to-your-keystore
  # Keystore password. You can obfuscate this with
  java -jar flexnetls.jar -password your-password-here
  keystore-password: your-password-here
```

Note the following:

- The password is not required to be stored in plain text; you can obfuscate it first. The following shows an example session to obfuscate the password:

  ```
  $ java -jar flexnetls.jar --password=abracadabra
  abracadabra => OBF:1ri71v1r1v2n1ri71shq1ri71shs1ri71v1r1v2n1ri7
  ```

Be sure to set the `keystore-password` value to the entire obfuscated string, including the `OBF:` prefix.

- The standard HTTPS port is 443 (as defined by the HTTPS protocol)
  - Best practice on Linux is to use a port number greater than 1024 to avoid issues with possibly having to obtain extra privileges to use a lesser port number. Should port 443 be required at a customer enterprise, you can always use iptables to redirect the port from 443.
  - Ensure that a firewall is not blocking the license server.

For detailed instructions about editing the `local-configuration.yaml` file, see Edit "local-configuration.yaml" (Windows) and Edit "local-configuration.yaml" (Linux).

Existing Installations

If you are using an existing installation of the local license server, which provides access to the "server" certificate using a "server" configuration file, you can do either of the following:

- Update the existing "server" configuration file to use the new values. See Update the "server" Configuration File.
- Transfer the HTTPS values from the "client" configuration file to `local-configuration.yaml`. See Transferring HTTPS Values from "server" Configuration File to YAML.

Update the "server" Configuration File

Task ***To update the "server" configuration file***

1. Update the following values in the existing "server" configuration file:

   ```
   keystore.path=path-to-server-certificate
   keystore.password=password
   https.port=1443
   ```

2. If required, use the appropriate step to tell the license server where the "server" configuration file is located:
   - In Windows, edit the `flexnetls.settings` file to include this option to set the path to the configuration file:

     ```
     HTTPS_SERVER_CONFIG=path-to-configuration-file
     ```

   - In Linux, edit following variable in the `/etc/default/flexnetls-producer_name` file, replacing the comment with the path to the configuration file:

     ```
     HTTPS_SERVER_FILE=    #empty, can replace by -https-server-configuration path
     ```

See Editing the Local Settings Post-Installation in the Getting Started chapter for complete instructions on editing the local settings file.

Transferring HTTPS Values from "server" Configuration File to YAML

Instead of using the "server" configuration file, you can use the `https-in` setting in `local-configuration.yaml`. The following table shows how to transfer the values from the "server" configuration file to `local-configuration.yaml`. Note that the settings `keystore-path` and `keystore-password` in `local-configuration.yaml` use a hyphen instead of a dot.

Table 4-9 ▪

"client" Configuration File Content	local-configuration.yaml File Content
`keystore.path=path-to-server-certificate` `keystore.password=password` `https.port=1443`	`https-in:` `  # Set to true to enable` `  enabled: true` `  # HTTPS listening port` `  port: 1443` `  # Path to keystore` `  keystore-path: path-to-your-keystore` `  # Keystore password. You can obfuscate this with` `  java -jar flexnetls.jar -password your-password-here` `  keystore-password: your-password-here`

Step 3: Define Scope of HTTPS Communications

At this point, you should have a license server that is operating in both HTTP and HTTPS modes. The final step is to define the scope of HTTPS usage for your license server. For example, maybe you want to run the FlexNet Embedded local license server entirely in HTTPS mode. To enable this scope, switch off the HTTP listening port by changing the `PORT` value to zero (0) in the `local-configuration.yaml` file (Windows and Linux), the `flexnetls.settings` file (in Windows) or the `/etc/default/flexnetls-`*`producer_name`* file (in Linux). See Editing the Local Settings Post-Installation in the Getting Started chapter for complete instructions on editing the local settings file.

Self-Signed Certificate for the Local License Server

When evaluating the local license server, you may not wish to acquire a proper server certificate. It's possible to use a self-signed certificate instead, although this is not recommended for production use.

This section explains how to set up the server to communicate with the FlexNet License Server Administrator command-line tool (flexnetlsadmin) using a self-signed certificate.

The following procedure uses the **keytool** utility to generate the certificate; other tools are available.

Task ***To configure communication between the local license server and flexnetlsadmin using a self-signed certificate***

1. Use a command similar to the following to create a self-signed PKCS#12 certificate:

```
# HTTPS server mode
$ keytool \
```

```
-genkeypair \
-ext san=dns:yourhost.yourcompany.com \
-storetype PKCS12 -keystore self-signed.p12 \
-storepass self-signed-password \
-alias self-signed \
-keyalg RSA \
-keysize 4096 \
-validity 360 \
-dname "C=,S=,L=,O=,OU=,CN=yourhost.yourcompany.com"
```

2. Specify the certificate that you just created in `local-configuration.yaml`:

```
# HTTPS server mode
https-in:

  # Set to true to enable
  enabled: true

  # HTTPS listening port
  port: 1443

  # Path to keystore
  keystore-path: self-signed.p12

  # Keystore password. You can obfuscate this with
  java -jar flexnetls.jar -password your-password-here
  keystore-password: self-signed-password
```

3. Export the certificate in PEM format for flexnetlsadmin:

```
$ keytool -exportcert -alias self-signed -keystore self-signed.p12 -rfc -file self-signed.pem
```

4. Start the local license server.

5. Use a command such as the following to test the communication between the server and flexnetlsadmin:

```
$ flexnetls-admin --partitions -custom-trust self-signed.pem
[ {
  "id" : 1,
  "name" : "default",
  "activeFeatureSlices" : [ ],
  "lastModified" : "2023-02-13T11:31:53.622Z"} ]
```

Proxy Support for Communication with the Back Office

For security purposes, your corporate network might require a networking proxy as an intermediary between your internal systems and the Internet. To address this requirement, the FlexNet local license server supports communication with FlexNet Operations through an HTTP proxy. The local license server depends on Java Runtime to provide networking services.

Note ▪ *The FlexNet Embedded local license server supports both authenticating and non-authenticating HTTP proxies.*

The following sections help you set up the license server to use this proxy:

Configuring the License Server for Proxy Support

Use this information to configure the license server for HTTP communications with the back office through a networking proxy:

Proxy Configuration Basics

To configure the license server to use a networking proxy when communicating with the back office, you must define a set of proxy parameters in the server's local settings file on the server (`flexnetls.settings` on Windows or `/etc/default/flexnetls-`*`producer_name`* on Linux). These parameters are passed to the Java Runtime system to identify the HTTP proxy.

You specify these parameters for the `EXTRA_SYSPROPERTIES` setting, which you must "uncomment" if it is currently "commented" in this file.

For complete instructions on editing the local settings file, refer to the Configuring, Installing, and Starting the Local License Server and Editing the Local Settings Post-Installation sections in the Getting Started chapter.

Supported Proxy Parameters

The following proxy parameters are supported:

- `http.proxyHost`
- `http.proxyPort`
- `http.proxyUser`
- `http.proxyPassword`

Consider the following:

- If not included, the `http.proxyPort` parameter defaults to `80`.
- The `http.proxyUser` and `http.proxyPassword` parameters are optional. They are required only if the proxy requires authentication.
- Best practice is to use an obfuscated password instead of a plain-text value for `http.proxyPassword`. See Obfuscating the Proxy Password for encryption details.

Parameter Format

Follow these instructions when adding the proxy parameters to the EXTRA_SYSPROPERTIES setting in the local settings file:

- Each parameter specified for EXTRA_SYSPROPERTIES must use the following Java system syntax:

  `-D`*`property=value`*

- The parameters are separated from each other with a space, and the entire set of parameters are enclosed in double quotations. The following shows an example EXTRA_SYSPROPERTIES setting with proxy parameters specified:

  ```
  EXTRA_SYSPROPERTIES="-Dhttp.proxyHost=10.90.3.133 -Dhttp.proxyPort=3128 -Dhttp.proxyUser=user1a
  -Dhttp.proxyPassword=user1apwd35"
  ```

  Even if you specify only a single parameter, it must be enclosed in double quotations:

  ```
  EXTRA_SYSPROPERTIES="-Dhttp.proxyHost=10.90.3.133"
  ```

Obfuscating the Proxy Password

Best practice is to obfuscate the proxy password and use this encrypted value for the http.proxyPassword parameter.

The following is an example session showing both the command used to obfuscate a plain-text password (in this case, user1apwd35) and the resulting obfuscated string:

```
$ java -jar flexnetls.jar --password=user1apwd35
user1apwd35 => OBF:1ri71v1r1v2n1ri71shq1ri71shs1ri71v1r1v2n1ri7
```

Use the entire obfuscated string, including its OBF: prefix, for the http.proxyPassword value, as shown in this example:

```
-Dhttp.proxyPassword=OBF:1ri71v1r1v2n1ri71shq1ri71shs1ri71v1r1v2n1ri7
```

Trusted Storage Backup and Restoration

If the producer has enabled your local license server to perform backups of trusted storage, you can restore the last backup copy, should trusted storage become corrupted, without having to contact FlexNet Operations. The following sections describe the trusted-storage backup and restoration process:

- Overview of the Backup Process
- Running a Trusted Storage Restoration

Overview of the Backup Process

You can restore trusted storage from a backup copy only if the producer has enabled trusted-storage backups to occur on the license server. If you need information about enabling these backups, contact the producer.

The following provides an overview of the backup process for trusted storage on the license server:

- **One.** The license server automatically initiates a trusted-storage backup whenever the following events occur:
  - A capability response from the back office is processed
  - A quantity of capability requests from FlexNet Embedded clients have been processed (based on producer-defined thresholds for the license server)
- **Two.** Once the backup is initiated, the license server suspends the processing of capability responses and requests until the backup completes.
- **Three.** If the backup is successful, the backed-up trusted storage data is stored in the file `tsBackup/flexnetls_licenses.mv.db.ts`, located in the directory containing trusted storage. (A failed backup can be an indicator that trusted storage is corrupt.)

  The license server maintains only one copy of the backed-up data at any given time.

Running a Trusted Storage Restoration

If trusted storage is determined to be corrupt, you can restore the last successful backup of trusted storage.

Task ***To restore trusted storage from the backup***

1. Stop the license server.
2. Use the `-restore-database` or `-restore-service-database` argument with the command that runs the license server script to restore trusted storage.

   The *path* value used in this command is the path to the trusted-storage backup file `flexnetls_licenses.mv.db.ts`, located under `tsBackup` in the trusted storage directory. Note that the explicit path values listed in this step use the default value (**`${base.dir}`**) for the license server policy `server.trustedStorageDir`, which defines the trusted storage directory. You might need to adjust the path based on this policy definition on your license server. (See Reference: License Server Policy Settings.)

   If more than one user log-in will be doing a restore, `server.trustedStorageDir` should be set to a path other than **`${base.dir}`**. See the producer for updating this policy.

   On Windows

   - In console mode, enter:

     `flexnetls.bat -restore-database` ***`path`***

     where *path* is `C:\Users\`*`user_name`*`\flexnetls\`*`producer_name`*`\tsBackup\flexnetls_licenses.mv.db.ts`.
   - As a service, enter:

     `flexnetls -restore-service-database` ***`path`***

     where *path* is `C:\Windows\ServiceProfiles\NetworkService\flexnetls\`*`producer_name`*`\tsBackup\flexnetls_licenses.mv.db.ts`.

   On Linux

   - In console mode, enter:

```
./flexnetls -restore-database path
```

where *path* is `/var/opt/flexnetls/producer_name/tsBackup/flexnetls_licenses.mv.db.ts`.

- As a service (Systemd), use the same command as used for console mode. Note that the database can be restored only if a custom installation location is defined.

3. Restart the license server.

Best practice is *never* to delete any files in directory containing the corrupted trusted storage; allow the restoration process handle the fix. However, if for some reason you need to remove the corrupted data, delete only `flexnetls_licenses.mv.db`.

One basic method you can use to verify that trusted storage has been restored to its previous uncorrupted state is to check the feature counts in trusted storage, using the license server administrator tool provided by the producer.

Public Key Upload

Depending on the type of FlexNet Embedded licensing with which your purchased product is enabled, you might be required to upload a public key from the client to the license server. The producer will inform you whether this upload is necessary and will provide either the public key itself or the instructions on how to obtain it. You can upload this key using the FlexNet Embedded License Server Administrator command-line tool or the FlexNet License Server Manager or a tool or method provided by the producer. The producer will direct you as to which means to use.

In general, the public key is an RSA key (2048-bit DER-encoded) in an octet-stream format. You must have license-server administrator credentials to upload it.

5

Managing a CLS Instance

If you are using a CLS instance (a Cloud Licensing Service license server), which is hosted in the Cloud, you must use the FlexNet Operations End-User Portal to manage the server. For management instructions, you can access the portal's help system, or refer to this chapter, which highlights important license-server administration procedures extracted from the portal's help system.

For more information about the CLS instance and its relationship to the FlexNet Embedded local license server, see Local License Servers vs. License Servers Hosted in the Cloud in the Introduction chapter.

This chapter covers the following procedures for managing the CLS instance:

- Attributes of the CLS Instance
- Searching for a CLS Instance
- Working with CLS Instances
- Creating a CLS Instance
- Managing Administrative Security

Attributes of the CLS Instance

The following attributes are used when searching for, creating, and managing CLS instance.

Table 5-1 ▪ CLS Instance Attributes

Attribute	Description
Deployment	The deployment type for the license server. For a CLS instance, this is **Cloud**.

Table 5-1 ▪ CLS Instance Attributes (cont.)

Attribute	Description
License Server ID	The unique identity for the license server. License Server ID is not shown when creating license servers with the **Cloud** deployment type. In this case, FlexNet Operations automatically generates a License Server ID. However, you can see this attribute on the **View Server** page for the CLS instance.
Auto Provisioned	Indicates whether the producer created the CLS instance through the auto-provision feature in FlexNet Operations.
Model	The device model used to create the CLS instance.
Site Name	A label used to distinguish between different license servers if your organization has multiple CLS instances.
Organization	The name of your organization specified as Organization ID (or Owner ID).
Server Status	The status of the license server as either ACTIVE, RETURNED, or OBSOLETE.

Searching for a CLS Instance

An existing CLS instance is often most easily located by performing a quick search based on one or more attributes of the license server.

Task ***To search for a CLS instance***

1. Click **Devices > Devices**. This opens the **Devices** page.
2. (Optional) To filter the devices list to show only CLS instances:
   a. On the **Devices** page, hover the cursor over the  filter icon to the display the filter selection list.
   b. On the selection list, select **Cloud License Servers** for the device **Type**, and select one or more **Status** types.
   c. Click **Apply** to filter the devices according to your selections.
3. Perform a simple or advanced search for the license server.

   Simple search

   a. On the **Devices** page, click the drop-down list next to the filter icon, and select the search criterion.
   b. Type the search string in the text field next to the drop-down.
   c. Click **Search**.

   Advanced search

a. On the **Devices** page, click the **+** icon (next to the **Search** button) to open the Advanced Search window.

b. Type the search criterion in the attribute text fields.

c. Click **Search**.

When you select a CLS instance from the **Devices** list, its **View Server** page opens. From this page, you can view the server's attributes and access options to perform server management activities—such as viewing server history or moving a server. See Working with CLS Instances for further information.

Working with CLS Instances

From the **View Server** page for a CLS instance, you can view the server's attributes or access options to manage the server. The following topics address the server-management activities you can perform.

For instructions on accessing a license server's **View Server** page, see the previous section Searching for a CLS Instance.

Table 5-2 ▪ Overview of CLS Management Tasks

Topic	Description
View the History of a License Server	Shows how to view a history of the license server's transactions.
View Served Clients of a CLS Instance	Shows how to view the clients currently served by the license server.
Map Entitlements to a CLS Instance	Explains how to map add-ons to the license server.
Remove Licenses from a CLS Instance	Explains how to remove add-ons from the license server.
Move a CLS Instance	Describes how to change the owner of the license server.

View the History of a License Server

You can view the history of a specific CLS instance through its **View Server** page. The **Server History** page lists all the events that have occurred in the server's history—from its creation, to ownership changes, to activation requests.

Task ***To view the history of a CLS instance***

From the **View Server** page for the CLS instance, click **View** > **View History**.

The End-User Portal opens the **Server History** page.

View Served Clients of a CLS Instance

Through the **View Server** page for the CLS instance, you can view the client devices currently served by the instance.

Task ***To view the served devices of a license server***

From the **View Server** page for the CLS instance, click **View** > **View Served Devices**.

The End-User Portal opens the **Devices** page, showing the a list of client devices currently served by the CLS instance.

Map Entitlements to a CLS Instance

CLS instances are often provisioned with pre-built capabilities. However, these servers also permit the addition of new features or more capacity via additional entitlements.

When the current owner of a CLS instance is entitled to additional capabilities, you can manually map these entitlements to the server through the **View Server** page.

Task ***To map an entitlement***

1. From the **View Server** page for the CLS instance, click **Action** > **Map Entitlements**. The End-User Portal opens the **Map Entitlements** page. The table on this page lists the entitlements that can be mapped to the current CLS instance.
2. In the table, enter a count in **Qty to Add** to specify the quantity for each entitlement you want to map.
3. (Optional) For any entitlement that is enabled, you can override the its **Expiration**, as shown in the table, with a shorter-term expiration. Click the pencil icon next to the expiration date, and select a new date from the displayed calendar. (After the change is saved, the new expiration appears in the **Licenses** table on the **View Server** page and is denoted by an asterisk in the table on the device's **Map Entitlements** page.)
4. Click **Save**.

The End-User Portal maps the entitlements to the current CLS instance and returns to server's **View Server** page.

Remove Licenses from a CLS Instance

Licenses already mapped to a CLS instance can be manually removed through its **View Server** page.

Note ▪ *Metered-model licenses cannot be removed.*

Task ***To remove licenses***

1. From the **View Server** page for the CLS instance, click **Action** > **Remove Licenses**. The End-User Portal opens the **Remove Entitlements** page. The **Licenses on Device** table on this page lists licenses that can be removed from the current server.
2. In the **Licenses on Device** table, enter a count in **Qty to Remove** for those licenses you want to remove.
3. Click **Save**.

The End-User Portal removes the specified license counts for the current CLS instance and returns to the server's **View Server** page.

Move a CLS Instance

A CLS instance that has not yet been assigned to a customer organization can be moved from your organization to a related organization. For example, the parent organization can move the license server from one of its child organizations (a regional division, for instance) to another child organization.

When a CLS instance is moved, any entitlements that belong to that license server are moved as well. The result of such a move on the receiving organization is that new entitlements are added or existing entitlements that match moved entitlements are incremented.

This action is preformed through the **View Server** page for the CLS instance.

Task ***To move a license server***

1. From the **View Server** page for the CLS instance, click **Action** > **Move Server**. The End-User Portal opens the **Move Server** page.
2. For **To Organization**, select the organization to receive the CLS instance.
3. Click **Submit**.

The End-User Portal changes the owner of the license server and returns to the server's **View Server** page.

Creating a CLS Instance

In some cases, if your enterprise is large, it might be desirable to have multiple CLS instances. You can define new CLS instances in the End-User Portal. And, once defined, a CLS instance can submit its own capability requests to FlexNet Operations and serve its own licenses to served clients.

For descriptions of the attributes used to create a CLS instance, see Attributes of the CLS Instance.

Task ***To create a CLS instance***

1. From the End-User Portal menu bar, click **Devices** > **Create Device**. The **Device | New Device** page opens.
2. Type a device **Name**.

3. Select the **Runs license server?** option. For **Deployment**, select **Cloud**.
4. Optionally, select a server **Model** from the drop-down list.
5. Optionally, select an **Organization** from the drop-down list. The organization identifies the customer or partner organization of the server.
6. Optionally, type a value for **Site Name**.
7. Click **Save**.

The End-User Portal creates the new CLS instance and opens its **View Server** page.

Managing Administrative Security

If administrative security is enabled for your CLS instance, a default license-server administrator account is created for you when the CLS instance is created. This account gives you access to a full range of administrative functionality on the CLS instance, including privileges to create and manage other enterprise user accounts. For details, see Managing Administrative Security on a Local License Server or CLS Instance in the Getting Started chapter.

This section highlights basic tasks that you perform as the license server administrator:

- Set Your Administrator Password
- Create and Manage Other User Accounts
- Manage the License Server

Set Your Administrator Password

You are responsible for setting the password for your default administrator account before you access the CLS instance. (The default administrator name for this account is “admin”, which is case-sensitive.)

Task ***To set your password***

1. From the **View Server** page for the CLS instance, click **Action** > **Set Password**.
2. Provide your password and re-enter it to confirm. (The password must meet the criteria specified in Credential Requirements in the Getting Started chapter.)

Create and Manage Other User Accounts

As a license server administrator, you can use your credentials to create other user accounts—either for enterprise users or for another administrator. You can perform these operations using the FlexNet License Server Administrator command-line tool (see Using the FlexNet License Server Administrator Command-line Tool) or the custom administrative tool provided by the producer.

Manage the License Server

Use the FlexNet License Server Administrator command-line tool or the producer's custom administrative tool to perform other administrative tasks such as creating and managing reservations, suspending the server, and querying features and information about licenses served to FlexNet Embedded clients. For instructions, see Using the FlexNet License Server Administrator Command-line Tool or the producer's instructions.

A

Reference: License Server Policy Settings

Use the following chart as reference to the license server policy settings provided by the producer to control the various operations that your local license server can perform, such as synchronization to and from the back office, licensing distribution, logging, license server failover, and others.

The **Editable?** column indicates whether you as the license server administrator can update the given policy.

***Note** ▪ These policies are of interest to the administrator of a local license server. The administrator of a CLS instance cannot edit license server policies.*

For more information about the process of overriding settings, go to the appropriate section:

- Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.
- "License Server Properties View" in the FlexNet License Server Manager Guide (available on the Revenera Product Documentation site, https://docs.revenera.com). (As a reference, the chart shows the field name equivalents used in FlexNet License Server Manager for various policy settings in `producer-settings.xml`.)

Table A-1 ▪ License Server Policy Settings

Setting	Description	Edit-able?
database.backup-enabled	The property that determines whether the license server takes a backup of trusted storage at given times and stores it on the server. Should trusted storage become corrupt, the license server administrator can then restore it from the backup without contacting the back office. The default is `false`. See Trusted Storage Backup and Restoration for details.  ***Note** ▪ Note that regular trusted-storage backups can negatively affect the license server performance.*	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
Licensing Policies		
licensing.allowDuplicateClients	Changes the logic for processing capability requests. (Default is `false`.) When set to `true`, licensing behavior can be configured at the license-model level using the **Multi-Access Licensing: Allow duplicate client checkouts?** option in FlexNet Operations. If configured, the license-model setting takes precedence over this policy. When set to `false`, all features use conventional licensing behavior regardless of any license-model configuration.  ***Tip ▪*** *To use a combination of conventional and multi-access licensing across different features, set this policy to `true` and configure the desired behavior in each license model.*	No
licensing.clientExpiryTimer	The frequency of checks for expired features on clients. (An expired feature is one whose borrow interval has expired or that has reached its expiration date as defined in the back office.) Expired features found during a given check are returned to the license-server feature pool. The frequency value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. The default frequency for these checks is `2s`. Consider increasing this value if the current frequency is interfering with capability-request processing or with overall throughput, particularly when features have large borrow intervals. (The minimum value is `1s`.)	Yes
licensing.dropClientEnforcedDelay	The delay that is enforced between client deletion requests. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. This setting can also be used to disallow the deletion of clients; in this case, set the value `DROP_CLIENT_DISALLOWED`. (Default is `0s`, meaning deleting client records is allowed and there is no enforced delay between deletions.)	No
licensing.responseLifetime	The lifetime of a served-license response on the client. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. If this value is `0` (zero), the response has an unlimited lifetime. (Default is `1d`.)	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
licensing.allowVirtualClients	The property that determines whether virtual client devices are allowed to obtain licenses. Default is `true`.	No
licensing.allowVirtualServer	The property that determines whether the license server is allowed to run on a virtual host. Default is `true`.	No
licensing.defaultBorrowGranularity	The time unit to which the borrow interval used by the license server rounds up. Valid values include `day`, `hour`, `minute`, or `second`. (The default is `second`.) For example, if the borrow interval (which is always expressed in seconds) is 60 seconds, and the borrow granularity is `day`, then a license issued at 5:05:01 PM expires at 11:59:59 PM—the borrow interval (5:06:01 PM) rounded to the *end of the nearest day*. Likewise, if granularity is `minute`, expiration is at 5:06:59 PM. If the granularity is `second`, expiration is 5:06:01 PM. This setting is used for those client devices that do not specify one.  ***Note ▪*** *For FlexNet Embedded client SDKs released before version 4.0, the granularity is always "day", regardless of this setting.*	No
licensing.borrowIntervalOverridesVersion	Changes the rules that the license server applies when granting feature requests. When set to `false`, the sorting logic applies the following rules, in order, when fulfilling capability requests: 1. **Feature version**—Prioritizing lower versions over higher versions. 2. **Grace period status**—Non-grace period features are preferred over those in a grace period. 3. **Borrow interval**—Features with the shortest expiry date that also satisfy the requested borrow interval. When set to `true`, the license server first allocates features that match the requested borrow interval, irrespective of the specified version. The sorting logic applies the following rules: 1. **Borrow interval**—Features with the shortest expiry date that also satisfy the requested borrow interval are given top priority. 2. **Grace period status**—Non-grace period features are preferred over those in a grace period. 3. **Feature version**—Prioritizing lower versions over higher versions. (Default is `false`.)	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
licensing.borrowInterval	The borrow interval for served licenses. This *server borrow interval* is only considered if the back office does not specify a borrow interval for a feature within the license model. The current version of FlexNet Operations mandates that a borrow interval (also referred to as the *feature borrow interval*) be specified for new license models. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `1w`.) For information on how to determine the effective borrow interval, see licensing.borrowIntervalMax.	No
licensing.borrowIntervalMax	Restricts the borrow period of the clients. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. Default is `NOT_CONFIGURED` (`0`). This is also referred to as the *admin borrow interval*. The following helps you determine the effective borrow interval. ● If the feature borrow interval has been set in the back office, the borrow interval is the lowest of the following values: ● feature borrow interval (set in the back office) ● client borrow interval (set in a client capability request) ● admin borrow interval (set using `licensing.borrowIntervalMax`) ● If the feature borrow interval has **not** been set in the back office, the borrow interval is the lowest of the following values: ● server borrow interval (defined in `producer-settings.xml` by the property `licensing.borrowInterval`) ● client borrow interval (set in a client capability request) ● admin borrow interval (set using `licensing.borrowIntervalMax`) A feature's current borrow expiration can never exceed the final expiration time for that feature. In addition, a borrow-interval granularity may be applied to the effective borrow interval. This parameter cannot be used for metered features.	Yes

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
licensing.renewInterval	The default renew interval is set as a percentage of the effective borrow interval. This value specifies how often—if ever—the client may attempt to recontact the local license server. Successful contact extends the expiration based on the effective borrow period (in other words, the timer for the effective borrow interval is restarted). If set to zero, the renew interval is at client discretion. (Default is `15`.)  ***Important ▪*** *This specification by itself does not lead to enforcement. The client-side APIs must extract this value from the license server capability response and take appropriate action.* For information on how to determine the effective borrow interval, see licensing.borrowIntervalMax.	No
licensing.hostIdValidation Interval	The frequency with which the license server validates that its host ID has not changed. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If this value is `0` (zero), validation is disabled. Default is `2m`.	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
licensing.enableBuiltinHostId	The property that enables producers to configure the local license server to pick the built-in Ethernet address as hostid. Set `licensing.enableBuiltinHostId=true` if the first built-in Ethernet address should be selected as hostid, otherwise set `licensing.enableBuiltinHostId=false`. The following settings override the `licensing.enableBuiltinHostId` property: ● **server.publisherDefinedHostId.policy**—If a producer-defined hostid type is set using the `server.publisherDefinedHostId.policy` property, the `licensing.enableBuiltinHostId` property is ignored. ● **ACTIVE_HOSTID**—If ACTIVE_HOSTID is set either through the REST API or using a configuration file during server startup, the `licensing.enableBuiltinHostId` property is ignored. **Limitations** In the following scenarios, setting the built-in Ethernet address as hostid using `licensing.enableBuiltinHostId` is likely to fail: ● If a user's machine is set to use a randomized wireless MAC address. ● If the option to automatically disable the wireless network connection when an Ethernet cable is connected is enabled. Therefore, if the Ethernet address is to be used as hostid, the relevant configurations must be disabled on the user's machine. (Default is false.)  ***Tip ▪*** *If you want to change the built-in hostid that is stored in the license server's database, you can call the REST endpoint* ***/hostids/selectedbuiltin*** *and delete the selected built-in hostid from the database. This forces the license server to fetch the first built-in hostid from the dynamic hostid list, instead of retrieving it from its database. For more information, see Deleting the Selected Built-in Hostid.*	No
licensing.defaultTimeZone	Defines the time zone that will be applied when determining a feature's expiry date and start date. Valid values: ● UTC— If UTC is set, a feature's start date is the start of the specified day in Coordinated Universal Time (UTC). Equally, a feature will expire at the end of the day of the configured expiry date in UTC time. This is the default value. ● SERVER—If SERVER is set, a feature's start date is the start of the specified day in the server's default time zone. Equally, a feature will expire at the end of the day of the configured expiry date in the server's default time zone. See Editing the Local Settings Post-Installation for additional information.	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
licensing.security.json.enabled	The option that enables (`true`) or disables (`false`) security for JSON capability exchanges. Contact the producer to determine whether this policy applies to the licensed product you are using and whether the policy should be enabled. (Default is `true`.)	Yes
licensing.backup.uri	(Defined on back-up or main license server in a failover configuration; optional) The URI of the back-up license server to be included as reference information in the capability response to the client device. Use the following format: `http://server:port/fne/bin/capability` where *server*:*port* is the back-up license server's name and port number, as in: `http://22.22.2.222:7070/fne/bin/capability`	Yes
licensing.main.uri	(Defined on back-up or main license server in a failover configuration; optional) The URI of the main server to be included as reference information in the capability response to the client device. Use the following format: `http://server:port/fne/bin/capability` where *server*:*port* is the main license server's name and port number, as in: `http://11.11.1.111:7070/fne/bin/capability`	Yes
licensing.trackDeniedRequests	Set `licensing.trackDeniedRequests` to **`true`** to track failed capability requests for Cloud Licensing Service (CLS) instances and local license servers, otherwise set to **`false`**. When set to **`true`**, information about failed capability requests is sent to the Data Warehouse, where producers can query and analyze the information. Default setting for local license servers: `false`. Default setting for CLS instances: `true`. If you want to track failed capability requests for local license servers, in addition to setting `licensing.trackDeniedRequests` to **`true`**, you must also configure the policy settings `usageService.sync.enabled` and `usageService.sync.url`.	No
License Server Settings		
server.trustedStorageDir	The directory in which trusted storage resides. (Default is `${base.dir}`, which points to the `flexnetls/`*producer_name* folder in the service's or user's home directory.)	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
server.accessLogPattern	This property has been deprecated and should no longer be used. It is currently not possible to change the naming pattern of the access log file. **Access Log File Naming Pattern** Name of current log file: `access_request.log`. Name of rolled over files: `access_yyyy-mm-dd.request.log`. When a new day starts, a new access log file named `access_request.log` is created and populated with new logs. The access log file from the previous day is renamed to `access_yyyy-mm-dd.request.log` (for example, `access_2022-12-02.request.log`).	No
server.publisher DefinedHostId.policy	The property that determines whether to enable support for the use of a producer-defined hostid to identify the license server. To enable support, use the value `STRICT`. (Default is `false`, meaning support for this feature is disabled.) If the property is set to `STRICT`, the `server.hostType.order` and `licensing.enableBuiltinHostId` properties (if set) are ignored.	No
server.extendedHostId. enabled	The property that enables support for the use of extended hostids to identify the license server. (Default is `true`.)	No
server.hostType.order	The property that enables the producer to specify the order in which the local license server picks the hostid type. The order of the hostid types is specified as a comma-separated list, for example: `server.hostType.order=ETHERNET,FLEXID9,FLEXID10,VM_UUID` Valid values are all hostid types, with the exception of producer-defined hostid types. The following settings override the `server.hostType.order` property: • **server.publisherDefinedHostId.policy**—If a producer-defined hostid type is set using the `server.publisherDefinedHostId.policy` property, the `server.hostType.order` property is ignored. • **ACTIVE_HOSTID**—If `ACTIVE_HOSTID` is set either through the REST API or using a configuration file during server startup, the `server.hostType.order` property is ignored.	No
server.forceTSResetAllowed	The property that determines whether trusted storage can be reset when unsynchronized data still exists on the license server. (Default is `false`.)	No

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
server.backupMaintenance.interval	(Defined on back-up license server in a failover configuration; required) The maximum amount of time that the back-up server can serve licenses in a failover event. This value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. If this value is set to `0`, the back-up license server will serve licenses for an unlimited time while in failover mode. (Default is `3d`.)	No
server.syncCompatibility	(Used for migration from the FlexNet Embedded server application) The property that enables proper conversion of time units used for synchronization to and from the back office during the migration from the FlexNet Embedded server application to the FlexNet Embedded local license server. (Default is `false`.)	No
server.tenantId	(Required only for local license servers) Identifies the organisational account that the license server belongs to within the FlexNet Embedded ecosystem. This value is used by the license server to associate itself with the correct tenant in the back-office system (FlexNet Operations).	No
binding.enable.resilient.hostid	Enables resilient hostid handling, which makes hostids tolerant of changes to individual system identifiers. For an overview of this functionality, see Resilient Hostid Handling. Set to `true` to enable resilient hostids; set to `false` to disable. When resilient hostid handling is enabled, the local license server uses additional system attributes to preserve a stable hostid, even during routine and legitimate changes such as network reconfiguration or operating system updates. When the license server is registered in the back office, the collected metadata is securely stored with the device record. A license is served only if the weighted metadata comparison meets the required threshold, allowing the hostid to remain stable even when individual identifiers—such as Ethernet IDs—change.  ***Note ▪*** *Resilient hostid handling is currently supported for:* ● *Windows: physical and virtual machines (command-based installer only)* ● *Linux: virtual machines only*	No
Back Office URL		
lfs.url	The URL for back office to which the license server sends capability requests and synchronization data. The property is required for the online deployment model of the license server.	No
Policies for Polling Back Office for License Updates		

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
lfs.capability.enabled	The property that determines whether capability-request polling is enabled. If polling is enabled, a capability request is sent to the back office periodically to update the license server's license rights. This property is used for the online deployment model of the license server. (Default is `true`.)	No
lfs.capability.repeats	The amount of time between polling sessions to the back office. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `1d`; minimum is `10s`.)	Yes
lfs.capability.retryCount	The number of polling attempts allowed if the initial attempt fails. (Default is `3`.)	Yes
lfs.capability.retryRepeats	The amount of time between polling attempts, if the initial attempt fails. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `30s`; minimum is `1s`.)	Yes
Policies for Synchronizing to Back Office		
lfs.syncTo.enabled	The property that determines whether synchronization to the back office is enabled. This property should be viewed in combination with `lfs.syncTo.includeAll`: ● `lfs.syncTo.enabled=true` and `lfs.syncTo.includeAll=true`: (Online synchronization) This mode collects all historical client actions in the synchronization history and uploads this data to the back office as part of the synchronization. ● `lfs.syncTo.enabled=true` and `lfs.syncTo.includeAll=false`: (Online synchronization) This mode collects only the current state for each active client device at the point of synchronization and uploads this data to the back office ● `lfs.syncTo.enabled=false` and `lfs.syncTo.includeAll=true`: (Offline synchronization) This mode collects all historical and current client actions. This data is retained on the license server until the offline synchronization tools are run (see Offline Synchronization to the Back Office). ● `lfs.syncTo.enabled=false` and `lfs.syncTo.includeAll=false`: No synchronization data is collected (synchronization is disabled). Client data is deleted from the license server as soon as the client expires. (Default is `false`.)	No
lfs.syncTo.pagesize	The maximum number of client records to include in a synchronization message to the back office. A smaller page size limits the memory overhead at the expense of having multiple synchronization transactions. (Default is `100`; minimum is `10`; maximum is `256`.)	Yes

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
lfs.syncTo.threads	The number of parallel threads allocated to handle the synchronization of metered-usage and license-distribution data to the back office. (Default is `1`.)	Yes
lfs.syncTo.repeats	The amount of time between synchronization sessions to the back office. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `5m`; minimum is `10s`.)	Yes
lfs.syncTo.retryCount	The number of synchronization attempts to the back office allowed when an initial attempt fails. (Default is `4`.)	Yes
lfs.syncTo.retryRepeats	The amount of time between synchronization attempts when an initial attempt fails. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `5m`; minimum is `1s`.)	Yes
lfs.syncTo.delay	At license server startup, the amount of time the server should wait before initiating a synchronization session to the back office. (Default is `2s`; minimum is `2s`.)	Yes
lfs.syncTo.includeAll	The property that determines whether historical license-distribution data for concurrent features is collected and sent to the back office as part of the synchronization. This property should be viewed in combination with `lfs.syncTo.enabled`: • `lfs.syncTo.enabled=true` and `lfs.syncTo.includeAll=true`: (Online synchronization) This mode collects all historical client actions in the synchronization history and uploads this data to the back office as part of the synchronization. • `lfs.syncTo.enabled=true` and `lfs.syncTo.includeAll=false`: (Online synchronization) This mode collects only the current state for each active client device at the point of synchronization and uploads this data to the back office • `lfs.syncTo.enabled=false` and `lfs.syncTo.includeAll=true`: (Offline synchronization) This mode collects all historical and current client actions. This data is retained on the license server until the offline synchronization tools are run (see Offline Synchronization to the Back Office). • `lfs.syncTo.enabled=false` and `lfs.syncTo.includeAll=false`: No synchronization data is collected (synchronization is disabled). Client data is deleted from the license server as soon as the client expires. (Default is `true`.)	No
Policies for Synchronizing from Back Office		

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
lfs.syncFrom.enabled	The property that determines whether license-recovery from the back office is enabled. If recovery is enabled, the metered-usage data and license-distribution state for concurrent features is recovered from the back office when the license server initially starts up with a new or reset trusted storage. (Default is `false`.)	No
Policies for License Server Failover		
fne.syncTo.enabled	(Defined on back-up license server only; required) The property that determines whether to enable "license server to license server" synchronization in a failover configuration. (Default is `false`.)	Yes
fne.syncTo.mainUri	(Defined on back-up license server only; required) The URI of the main license server in a failover configuration. Use the following format: `http://server:port/fne/bin/capability` where *server*:*port* is the main license server's name and port number, as in: `http://11.11.1.111:7070/fne/bin/capability`	Yes
fne.syncTo.repeats	(Defined on back-up license server only) The amount of time between synchronization sessions from the main server to the back-up server in a failover configuration. (The back-up server initiates the sessions.) The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. (Default is `300s`; minimum is `5m`.)	No
fne.syncTo.pagesize	(Defined on back-up license server only) The maximum number of client records to include in a synchronization message to the back-up server. A smaller page size limits the memory overhead at the expense of having multiple synchronization transactions. (Default is `100`.)	Yes
fne.syncTo.retryCount	(Defined on back-up license server only) The number of synchronization attempts from the main server allowed when an initial attempt fails. (Default is `1`.)	Yes
fne.syncTo.retryRepeats	(Defined on back-up license server only) The amount of time between synchronization attempts when an initial attempt fails. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. The default is `60s`.	Yes
Usage Data Policies		

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
usageService.sync.enabled	When set to `true`, usage data from online local license servers is sent to the Data Warehouse, where producers can query and analyze the information. Default setting for local license servers: `false`. Default setting for CLS instances: `true`. If you set `usageService.sync.enabled` to `true`, you must also configure `usageService.sync.url` (see below). For local license servers, you must also specify `server.tenantId`.  ***Note ▪*** *Usage data from Cloud Licensing Service (CLS) instances is sent to the Data Warehouse by default; no separate configuration is necessary.*	No
usageService.sync.url	Specifies the usage service URL. The URL is required to push usage data from a local license server to the Data Warehouse. The setting must point to the usage service URL, which is `https://siteId.compliance.flexnetoperations.com/usage/api/1.0/sync` where *siteId* is replaced with the specific site ID supplied by Revenera, for example: `https://flex1234.compliance.flexnetoperations.com/usage/api/1.0/sync` This URL must be provided if `usageService.sync.enabled` is to `true`. This setting requires that `server.tenantId` is set.  ***Note ▪*** *Usage data from Cloud Licensing Service (CLS) instances is sent to the Data Warehouse by default; no separate configuration is necessary.*	No
Security Policies		
security.enabled	The option that enables (`true`) or disables (`false`) administrative security on the license server. When administrative security is enabled, operations used to administer the license server are "secured" (that is, credentials are required to perform them). See Managing Administrative Security on a Local License Server or CLS Instance. When this option is `true`, the remaining policies in this *Security Policies* section are in effect. (Default is `false`.)	Yes

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
security.token.duration	The duration of the JSON web token (generated when a user successfully authenticates credentials on the license server). When the token expires, credentials must be re-entered to re-authorize. The value can be specified with an optional unit-suffix letter—`s`, `m`, `h`, `d`, or `w`—indicating seconds, minutes, hours, days, or weeks. If no suffix is used, the server assumes the value is in seconds. The default is `1d`. This policy is not editable in the FlexNet License Server Manager.	Yes
security.http.auth.enabled	The option that enforces the use of HTTPS to perform secured administrative (and possibly licensing) operations on the license server. ● When `false`, the policy enforces the use of HTTPS to perform secured operations. (An error is generated for any attempt to perform a secured operation using HTTP.) ● When `true`, the policy allows either HTTP or HTTPS to perform secured operations. (This is the default.) This policy is not editable in the FlexNet License Server Manager.	Yes
security.ip.whitelist	(Available only when `security.enabled` is `true`) The list of IP addresses for those components (devices) that you determine should have access to the license server without having to provide credentials to access secured REST API endpoints. For example, you might want a machine in your IT department to have such access to the endpoints for fixing issues or performing maintenance. List only IP4 or IP6 addresses; and separate each address with a comma, as this example value shows: `111.222.2.2,111.333.3.3` This policy is not editable in the FlexNet License Server Manager.  ***Important ▪*** *In certain contexts, configuring `security.ip.whitelist` can be considered a security risk. For example, if **`X-Forwarded-For`** is active and no gateway or reverse proxy server is in use, potential attackers could spoof the header to hide their actual IP addresses or impersonate other users. To mitigate risks, consider disabling **`X-Forwarded-For`** (XFF) headers (set `disable-xforwarded-for=true` in the `local-configuration.yaml` file).*	Yes

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
security.anonymous	The option that determines whether or not users need credentials for "read" access to the license server's endpoints: • When the value is `true`, all user accounts are automatically given "read" rights (ROLE_READ) and do not need to provide credentials for "read" access. • When the value is `false`, a given user account must be explicitly assigned ROLE_READ in order to perform "read" operations. (The exception occurs when no role is assigned to an account, in which case ROLE_READ is assigned as the only role by default.) Credentials are then required to perform any "read" operation. If an account is not authorized for ROLE_READ, no "read" access is given. This setting provides additional protection against unauthorized queries on the license server. This policy is not editable in the FlexNet License Server Manager. (Default is `false`.)	Yes
Logging Policies		
logging.directory	The directory to which the license server writes the log for the license server. The default is `${base.dir}/logs`, where `${base.dir}` points to the `flexnetls/`*`producer_name`* folder in the service's or user's home directory.	No
logging.threshold	The lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`. For example, if `FATAL` is set, only messages about fatal events are recorded. However, if `WARN` is set, fatal-event, error, and warning messages are recorded. (Default is `INFO`.) **Logging categories** `FATAL`—Errors that prevent the server from starting up `ERROR`—Serious errors `WARN`—Warnings `INFO`—Informational messages `LICENSING`—Server responses such as, for example, capability responses and JSON replies `POLICY`—Additional information for checkout filters (these are selective license filters customizable by the producer) `DEBUG`—Additional debug-level information. The license server should not use a logging level of `DEBUG` for a long period, because it can have a negative impact on license server performance. It is not recommended to use `DEBUG` on production license servers.	Yes

Table A-1 ▪ License Server Policy Settings (cont.)

Setting	Description	Edit-able?
graylog.host	The host name of a Graylog server, if any, to which logging messages are sent.	Yes
graylog.threshold	The lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`. For example, if **`FATAL`** is set, only messages about fatal events are recorded. However, if **`WARN`** is set, fatal-event, error, and warning messages are recorded. (Default is `WARN`.)	Yes

B

Model Definition Grammar for Named License Pools

The model definition defines named license pools and conditions for rules that allow or deny access to features.

The following sections provide information and examples for creating model definitions:

- Model Definition Grammar and Syntax—EBNF
- Use Case Examples and Their Model Definitions for Named License Pools
- Model Definition Examples for Reservations Converted to Named License Pools

Model Definition Grammar and Syntax—EBNF

The following section shows all possible grammar structures of the language used to write a model definition. The syntax representation is based on the Extended Backus–Naur Form (EBNF), which is a formal notation that can be used to describe other languages. It is grouped into the Parser and the Lexer. Both can be used with ANTLR (for more information, see antlr.org).

Important ▪ *Each definition can have only one model. Reserved model names are* `reservations` *and* `default`*.*

Parser

```
parser grammar model;
options { tokenVocab=modelLexer; }

// Parser
model: MODEL model_name LBRACE partitions? modelRule* RBRACE EOF;

model_name: QUOTEDSTRING ;
partitions: PARTITIONS LBRACE partition* RBRACE ;

partition: PARTITION partition_name LBRACE feature_spec* RBRACE ;
partition_name: QUOTEDSTRING ;
```

```
feature_spec: feature_name feature_version amount vendor_match?  ;
feature_name: QUOTEDSTRING ;
feature_version: major_version ( DOT minor_version )? ;
major_version: DIGIT+ ;
minor_version: DIGIT+ ;
amount: integer_amount | percentage_amount | REMAINDER ;
integer_amount: DIGIT+ ;
percentage_amount: DIGIT+ PERCENT ;

vendor_match: VENDOR STRING MATCHES match_string;
match_string: QUOTEDSTRING ;

modelRule: ON compound_matcher LBRACE rule_body RBRACE ;

compound_matcher: negatable_matcher compound_rhs* ;

compound_rhs: compound_and_rhs | compound_or_rhs ;

compound_and_rhs: (AND | C_AND) negatable_matcher ;
compound_or_rhs:  (OR  | C_OR) negatable_matcher ;

negatable_matcher: negated_matcher | simple_matcher ;

negated_matcher: (NOT | C_NOT) simple_matcher ;

simple_matcher: hostid_matcher | hostname_matcher | hosttype_matcher | vendor_dictionary_matcher |
any_matcher ;

hostid_matcher: HOSTID LPAREN parameter_list RPAREN ;
hostname_matcher: HOSTNAME  LPAREN parameter_list RPAREN ;
hosttype_matcher: HOSTTYPE LPAREN parameter_list RPAREN ;
vendor_dictionary_matcher: DICTIONARY LPAREN keyword_parameter_list RPAREN ;
any_matcher: ANY LPAREN RPAREN ;

parameter_list: parameter (COMMA parameter)* ;
parameter: QUOTEDSTRING ;
keyword_parameter_list: keyword_parameter (COMMA keyword_parameter)* ;

keyword_parameter: keyword_key COLON keyword_value ;
keyword_key: QUOTEDSTRING ;
keyword_value: WILDCARD | QUOTEDSTRING ;

rule_body: use_statement? server_specified? action? ;
use_statement: USE partition_name_list ;

server_specified: WITHOUT REQUESTED FEATURES LBRACE partition_server_specified |
feature_server_specified RBRACE ;

partition_server_specified: ALL FROM partition_name_list ;
feature_server_specified: feature_spec* ;

partition_name_list: partition_name (COMMA partition_name)* ;

action: ACCEPT | DENY | CONTINUE ;
```

Lexer

```
lexer grammar modelLexer;

MODEL : 'model';
ON : 'on';
PARTITIONS : 'partitions';
PARTITION : 'partition';
REMAINDER : 'remainder';
ANY : 'any';
HOSTID : 'hostid';
USE: 'use';
ACCEPT : 'accept';
DENY : 'deny';
CONTINUE : 'continue';
WITHOUT : 'without';
REQUESTED : 'requested';
FEATURES : 'features';
ALL : 'all';
FROM : 'from';
HOSTNAME: 'hostname';
HOSTTYPE: 'hosttype';
DICTIONARY: 'dictionary';
VENDOR: 'vendor';
STRING: 'string';
MATCHES: 'matches';

LPAREN : '(';
RPAREN : ')';
LBRACE : '{';
RBRACE : '}';
COMMA  : ',';
COLON  : ':';
WILDCARD : '*';
PERCENT : '%';
DOT     : '.';
AND :   'and';
OR : 'or';
NOT : 'not';
C_AND : '&&';
C_OR  : '||';
C_NOT :  '!';

DIGIT: '0'..'9' ;
QUOTEDSTRING: '"' (~('"' | '\\' | '\r' | '\n') | '\\' ('"' | '\\'))* '"';

// Whitespace, comments
WS           : [ \t\r\n\u000C]+ -> skip;
COMMENT      : '/*' .*? '*/' -> skip ;
LINE_COMMENT : '//' ~[\r\n]* -> skip ;

UNKNOWN_CHAR : . ;
```

Use Case Examples and Their Model Definitions for Named License Pools

The following examples illustrate possible use cases for sharing feature counts between license pools:

Background Information for Use Cases

The following is information to keep in mind when reading the information in this section.

Vendor Dictionary Data

Many of the following use cases use vendor dictionary data to define conditions for license allocation.

The vendor dictionary enables the software producer to send custom data in a capability request (in addition to the FlexNet Embedded–specific data) to the license server and vice-versa. Basically, the vendor dictionary provides a means to send information back and forth between the client and server for any producer-defined purposes, as needed; FlexNet Embedded does not interpret this data.

Vendor dictionary data is stored as key–value pairs. The key name is always a string, while a value can be a string or a 32-bit integer value. (In certain cases, the value can also be an array of a string and/or 32-bit integer.) Keys are unique in a dictionary and hence allow direct access to the value associated with them.

A common theme across the use cases in this appendix is that feature counts are shared between business units called Sales and Engineering. These business units would be identified by `"business-unit" : ["engineering", "sales" ]` entries in the vendor dictionary.

For information about whether the vendor dictionary supports arrays and the data that might be available, contact your software producer.

Tip ▪ *For an example that demonstrates a vendor dictionary condition that uses an array, see Use Case: Making Feature Counts Available to Multiple Business Units.*

Default Behavior If No Conditions Are Met

When a feature request does not meet any of the conditions in the model definition, an `on any()` condition is expected that either denies or allows access to a license pool. If no such `on any()` condition is provided, the default is that the feature will be served from the default license pool (if features are available). This behavior is equivalent to the following code:

```
on any() {
    use "default"
    accept
}
```

The examples in this chapter generally do not explicitly show this final `on any()` condition.

Use Case: Simple Allow List

Requirement: Allow access to specific hostids

In this scenario, only declared hostids are allowed access. Note that no named license pools are needed in this scenario.

Instead of specifying all hostids in one condition within a rule of access, you can also include multiple rules with the “on hostid” condition in the model definition.

Model Definition Example

```
model "exampleModel" {
    on hostid("F01898AD8DD3/ETHERNET", "5E00A4F17201/ETHERNET") {
        use "default"
        accept
    }

    on any() {
        deny
    }
}
```

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

Use Case: Simple Block List

Requirement: Deny access to specific hostids

In this scenario, all users except a declared few are allowed access.

Model Definition Example

```
model "exampleModel" {
    on hostid("user1@example.com/USER", "user2@example.com/USER") {
        deny
    }

    on any() {
        use "default"
        accept
    }
}
```

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

Use Case: Sharing Counts Between Business Units

Requirement: Share feature counts between two business units.

This scenario features two business units called Sales and Engineering, which have been defined by entries in the vendor dictionary.

The model definition defines two named license pools, one for the Engineering business unit and another for the Sales business unit. The definition shares the total feature count of 10 for feature f1 equally between both business units, meaning each business unit gets 5.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "f1" 1.0 5
        }
        partition "sales" {
            "f1" 1.0 5
        }
    }

    on dictionary("business-unit" : "engineering") {
        use "engineering"
        accept
    }

    on dictionary("business-unit" : "sales") {
        use "sales"
```

```
        accept
    }
}
```

Use Case: Assigning Extra Counts To Business Unit

Requirement: Make specified number of features accessible only to one particular business unit.

This scenario features the same business units as before—Sales and Engineering.

For this demonstration, let's assume that the total feature count for f1 is 10. Of the total count, 3 licenses are allocated exclusively to the named license pool called `engineering`. In addition, `engineering` is allowed access to the remaining 7 licenses (on a first-come-first-served basis) from the default license pool.

The Sales business unit (which has no pre-allocated counts in this scenario) only has access to the default license pool.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "f1" 1.0 3
        }
    }

    on dictionary("business-unit" : "engineering") {
        use "engineering", "default"
        accept
    }
}
```

Use Case: Exclusive Use of Feature Counts for Business Unit

Requirement: Grant exclusive use of features to a specified business unit

This scenario uses the same business units and named license pools as the previous example: the default license pool, and a named license pool called `engineering` for the Engineering business unit. Other business units also exist (for example, the Sales business unit).

Let's assume that the total feature count for a feature called "cad" is 10. The entire count (expressed here as `100%`) is allocated to the named license pool `engineering`, for exclusive use by the Engineering business unit. This effectively denies access to feature "cad" for any capability request that comes from a different business unit (that is, any request that does not include the vendor dictionary entry `"business-unit" : "engineering"`).

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "cad" 1.0 100%                              // entire feature count
        }
```

```
    }

    on dictionary("business-unit" : "engineering") {
        use "engineering"
        accept
    }
}
```

Use Case: Exclusive Use of Feature Counts for Business Unit With Exception of Specific Clients

Requirement: Grant exclusive use of features to a specified business unit, with the exception of specified clients.

This scenario uses the same framework as the previous use case, Use Case: Exclusive Use of Feature Counts for Business Unit, but adds a filter that blocks specified clients.

Capability requests are granted for all clients from the business unit Engineering, except for requests coming from clients with the hostid `"5e00a4f17204/ETHERNET"` or `"5e00a4f17205/ETHERNET"`.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "cad" 1.0 100%                                  // entire feature count
        }
    }

    on dictionary("business-unit" : "engineering") and not hostid("5e00a4f17204/ETHERNET",
    "5e00a4f17205/ETHERNET") {
        use "engineering"
        accept
    }
}
```

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

For information about the operators AND and NOT used in this example, see AND, OR, and NOT Operators.

Use Case: Exclusive Use of Feature Counts for Business Unit and Specified Clients from other Business Units

Requirement: Grant exclusive use of features to a specified business unit and to a number of specified clients.

This scenario uses the same framework as the use case described in Use Case: Exclusive Use of Feature Counts for Business Unit With Exception of Specific Clients. However, instead of filtering out specified clients, it also grants access to a number of select clients that are not part of the Engineering business unit.

Capability requests are granted for all clients from the business unit Engineering, and for clients with the hostid "5e00a4f17204/ETHERNET" or "5e00a4f17205/ETHERNET" (which may well be part of other business units).

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "cad" 1.0 100%                                  // entire feature count
        }
    }

    on dictionary("business-unit" : "engineering") or hostid("5e00a4f17204/ETHERNET", "5e00a4f17205/
    ETHERNET") {
        use "engineering"
        accept
    }
}
```

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

For information about the operators AND and NOT used in this example, see AND, OR, and NOT Operators.

Use Case: Assigning Features Based on Combined hosttype and hostname Properties

Requirement: Allow only clients that match specified `hosttype` and `hostname` properties access to a named license pool.

This scenario uses the same Engineering business unit and named license pool that you already know from previous examples.

In this example, a producer wants to allow client devices that match both a particular host type and host name access to the named license pool `engineering`. This can be achieved by combining the `hosttype` and `hostname` conditions using the AND operator. Requests from devices that match only the host type or host name, or none of these, are served from the default license pool.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "f1" 1.0 100
        }
    }

    on hosttype("tv") and hostname("device-1"){
        use "engineering"
        accept
    }
}
```

Use Case: Assigning Features Based on Vendor String Property

Requirement: Allocate feature counts to named license pools based on the product name (specified through the vendor string property), and return counts to their original named license pool after use.

In this scenario, feature counts come from a product for which two different versions exist: an Evaluation version and a Permanent version. The Evaluation version includes three features (f1, f2, f3), and the Permanent version includes four features (f1, f2, f4, f5).

As a prerequisite, the product names `Eval` and `Permanent` must be defined in the vendor string `{EntitlementLineItem.productName}` for the respective product version.

The model definition defines four named license pools. The `vendor string matches` directive is used to allocate counts—expressed in percent—from the Evaluation and Permanent product versions to different named license pools. The directive is evaluated when the license server loads the model definition.

Requests for feature counts are accepted or denied based on vendor dictionary data. In this scenario, the following vendor dictionary key-value pairs must be set up:

```
business-unit:engineering
business-unit:sales
ProductName:Eval
ProductName:Permanent
```

The model definition's rules of access only allow access to a particular named license pool if a capability request satisfies both conditions in the rule: it must match the dictionary entry for `ProductName` and for `business-unit`. Capability requests that only satisfy one of the conditions or none are rejected, and access to the named license pool is blocked.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "eval-engineering" {
            "f1" 1.0 75% vendor string matches "ProductName:Eval"
            "f2" 1.0 75% vendor string matches "ProductName:Eval"
            "f3" 1.0 75% vendor string matches "ProductName:Eval"
        }

        partition "permanent-engineering" {
            "f1" 1.0 75% vendor string matches "ProductName:Permanent"
            "f2" 1.0 75% vendor string matches "ProductName:Permanent"
            "f4" 1.0 75% vendor string matches "ProductName:Permanent"
            "f5" 1.0 75% vendor string matches "ProductName:Permanent"
        }

        partition "eval-sales" {
            "f1" 1.0 25% vendor string matches "ProductName:Eval"
            "f2" 1.0 25% vendor string matches "ProductName:Eval"
            "f3" 1.0 25% vendor string matches "ProductName:Eval"
        }

        partition "permanent-sales" {
            "f1" 1.0 25% vendor string matches "ProductName:Permanent"
            "f2" 1.0 25% vendor string matches "ProductName:Permanent"
```

```
            "f4" 1.0 25% vendor string matches "ProductName:Permanent"
            "f5" 1.0 25% vendor string matches "ProductName:Permanent"
        }
    }

    on dictionary("ProductName":"Eval") and dictionary ("business-unit" : "engineering") {
        use "eval-engineering"
        accept
    }
    on dictionary("ProductName":"Permanent") and dictionary ("business-unit" : "engineering") {
        use "permanent-engineering"
        accept
    }
    on dictionary("ProductName":"Eval") and dictionary ("business-unit" : "sales") {
        use "eval-sales"
        accept
    }
    on dictionary("ProductName":"Permanent") and dictionary ("business-unit" : "sales") {
        use "permanent-sales"
        accept
    }
    on any() {
        deny
    }
}
```

The following diagram illustrates the allocation of features to the four named license pools:

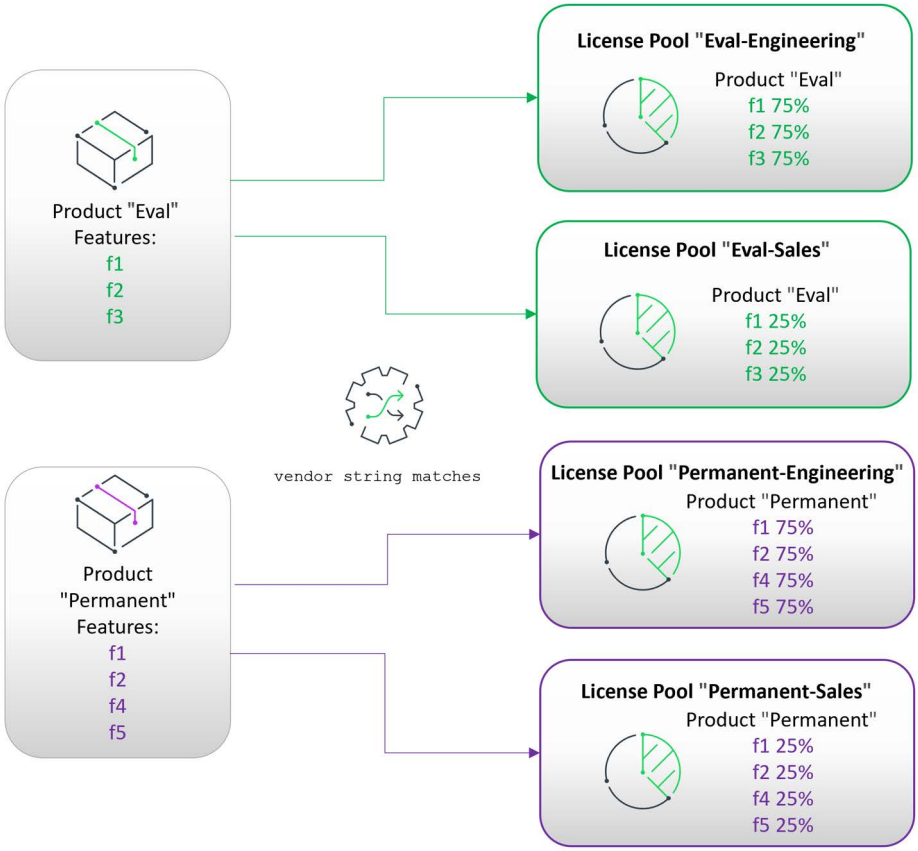


Figure B-1: "vendor string matches" Directive

General Information for Using "Vendor String Matches"

The use case "Assigning Features Based on Vendor String Property" illustrates how to allocate feature counts to named license pools based on the product name. However, the vendor string can be used to allocate counts on the basis of any of the substitution variables that FlexNet Operations supports.

The producer defines the vendor string in the license model when they create a line item in the back office. The vendor string can hold arbitrary producer-defined license data. Data can range from feature selectors to pre-defined substitution variables.

Vendor String Uniqueness

The vendor string must be unique, that is, no duplicate entries must appear in the same model definition.

In order to avoid unexpected behaviour, a model definition should always result in a single feature slice being generated per line item in each named license pool (that is, each slice within a named license pool should be mapped to a unique feature ID).

Allocating Multiple Line Items of the Same Feature to Same Named License Pool

The `vendor string matches` directive can also be used to allocate feature counts from designated line items—sharing the same feature name and version—to the ***same named license pool***. Line items are specified using a key/value pair provided via the vendor string.

Example

Feature counts come from a product with the feature f1 version 1.0, which is represented by two distinct feature IDs:

- `f1 1.0 10, vendor string: "%%ACTID:act-1%%", featureId: fid-1`
- `f1 1.0 10, vendor string: "%%ACTID:act-2%%", featureId: fid-2`

The following vendor dictionary key-value pairs must be set up:

```
ACTID:act-1
ACTID:act-2
```

In the model definition, the `vendor string matches` directive is used to allocate counts to the same named license pool **p1**. The directive is evaluated when the license server loads the model definition.

```
partitions {
    partition "p1" {
        "f1" 1.0 2 vendor string matches "ACTID:act-1"
        "f1" 1.0 1 vendor string matches "ACTID:act-2"
    }
}
```

The resulting named license pool "p1" after posting the model definition will contain the following feature slices (feature slices are portions of feature counts allocated to a named license pool):

- f1 1.0 **2**, featureId: `fid-1`
- f1 1.0 **1**, featureId: `fid-2`

Use Case: Device-specific Handling—Sharing Feature Counts Based On Hosttype

Requirement: Use the `hosttype` property to route the capability request to the appropriate named license pool.

In this scenario there are three different models of routers, which are distinguished through the `hosttype` property in the capability request. Counts of the “router” feature are split between three different named license pools—called `small`, `medium`, and `large`—and the “on hosttype” condition is used to route the request to the appropriate named license pool.

The hosttype is a value that is optionally set by the software producer on a client device. (This hosttype is also sometimes referred to as a device type.) A hosttype is a human-readable “alias”—in contrast to the hostid—which can optionally be included in a capability request. Contact your software producer to find out about hosttypes that are available to you.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "small" {
            "router" 1.0 5
        }
        partition "medium" {
            "router" 1.0 10
        }
        partition "large" {
            "router" 1.0 20
```

```
        }
    }

    on hosttype("small") {
        use "small"
        accept
    }
    on hosttype("medium") {
        use "medium"
        accept
    }
    on hosttype("large") {
        use "large"
        accept
    }
}
```

Use Case: Named License Pool Receiving Entire Remaining Feature Count

Requirement: Distribute all counts of a feature between three named license pools as equally as possible, with none in the default named license pool.

If you declare three named license pools, two with a count specifier of 33% and one with 34%, integer rounding might cause some counts to slip into the default license pool. You can avoid this by replacing the last count specifier with the `remainder` keyword, which will match all available remaining counts.

Note that any license pool lower than the one specifying the `remainder` keyword in the model definition receives zero counts for that feature.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "p1" {                    // Total feature count of f1: 10
            "f1" 1.0 33%                    // Feature count in p1: 3
        }
        partition "p2" {
            "f2" 1.0 33%                    // Feature count in p2: 3
        }
        partition "p3" {
            "f3" 1.0 remainder              // Feature count in p3: 4
        }
    }
```

Use Case: Using Regular Expression to Allocate License Counts

Requirement: Allow clients that match a specific hosttype pattern access to a specified named license pool. Allow clients from business units whose name matches a specific pattern access to two specified named license pools.

As before, this scenario features two business units called Sales and Engineering, which have been defined by entries in the vendor dictionary. Using a regular expression in each condition means that the rule can be simplified—instead of creating a separate condition for every valid hostname or vendor dictionary string, a regular expression defines the pattern that a hostname and string must match to gain access to feature counts.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "engineering" {
            "f1" 1.0 50%
        }
        partition "sales" {
            "f1" 1.0 50%
        }

    on hostname("~/[A-Z]{2}[0-9]{3}/") {
        use "engineering"
        accept
    }

    on dictionary("business-unit":"~/Eng[0-9]+/") {
        use "engineering", "sales"
        accept
    }

    on any() {
        deny
    }
```

Interpreting the Regular Expression

- Requests from clients with a hostname matching the following pattern get access to the `engineering` license pool:

  `[A-Z]` any uppercase letter
  `{2}` matched exactly twice
  `[0-9]` any number
  `{3}` matched exactly three times

  Examples for valid hostnames: `AB123`, `CD789`

- Requests from the `engineering` business unit (as defined in the vendor dictionary) matching the following pattern get access to the `engineering` and `sales` license pools:

  `Eng` must start with “Eng” (matching capitalization)
  `[0-9]` followed by any number

  Examples for valid vendor dictionary entries: `Eng123`, `Eng456789`

Use Case: Making Feature Counts Available to Multiple Business Units

Requirement: Make feature count from multiple named license pools available to users that belong to more than one business unit.

Important ▪ *This use case is only applicable to scenarios where the vendor dictionary value can be an array. For information about whether the vendor dictionary supports arrays and the data that might be available, contact your software producer.*

Previous use cases showed how the model definition could allocate feature counts based on a business unit named in the vendor dictionary. However, these only catered for scenarios where a user belongs to a single business unit.

This use case demonstrates a simple way of allocating feature counts to multiple business units—called DevOps and Engineering in this example—that are specified in an array in the vendor dictionary.

The vendor dictionary entry would look similar to this:

```
"vendorDictionary": {
    "business-unit": [
        "Engineering",
        "DevOps",
        "Security"
        ],
    "Region": "EMEA"
}
```

Consider the following capability request that contains an array of strings associated with the `business-unit` key in the vendor dictionary:

```
{
    "features": [
        {
            "count": 20,
            "name": "cad",
            "version": "1.0"
        }
      ],
      "hostId": {
            "type": "string",
            "value": "h1"
    },
    "vendorDictionary": {"Region":"EMEA", "business-unit" : ["Engineering","DevOps"]},
    "borrow-interval": "300"
}
```

The vendor dictionary will match any of these clauses:

- `on dictionary("business-unit", "Engineering")`
- `on dictionary("business-unit", "DevOps")`
- `on dictionary("Region", "EMEA")`

If the vendor dictionary element contains a simple value (string, 32-bit integer) an equality match is done.

If the vendor dictionary element contains an array, a "contains" match is done, meaning that it is sufficient if only one matching element exists in the array.

The conditions in the following model would grant such a capability request—coming from a client device that belongs to multiple business units—access to features in both named license pools, `EngDevOps` and `EngSecurity`.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "EngDevOps" {
            "cad" 1.0 30%
        }
        partition "EngSecurity" {
            "cad" 1.0 30%
        }
    }

    on dictionary("business-unit":"Engineering") and dictionary("Region":"EMEA"){
        use "EngDevOps"
        accept
    }

    on dictionary("business-unit":"Security") and dictionary("Region":"EMEA"){
        use "EngSecurity"
        accept
    }

    on any() {
        use "default"
        accept
    }
}
```

Use Case: Letting Server Specify Counts

Requirement: Let the server specify the desired counts, instead of the client

The following model definition demonstrates how to let the server specify the features that should be assigned to a client device. The model uses the `without requested features` directive in combination with a feature specification. Note that the server cannot override a capability request containing desired features coming from the client.

The following rule of access assigns a count of feature f1 to each capability request with the hostname “myhost”.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "p1" {
            "f1" 1.0 10
        }
    }

    on hostname("myhost") {
```

```
        use "p1", "default"
        without requested features {
            "f1" 1.0 1
        }
        accept
    }
}
```

You can also combine the `without requested features` directive with `all from "license_pool_name"`. This variation will result in the first capability request receiving all counts from the specified named license pool. For a scenario where `all from "license_pool_name"` might be used, see Scenario: Server-specified Counts.

Note ▪ *Server-specified desired feature counts should be marked with the "partial" attribute, indicating that partial checkout is allowed if the feature's count falls short of the desired count.*

Use Case: Accumulating Counts from Multiple Named License Pools ("Continue" Action)

Requirement: Allow a capability request to collect feature counts from more than one license pool

This use case defines two named license pools for feature counts. If a capability request comes from Europe (containing the key-value pair `"region"/"Europe"`, defined by the producer using vendor dictionary data), it gets access to named license pool p1. If the request comes from a device labeled "10A" (containing the key-value pair `"model" : "10A"`, defined by the producer using vendor dictionary data), it gets access to named license pool p2. If both conditions are true, the request gets access to both p1 and p2.

Model Definition Example

```
model "exampleModel" {
    partitions {
        partition "p1" {
            "f1" 1.0 5
        }
        partition "p2" {
            "f1" 1.0 5
        }
    }

    on dictionary("region" : "Europe") {
        use "p1"
        continue
    }

    on dictionary("model" : "10A") {
        use "p2"
        continue
    }
}
```

Use Case: Restrict Access to a License Pool for a Specific User

Note ▪ *The user identifier condition can be used only if access requests include user identifier information in the* `userIdentifier` *field.*

Requirement: Grant exclusive use of features to a specified business unit and to a number of specified users.

In this example, the model definition grants access to the named license pool `engineering` for all clients belonging to a particular business unit, as well as for a specific user who may not belong to that business unit. Specifically, access to the `engineering` license pool is granted to:

- Any client whose vendor dictionary contains `business-unit : prod-eng`, and
- Any client whose user identifier is `sam`

All other requests are handled by the fallback rule (feature requests that do not meet any of the conditions in the model definition are served from the `default` license pool, if features are available).

```
model "userIdentifiers" {
    partitions {
        partition "engineering" {
            "f1" 1.0 100
        }
     }

    on dictionary("business-unit":"prod-eng") OR userIdentifier("sam"){
        use "engineering"
        accept
    }

    on any() {
        use "default"
        accept
    }
}
```

Use Case: Allow Specified Users Access to Specified License Pool

Note ▪ *The user identifier condition can be used only if access requests include user identifier information in the* `userIdentifier` *field.*

Requirement: Allow multiple users to access the same named license pool.

This example demonstrates a recommended approach in which multiple users access the same named license pool, with user identifiers used in access rules to control eligibility.

```
model "userIdentifiers" {
```

```
    partitions {
        partition "engineering" {
            "f1" 1.0 100
        }
     }

    on userIdentifier("alice", "james", "umi"){
        use "engineering"
        accept
    }

    on any() {
        use "default"
        accept
    }
}
```

This model grants licenses from the `engineering` license pool to users `alice`, `james`, and `umi`. All other users are granted licenses from the `default` license pool (if features are available).

For best practices and limitations, see User Identifier Condition.

Model Definition Examples for Reservations Converted to Named License Pools

The following examples show model definitions that might result from manually converting reservations to named license pools. Conversion involves creating and uploading a model definition that reproduces the allocation achieved previously through reservations.

Important ▪ *You must delete any existing reservation groups before uploading the model definition.*

Scenario: Counts Reserved By Hostid

The original reservation model consists of per-hostid reserved counts, together with the ability to get counts from the default shared pool. Reservations might look like this when expressed in a model definition:

```
model "model-1" {
    partitions {
        partition "reservation-1" {
            "f" 1.0 1
        }
    }

    on hostid("F01898AD8DD3/ETHERNET") {
        use "reservation-1", "default"
        accept
    }

    on any() {
```

```
        use "default"
        accept
    }
}
```

Note that creating rules of access based on hostids is possible, but generally not recommended. Maintaining large lists of rules of access with the hostid condition is tedious and prone to error. Instead, consider basing the rules of access on hostname or hosttype properties or vendor dictionary key/value pairs.

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

Scenario: Server-specified Counts

Sometimes, a capability request does not contain any desired features. In this case, the server can be configured to specify the features that should be assigned to a client device. If a reservation exists for the client that sent the capability request, the reserved counts are acquired. The following model reservation example demonstrates how this behavior is replicated with named license pools. The `without requested features` directive replicates the server-specified count, and `all from "license_pool_name"` allocates all feature counts from the specified named license pool to the first client sending a capability request that fulfills the conditions defined in the rule of access. The `all from "license_pool_name"` instruction essentially acts as a shortcut to the feature counts specified for the relevant named license pool in the model definition. If the specified named license pool does not have sufficient counts to satisfy the request, counts are allocated from subsequent named license pools listed in the rules of access (provided that the client has access to those named license pools).

Let's assume that the original reservation model consists of per-hostid reserved counts, together with the ability to get counts from the default shared pool. An equivalent model definition and rule of access might look like this:

```
model "reservations" {
    partitions {
        partition "reservation-1" {
            "f1" 1.0 1
        }
    }

    on hostid("F01898AD8DD3/ETHERNET") {
        use "reservation-1", "default"
        without requested features {
            all from "reservation-1"
        }
        accept
    }
}
```

The `without requested features` directive can also be combined with a feature specification to produce more fine-grained rules of access. For a use case example, see Use Case: Letting Server Specify Counts.

As noted in the previous scenario, creating rules of access based on hostids is generally not recommended. Instead, consider basing the rules of access on hostname or hosttype properties or vendor dictionary key/value pairs.

Note ▪ *Server-specified desired feature counts should be marked with the "partial" attribute, indicating that partial checkout is allowed if the feature's count falls short of the desired count.*

Note ▪ *The hostid is specified as a `value/type` pair (for example, `7200014f5df0/ETHERNET`). If a hostid condition does not specify the hostid type, it is assumed that the hostid is of type `string`.*

C

Logging Functionality on the Local License Server

This appendix describes the logging functionality available for the local license server, specifically:

- Logging Style
- Custom Log Configurations
- Integration of License Server Logging With External Systems

Logging Style

A logging style configuration parameter can be used to configure rollover, JSON formatting and timestamp behavior. The different logging styles can be specified in the `local-configuration.yaml` file, using the `loggingStyle` property.

Set the property using the following format (the space after the colon is mandatory):

```
loggingStyle: value
```

The property can take one of the following values:

- `DAILY_ROLLOVER`—The log file is closed at midnight local time, compressed with gzip, and a new log file is started. Timestamp values use the local time zone. This is the default if no logging style is specified.
- `DAILY_ROLLOVER_UTC`—Same as `DAILY_ROLLOVER`, but with UTC timestamp.
- `CONTINUOUS`—Legacy value. Rollover is handled as `DAILY_ROLLOVER`.
- `CONTINUOUS_UTC`—Legacy value. Rollover is handled as `DAILY_ROLLOVER_UTC`.
- `JSON_ROLLOVER`—Logs are formatted using JSON. The log file is closed at midnight local time, compressed with gzip (suitable for Filebeat), and a new log file is started. Always uses ISO 8601 UTC timestamps.
- `JSON`—Logs are emitted as JSON to stdout only (suitable for Docker). Always uses ISO 8601 UTC timestamps.

***Note** ▪ The Docker-friendly `JSON` logging style is not supported when the local license server is run as a Windows or Linux service (because it only writes to stdout). If set, the style will be changed to `DAILY_ROLLOVER`.*

Custom Log Configurations

If the preconfigured logging styles do not meet your requirements, you can customize the logging configuration. To do so, prepare an external configuration file and specify the path to that file in an environment variable ($LOGGING_CONFIGURATION).

Documentation for logback appenders can be found here: https://logback.qos.ch/manual/appenders.html

To construct the external configuration file, extract a logback configuration from the `flexnetls.jar` file and modify it to suit your requirements. The following tables shows the existing configurations.

Table C-1 ▪

Logging Style	Extraction Command
CONTINUOUS	`unzip -p flexnetls.jar BOOT-INF/classes/logback-continuous.xml`
CONTINUOUS_UTC	`unzip -p flexnetls.jar BOOT-INF/classes/logback-continuous.xml`
DAILY_ROLLOVER	`unzip -p flexnetls.jar BOOT-INF/classes/logback-rollover.xml`
DAILY_ROLLOVER_UTC	`unzip -p flexnetls.jar BOOT-INF/classes/logback-rollover.xml`
JSON	`unzip -p flexnetls.jar BOOT-INF/classes/logback.xml`
JSON_ROLLOVER	`unzip -p flexnetls.jar BOOT-INF/classes/logback-rollover-json.xml`

***Note** ▪ The "_UTC" variants share a configuration file with their local time counterpart, as the difference is expressed in a "logging.timestamp" system property.*

Filtering Log Messages

If your custom logback configuration needs to suppress specific log messages, you can use the `MessageContainsEvaluator` filter. This is useful, for example, to prevent recurring warning messages (such as "Incompatible or syntactically incorrect IP addresses") from filling your log files.

Add the following filter block inside each `<appender>` element in your custom logback XML file where you want to suppress the message:

```
<filter class="ch.qos.logback.core.filter.EvaluatorFilter">
  <evaluator class="com.flexnet.glservice.logging.MessageContainsEvaluator">
    <expression>Incompatible or syntactically incorrect IP addresses</expression>
  </evaluator>
  <OnMatch>DENY</OnMatch>
```

```
  <OnMismatch>NEUTRAL</OnMismatch>
</filter>
```

Replace the `<expression>` value with the text of the message you want to suppress. The filter matches any log message that contains the specified text.

To see how this filter is used in context, you can extract the sample logback XML files included in the `flexnetls.jar` file (located in the `server` directory). For example:

```
unzip -p flexnetls.jar BOOT-INF/classes/logback-rollover.xml
```

Note ▪ *If you are upgrading from a version prior to 2026.06 and your custom logback configuration uses the older Janino-based `EvaluatorFilter` syntax (with `return message.contains("...");`), you must update to the new syntax shown above. The older Janino JaninoEventEvaluator is no longer supported.*

Integration of License Server Logging With External Systems

The local license server can pass log entries to external log monitoring software. Some common configurations are documented here, but others are possible by adapting these instructions.

Common configurations described in this section are:

- Graylog
- Elastic Stack
- logz.io

Examples for implementing some of these configurations are provided in the Example Configurations section.

Graylog

The local license server supports Graylog's GELF protocol (see graylog.org/features/gelf). GELF mode is enabled by supplying the following configuration values in `producer-settings.xml`:

- **graylog.host**—The host name of the Graylog server to which logging messages are sent.

  The format for `graylog.host` is: `[protocol:]host[:port]`

  The `protocol` and `port` components are optional, and default to "udp" and 12201, respectively. The supported protocol values are "udp" and "tcp".

- **graylog.threshold**—The lowest level of log-message granularity to record—`FATAL`, `ERROR`, `WARN`, `INFO`, `LICENSING`, `POLICY`, or `DEBUG`. For example, if `FATAL` is set, only messages about fatal events are recorded. However, if `WARN` is set, fatal-event, error, and warning messages are recorded. (Default is `WARN`.)

Instead of using `producer-settings.xml`, these values can also be supplied using the FlexNet License Server Administrator—see Managing License Server Policy Settings in the Using the FlexNet License Server Administrator Command-line Tool chapter.)

The section Graylog Logging shows an example of how you can implement this configuration.

Elastic Stack

To use Elastic Stack logging, you must configure the server for JSON logging. To do so, in the `local-configuration.yaml` file, specify the following property and value:

```
loggingStyle: JSON_ROLLOVER
```

The Elastic Stack consists of 3 components:

- **Logstash**. This component receives log entries, filters and transforms them and then passes the entries onwards.
- **Elasticsearch**. This component stores the log entries.
- **Kibana**. This component is the GUI where log entries can be searched and visualized.

The following methods are available to pass the log entries to Logstash:

- **Using Logstash's native support of the GELF protocol**—Set up the `graylog.host` and `graylog.threshold` configuration properties to refer to Logstash.
- **Using Elastic Stack's Filebeat to monitor the server's logging directory**—The use of Filebeat to send log entries to the Elastic stack is generally preferred to the use of GELF, because the JSON output is richer and more flexible than the GELF log entry encapsulation, and it is more robust when network interruptions happen. For more information about Filebeat, go to elastic.co/beats/filebeat.

The sections Elastic Stack and Filebeat and Elastic Stack and GELF show examples of how you can implement this configuration.

logz.io

logz.io is a managed and customized Elastic Stack logging offering. Refer to the methods in the previous section, Elastic Stack, for a short explanation of how to configure the server to work with it. For more information about logz.io, go to logz.io.

Example Configurations

This section contains examples for external logging configurations.

- Graylog Logging
- Elastic Stack and Filebeat
- Elastic Stack and GELF

Graylog Logging

The following example demonstrates Graylog logging.

Task ***To enable Graylog logging***

1. Start Graylog. You could run Graylog using Docker or run it as a a virtual machine appliance under VirtualBox, depending on your preference.
2. Set `graylog.host` to `localhost` to specify the Graylog server and set your desired log threshold (for example, `INFO` or `LICENSING`). If using the FlexNet License Server Administrator, you would use a command such as the following:

   ```
   flexnetlsadmin -authorize admin -passwordConsoleInput -config -set graylog.host=localhost
       graylog.threshold=LICENSING
   ```

   Instead of using the FlexNet License Server Administrator you could also request a new `producer-settings.xml` file from your software producer.
3. Restart the license server. Log information will now be sent to Graylog on port 12201 using UDP (the default port and protocol).
4. Configure Graylog to receive the log information. For information, consult the Graylog documentation.

Elastic Stack and Filebeat

This section explains how to log to a Docker installation of the Elastic Stack (Elasticsearch, Logstash and Kibana), using Filebeat to send log contents to the stack. Log data is persisted in a Docker volume called "monitoring-data".

This configuration enables you to experiment with local license server logging to the Elastic Stack. However, it is not recommended for production use, due to insufficient high availability, backup and indexing functionality.

This section is split into the following steps:

1. Preparing the Directory Structure
2. Preparing the Local License Server
3. Creating the “docker-compose” File
4. Creating the Elasticsearch Files
5. Adding a Logstash Configuration
6. Building the Elastic Stack
7. Preparing to Use Filebeat
8. Sending Log Entries to Logstash

Preparing the Directory Structure

This demo uses the following directory structure:

Directory structure

```
demo/
  |
  ----- docker-compose.yml
  |
  ----- elasticsearch/
  |       |
  |       ---- Dockerfile
  |       |
  |       ---- elasticsearch.yml
  |
  ----- filebeat/
  |
  ----- server/
  |       |
  |       --- flexnetls.jar
  |       |
  |       --- producer-settings.xml
  |       |
  |       --- local-configuration.yaml
  |
  ----- logstash.conf
```

Preparing the Local License Server

Follow the steps below to prepare the server to log to a Docker installation. This step assumes that Docker is installed.

Task ***To prepare the local license server for logging to a Docker installation:***

1. Copy the local license server files—`flexnetls.jar`, `producer-settings.xml`, and `local-configuration.yaml`—into the `server` directory.
2. Configure the server for logging in JSON format. Add the following entry to the top section of `local-configuration.yaml`:

   ```
   loggingStyle: JSON_ROLLOVER
   ```

   This causes logs to be written in JSON format to the default location of `$HOME/flexnetls/$PUBLISHER/logs`.
3. Start the license server using the following command from the `server` directory:

   ```
   $ java -jar flexnetls.jar
   ```
4. Confirm that log files with a .json extension are being created.

Creating the “docker-compose” File

Use the sample below to create the `docker-compose.yml` file. Note that this demo uses the producer name “acme”.

docker-compose.yml

```
version: '2.2'
services:
  elasticsearch:
    build:
      context: elasticsearch/
    container_name: elasticsearch
    volumes:
      - monitoring-data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xmx512m -Xms512m"
  logstash:
    image: docker.elastic.co/logstash/logstash:7.9.1
    container_name: logstash
    ports:
      - "5000:5000"
      - "5044:5044"
      - "12201:12201/udp"
    expose:
      - "5044/tcp"
      - "12201/udp"
    logging:
      driver: "json-file"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    volumes:
      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
    depends_on:
      - elasticsearch
  kibana:
    image: docker.elastic.co/kibana/kibana:7.9.1
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
volumes:
  monitoring-data:
    driver: local
```

Creating the Elasticsearch Files

Create the following files in the elasticsearch directory.

elasticsearch/Dockerfile

```
FROM elasticsearch:7.9.1

COPY ./elasticsearch.yml /usr/share/elasticsearch/config/elasticsearch.yml

# FileRealm user account, useful for startup polling.
RUN bin/elasticsearch-users useradd -r superuser -p esuser admin

RUN yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm \
    && yum update -y \
    && yum install -y jq \
    && yum upgrade
```

elasticsearch/elasticsearch.yml

```
cluster.name: "docker-cluster"
network.host: 0.0.0.0

# minimum_master_nodes need to be explicitly set when bound on a public IP
# set to 1 to allow single node clusters
# Details: https://github.com/elastic/elasticsearch/pull/17288
discovery.zen.minimum_master_nodes: 1

## Use single node discovery in order to disable production mode and avoid bootstrap checks
## see https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap-checks.html
discovery.type: single-node

node.data : true
discovery.seed_hosts : []
```

Adding a Logstash Configuration

Add a Logstash configuration such as the following to the `demo` directory:

logstash.conf

```
input {
  beats {
    port => 5044
  }
}

output {
    stdout { codec => rubydebug }

    if ( "lls-logs" in [tags] ) {
        elasticsearch {
            hosts => ["elasticsearch:9200"]
            id => "lls-logs"
            index => "lls-logs-%{+YYYY.MM.dd}"
            codec => "json"
        }
    }
}
```

Building the Elastic Stack

You can now build the Elastic Stack by executing this command in the `demo` directory:

```
$ docker-compose build
```

Preparing to Use Filebeat

Download and expand Filebeat into the `filebeat` directory. This tool will send JSON log entries to the Elastic Stack. You can obtain a copy from www.elastic.co/downloads/beats/filebeat. On Linux you can also use your package manager (DEB, RPM) to install Filebeat. Using the package manager will install it as a system service with systemd bindings. In this case the configuration can be found in `/etc/filebeat/`.

The Filebeat distribution contains an example `filebeat.yml` file. Replace it with this:

filebeat/filebeat.yml

```
filebeat.registry.path: ${HOME}/.filebeat-registry

filebeat.config:
  modules:
    path: ${path.config}/modules.d/*.yml
    reload.enabled: false

filebeat.inputs:
- type: log
  json.keys_under_root: true
  json.overwrite_keys: true
  json.add_error_key: true
  encoding: utf-8
  tags: ["lls-logs"]
  index : "%{[agent.name]}-lls-%{+yyyy.MM.dd}"
  paths:
    - ${HOME}/flexnetls/acme/logs/*.json

output.logstash:
  hosts: ["localhost:5044"]
```

Note ▪ *This sample uses the producer name "acme". In your file, replace "acme" with your producer name, as found in `producer-settings.xml`.*

Sending Log Entries to Logstash

Perform the steps below to log entries to Logstash. You can then view the entries in Kibana.

Task ***To send log entries to Logstash***

1. Bring up the Elastic Stack by executing this command from the demo directory:

   ```
   $ docker-compose up -d
   ```

2. Start the license server (if it is not already running).
3. Start Filebeat to start sending log entries to Logstash:

   ```
   $ ./filebeat -e
   ```

4. Open Kibana with a browser by going to http://localhost:5601. In Kibana's home page, click **Connect to your Elasticsearch index**.
5. Create an index pattern of 'lls-logs-*'. When asked, set '@timestamp' as the primary time field. For information on index patterns, see www.elastic.co/guide/en/kibana/current/tutorial-define-index.html.
6. Click **Discover** (in the grid menu); this should display license server log entries. Consult the Kibana documentation for information about searching the log entries.

Elastic Stack and GELF

The docker-compose configuration is the same as in section Creating the "docker-compose" File, but with a change to the Logstash part (note that the log `entrypoint` key would appear on a single line):

Elastic Stack docker-compose example 2

```
version: '2.2'
services:
  elasticsearch:
    build:
      context: elasticsearch/
    container_name: elasticsearch
    volumes:
      - monitoring-data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xmx512m -Xms512m"
  logstash:
    image: docker.elastic.co/logstash/logstash:7.9.1
    container_name: logstash
    ports:
      - "5000:5000"
      - "5044:5044"
      - "12201:12201/udp"
    expose:
      - "5044/tcp"
      - "12201/udp"
    logging:
      driver: "json-file"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    entrypoint: logstash -e 'input { gelf { } } output { elasticsearch { id => "lls" hosts => ["http:/
/elasticsearch:9200"] } }'
    depends_on:
      - elasticsearch
  kibana:
    image: docker.elastic.co/kibana/kibana:7.9.1
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
volumes:
  monitoring-data:
    driver: local
```

The Filebeat agent is not required in this scenario.

D

Reference: Special Characters Allowed in User Password

When you create an enterprise user account for the license server, the password must include at least one special character (see also Credential Requirements).

This reference lists the special characters that are allowed.

Allowed General Special Characters for User Password

This section lists the special characters supported for Windows and Linux. Special characters that are allowed when the password is enclosed in double (“) or single (‘) quotes are listed separately.

- Special Characters on Windows
- Special Characters on Linux
- Combined Special Character Support (Windows + Linux)

Special Characters on Windows

The following tables list the special characters that are allowed or not allowed in the user password on Windows:

- Windows: Special characters allowed in admin password (quotes not required)
- Windows: Special characters allowed in admin password (quotes required)
- Windows: Special characters not allowed in admin password

Table D-1 ▪ Windows: Special characters allowed in admin password (quotes not required)

Character	Name
!	Exclamation mark

Table D-1 ▪ Windows: Special characters allowed in admin password (quotes not required)

Character	Name
@	At symbol
#	Hash / Pound sign
%	Percent sign
*	Asterisk
-	Hyphen / Minus
_	Underscore
=	Equals sign
+	Plus sign
[	Left square bracket
]	Right square bracket
:	Colon
,	Comma
.	Period / Full stop
/	Forward slash
?	Question mark
~	Tilde
\	Backslash

Table D-2 ▪ Windows: Special characters allowed in admin password (quotes required)

Character	Name
$	Dollar sign
(	Left parenthesis
)	Right parenthesis
{	Left curly brace

Table D-2 ▪ Windows: Special characters allowed in admin password (quotes required)

Character	Name
}	Right curly brace
;	Semicolon
‘	Left single quotation mark. This must be passed in double quotes (").
`	Back tick / Grave accent
'	Apostrophe / Right single quotation mark. This must be passed in double quotes (").

Table D-3 ▪ Windows: Special characters not allowed in admin password

Character	Name
^	Caret
&	Ampersand
"	Double quote
<	Less-than sign
>	Greater-than sign
|	Vertical bar / Pipe

Special Characters on Linux

The following tables list the special characters that are allowed or not allowed in the user password on Linux:

- Linux: Special characters allowed in user password (quotes not required)
- Linux: Special characters allowed in user password (quotes required)

Table D-4 ▪ Linux: Special characters allowed in user password (quotes not required)

Character	Name
@	At symbol
#	Hash / Pound sign
%	Percent sign

Table D-4 ▪ Linux: Special characters allowed in user password (quotes not required)

Character	Name
^	Caret
*	Asterisk
-	Hyphen / Minus
_	Underscore
+	Plus sign
=	Equals sign
[	Left square bracket
]	Right square bracket
{	Left curly brace
}	Right curly brace
:	Colon
,	Comma
.	Period / Full stop
/	Forward slash
?	Question mark
~	Tilde

Table D-5 ▪ Linux: Special characters allowed in user password (quotes required)

Character	Name
!	Exclamation mark
$	Dollar sign
&	Ampersand
(	Left parenthesis
)	Right parenthesis

Table D-5 ▪ Linux: Special characters allowed in user password (quotes required)

Character	Name
|	Vertical bar / Pipe
\	Backslash
;	Semicolon
<	Less-than sign. Must be passed in single quotes (') or double quotes (").
>	Greater-than sign. Must be passed in single quotes (') or double quotes (").
`	Back tick / Grave accent. Must be passed in single quotes (') or double quotes (").
'	Apostrophe / Single quote. Must be passed in double quotes (").
"	Quotation mark / Double quote. Must be passed in single quotes (').

Combined Special Character Support (Windows + Linux)

The following tables list all special characters that are allowed or not allowed in the user password on Windows and Linux:

- Windows and Linux: Special characters allowed in user password (quotes not required)
- Windows and Linux: Special characters allowed in user password (quotes required)
- Windows and Linux: Special characters not allowed in user password

Table D-6 ▪ Windows and Linux: Special characters allowed in user password (quotes not required)

Character	Name
@	At symbol
#	Hash / Pound sign
%	Percent sign
*	Asterisk
-	Hyphen / Minus
_	Underscore
=	Equals sign

Table D-6 ▪ Windows and Linux: Special characters allowed in user password (quotes not required)

Character	Name
+	Plus sign
[	Left square bracket
]	Right square bracket
:	Colon
,	Comma
.	Period / Full stop
/	Forward slash
?	Question mark
~	Tilde

Table D-7 ▪ Windows and Linux: Special characters allowed in user password (quotes required)

Character	Name
!	Exclamation mark
$	Dollar sign
(	Left parenthesis
)	Right parenthesis
{	Left curly brace
}	Right curly brace
;	Semicolon
\	Backslash

Table D-8 ▪ Windows and Linux: Special characters not allowed in user password

Character	Name
^	Caret

Table D-8 ▪ Windows and Linux: Special characters not allowed in user password

Character	Name
&	Ampersand
"	Double quote
<	Less-than sign
>	Greater-than sign
|	Vertical bar / Pipe

Allowed Unicode Characters for User Name

The following tables list all unicode characters that are allowed in the user name:

- Latin-1 Symbols Allowed in User Names
- Latin-1 Maths Symbols Allowed in User Names
- Punctuation Allowed in User Names
- Currency Symbols Allowed in User Names

Table D-9 ▪ Latin-1 Symbols Allowed in User Names

Code	Symbol	Description
U+00A1	¡	Inverted exclamation mark
U+00A2	¢	Cent sign
U+00A3	£	Pound sign
U+00A4	¤	Currency sign
U+00A5	¥	Yen sign
U+00A6	¦	Broken bar
U+00A7	§	Section sign
U+00A8	¨	Diaeresis
U+00A9	©	Copyright sign
U+00AA	ª	Feminine Ordinal Indicator
U+00AB	«	Left-pointing double angle quotation mark

Table D-9 ▪ Latin-1 Symbols Allowed in User Names

Code	Symbol	Description
U+00AC	¬	Not sign
U+00AD		Soft hyphen
U+00AE	®	Registered sign
U+00AF	¯	Macron
U+00B0	°	Degree symbol
U+00B1	±	Plus-minus sign
U+00B2	²	Superscript two
U+00B3	³	Superscript three
U+00B4	´	Acute accent
U+00B5	µ	Micro sign
U+00B6	¶	Pilcrow sign
U+00B7	·	Middle dot
U+00B8	¸	Cedilla
U+00B9	¹	Superscript one
U+00BA	º	Masculine ordinal indicator
U+00BB	»	Right-pointing double-angle quotation mark
U+00BC	¼	Vulgar fraction one quarter
U+00BD	½	Vulgar fraction one half
U+00BE	¾	Vulgar fraction three quarters
U+00BF	¿	Inverted question mark

Table D-10 ▪ Latin-1 Maths Symbols Allowed in User Names

Code	Symbol	Description
U+007D	×	Multiplication sign

Table D-10 ▪ Latin-1 Maths Symbols Allowed in User Names

Code	Symbol	Description
U+00F7	÷	Division sign

Table D-11 ▪ Punctuation Allowed in User Names

Code	Symbol	Description
U+2013	–	En dash
U+2014	—	Em dash
U+2015	―	Horizontal bar
U+2017	‗	Double low line
U+2018	‘	Left single quotation mark
U+2019	’	Right single quotation mark
U+201A	‚	Single low-9 quotation mark
U+201B	‛	Single high reversed-9 quotation mark
U+201C	“	Left double quotation mark
U+201D	”	Right double quotation mark
U+201E	„	Double low-9 quotation mark
U+2020	†	Dagger
U+2021	‡	Double dagger
U+2022	▪	Bullet
U+2026	…	Horizontal ellipsis
U+2030	‰	Per mille sign
U+2032	′	Prime
U+2033	″	Double prime
U+2039	‹	Single left-pointing angle quotation mark
U+203A	›	Single right-pointing angle quotation mark

Table D-11 ▪ Punctuation Allowed in User Names

Code	Symbol	Description
U+203C	‼	Double exclamation mark
U+203E	‾	Overline
U+2044	⁄	Fraction slash
U+204A	⁊	Tironian sign et

Table D-12 ▪ Currency Symbols Allowed in User Names

Code	Symbol	Description
U+20A0	₠	Euro Currency
U+20A1	₡	Colon
U+20A2	₢	Cruzeiro
U+20A3	₣	French Franc
U+20A4	₤	Lira
U+20A5	₥	Mill
U+20A6	₦	Naira
U+20A7	₧	Peseta
U+20A8	₨	Rupee
U+20A9	₩	Won
U+20AA	₪	New Sheqel
U+20AB	₫	Dong
U+20AC	€	Euro
U+20AD	₭	Kip
U+20AE	₮	Tugrik
U+20AF	₯	Drachma
U+20B0	₰	German Penny

Table D-12 ▪ Currency Symbols Allowed in User Names

Code	Symbol	Description
U+20B1	₱	Peso
U+20B2	₲	Guarani
U+20B3	₳	Austral
U+20B4	₴	Hryvnia
U+20B5	₵	Cedi
U+20B6	₶	Livre Tournois
U+20B7	₷	Spesmilo
U+20B8	₸	Tenge
U+20B9	₹	Indian Rupee
U+20BA	₺	Turkish Lira
U+20BB	₻	Nordic Mark
U+20BC	₼	Manat
U+20BD	₽	Ruble
U+20BE	₾	Lari

E

Reference: Local License Server H2 Database Migration 1.4.x to 2.x

The migration procedure discussed in this appendix is intended for customers who have an existing local license server installation that uses an H2 1.4.x database (release 2026.06 and earlier) and want to preserve their data when moving to a release that uses H2 2.x (2026.07 and later).

Customers performing a new installation with a new database do not need to perform this migration.

Appendix Organization

This appendix is divided into the following sections:

- Local License Server H2 Database Migration Steps provides instructions for migrating both a Windows and a Linux database.
- Local License Server H2 Database Migration Best Practices describes the errors and scenarios that you may encounter when migrating the local license server H2 database from 1.4.x to 2.x, and provides the troubleshooting steps you can use to resolve any issues.

Local License Server H2 Database Migration Steps

Migrating the FlexNet Embedded Local License Server H2 database from 1.4.x to 2.x is a two-step process:

- **Export**—Run export on the old 1.4.x server to generate a migration data file.
- **Import**—Run import on the new 2.x server to load the data into a new database.

Key Concepts Required for Successful Migration

You should understand the following key concepts in order to perform a successful migration.

- Trusted Storage Directory
- Role of the Keystore File

Trusted Storage Directory

The database location is determined at runtime from either:

- `producer-settings.xml` (`server.trustedStorageDir`) or
- Base directory overrides defined in configuration

This directory contains:

- H2 database file (`flexnetls_licenses.mv.db`)
- Keystore file (`flexnetls_licenses.ks`)
- Migration file (`flexnetls-h2-1-4-x-migration-source.data`)

Role of the Keystore File

The `flexnetls_licenses.ks` file:

- Is created when the 1.x service starts.
- Is required to open the encrypted H2 database during export.
- Must be present during import to create/read the new database.

The same `flexnetls_licenses.ks` file must be used for both export and import. It must be copied when migrating to a new machine.

Producer Settings Requirement

The same `producer-settings.xml` must be used for both export and import. This ensures that the migration file can be decrypted successfully.

Import Behavior

Import is **idempotent**: If data already exists, import is skipped safely.

Exporting Your H2 Database

Important ▪ *The database export feature (`-export-database` and `--export-database`) is supported only in FlexNet Embedded Local License Server 2026.06. To migrate an H2 1.4.x database to H2 2.x, you must first upgrade the source license server to version 2026.06 and then perform the export. Exporting directly from releases earlier than 2026.06 is not supported.*

This section includes instructions for exporting a legacy H2 1.4.x trusted-storage database on both Windows and Linux platforms.

Windows Export Procedure

This section includes the steps to export the legacy H2 1.4.x database on a Windows machine to a migration data file.

- This is a **one-time, offline operation**.
- The license server **must be stopped** before exporting.
- The license server must be release **2026.06**.

Important ▪ *Run all commands in a **Command Prompt as an Administrator**.*

The Windows export procedure includes the following steps:

- Step 1: Stop the License Server
- Step 2: Export the Database
- Step 3: Verify the Export

Step 1: Stop the License Server

Task ***To stop the license server:***

1. Run the following commands:

   ```
   flexnetls.bat -stop
   flexnetls.bat -status
   ```

2. The service stops. The status command confirms it is no longer running, which is required for export.

   Note ▪ *In some environments, you may need to remove the existing Windows service before reinstalling or reconfiguring the license server. To delete the service, run:*

   `sc delete <service name>`

Step 2: Export the Database

Task ***To export the database:***

1. Run the following command:

   ```
   flexnetls.bat -export-database
   ```

2. A migration data file is created in the trusted storage directory:

   ```
   flexnetls-h2-1-4-x-migration-source.data
   ```

3. The original database file is renamed as follows:

   ```
   flexnetls_licenses.mv.db
   ```

to

```
flexnetls_licenses.mv-bak.db
```

Step 3: Verify the Export

Task ***To verify the export:***

1. Run the following commands:

   ```
   dir "<trusted-storage-dir>\flexnetls-h2-1-4-x-migration-source.data"
   dir "<trusted-storage-dir>\flexnetls_licenses.mv-bak.db"
   ```

   If both files exist, the export was successful.

2. You can now proceed to the import step: Windows Import Procedure.

Linux Export Procedure

Export is supported only for FlexNet Embedded Local License Server 2026.06.

The Linux export procedure includes the following steps:

- Step 1: Run Export
- Step 2: Verify the Export

Step 1: Run Export

Task ***To export the database:***

1. Run the following command:

   ```
   sudo ./install-systemd.sh --export-database
   ```

2. This command stops the service automatically and runs the export.

Step 2: Verify the Export

Task ***To verify the export:***

1. Run the following command:

   ```
   ls -lh /var/opt/flexnetls/<publisher>/
   ```

2. If both of the following files exist, the export was successful.

   ```
   flexnetls-h2-1-4-x-migration-source.data
   flexnetls_licenses.mv-bak.db
   ```

3. You can now proceed to the import step: Linux Import Procedure.

Importing Your H2 Database

This section includes instructions for importing a legacy H2 1.4.x trusted-storage database from release 2026.06 into a 2.x database (release 2026.07 or later) on both Windows and Linux platforms.

- Windows Import Procedure
- Linux Import Procedure

Windows Import Procedure

The Windows import procedure includes the following steps:

- Step 1: Export from Old Server
- Step 2: Copy Required Files (New Machine Only)
- Step 3: Ensure 2.x Service is Installed
- Step 4: Import the Database
- Step 5: Start the Service
- Step 6: Verify Migration
- Step 7: Verify H2 Database Version

Important ▪ *Run all commands in a* ***Command Prompt as an Administrator****.*

Step 1: Export from Old Server

Task ***To export from the old server:***

Complete the steps in Windows Export Procedure.

Step 2: Copy Required Files (New Machine Only)

Task ***To copy required files:***

1. Copy both of these files from the old machine to the new machine:

   `flexnetls-h2-1-4-x-migration-source.data` (migration file)
   `flexnetls_licenses.ks` (keystore file)

2. Place these files into the following directory on the new machine:

   `C:\Windows\ServiceProfiles\NetworkService\flexnetls\<publisher>\`

   This ensures correct decryption and database initialization.

Step 3: Ensure 2.x Service is Installed

Task ***To ensure the 2.x service is installed:***

1. Run the following command:

   ```
   flexnetls.bat -install
   ```

Step 4: Import the Database

Task ***To import the database:***

1. Run the following command:

   ```
   flexnetls.bat -import-database
   ```

2. This command imports data into a new H2 2.x database. It automatically detects the correct data directory.

Step 5: Start the Service

Task ***To start the service:***

1. Run the following command:

   ```
   flexnetls.bat -start
   ```

Step 6: Verify Migration

Task ***To verify migration:***

1. Check the log by entering the following command:

   ```
   type "<trusted-storage-dir>\logs\h2-2x-migration-import.log"
   ```

2. Perform an optional API check:

   ```
   curl http://localhost:7070/api/1.0/instances
   ```

3. These steps confirm that the migration completed successfully and that the service is operating correctly.

Step 7: Verify H2 Database Version

Note ▪ *Ensure that the server is installed successfully and is running and accessible.*

Starting with the 2026.07 release, the current H2 database version can be verified using the following methods.

Method 1: H2 Version Endpoint

- **Endpoint**: `/api/1.0/h2version`

- **Example**: `http://localhost:7070/api/1.0/h2version`

Method 2: Health Endpoint

The H2 database version is also exposed through the **/api/1.0/health** endpoint.

The version information is available under the `database` section of the health response.

Linux Import Procedure

The Linux import procedure includes the following steps:

- Step 1: Export from Old Server
- Step 2: Import the Database
- Step 3: Verify Migration
- Step 4: Verify H2 Database Version

Step 1: Export from Old Server

Task ***To export from the old server:***

Complete the steps in Linux Export Procedure.

Note ▪ *If an existing license server service is present, remove it before proceeding with the import. Delete the existing license server service files from the installation and configuration directories (for example, under `/opt` and `/etc`) to ensure a clean installation.*

Step 2: Import the Database

Task ***To import the database:***

1. Ensure that the migration file (`flexnetls-h2-1-4-x-migration-source.data`) is located in the trusted storage directory (`/var/opt/flexnetls/<publisher>/`) before running the import.
2. Run the following command:

   ```
   sudo ./install-systemd.sh --import-database
   ```

3. The data is imported and the service starts automatically.

Step 3: Verify Migration

Task ***To verify the migration:***

1. Check the following log file:

   ```
   cat /var/opt/flexnetls/<publisher>/logs/h2-2x-migration-import.log
   ```

2. Check the service by running the following command:

   ```
   sudo systemctl status flexnetls-<publisher>
   ```

3. Verify the migration using the API:

   ```
   ./flexnetlsadmin.sh -server http://localhost:7070/api/1.0/instances/~ -licenses -verbose
   ```

Step 4: Verify H2 Database Version

Note ▪ *Ensure that the server is installed successfully and is running and accessible.*

Starting with the 2026.07 release, the current H2 database version can be verified using the following methods.

Method 1: H2 Version Endpoint

- **Endpoint**: `/api/1.0/h2version`
- **Example**: `http://localhost:7070/api/1.0/h2version`

Method 2: Health Endpoint

The H2 database version is also exposed through the **/api/1.0/health** endpoint.

The version information is available under the `database` section of the health response.

Local License Server H2 Database Migration Best Practices

This section supplements the section Local License Server H2 Database Migration Steps by addressing error conditions and edge cases that may occur during export and import.

Export Scenarios

The following scenarios describe issues that may occur during export on both Windows and Linux systems and how to resolve them. Command examples may differ by platform.

- Scenario: Service Still Running
- Scenario: Export Already Completed
- Scenario: No Database Found

Scenario: Service Still Running

Error

The export fails if the license server is still running and holding an active connection to the database.

Fix

- **On Windows**—Run the following commands:

  ```
  flexnetls.bat -stop
  flexnetls.bat -status
  ```

- **On Linux**—This scenario is not possible in Linux because the following command stops the service and then exports the data:

  ```
  ./install-systemd.sh --export-database :
  ```

Explanation

Export requires exclusive database access. On Linux systems, the export command automatically stops the service when using `install-systemd.sh`.

Scenario: Export Already Completed

Symptoms

- `.mv-bak.db` exists.
- `.mv.db` missing.

Cause

Export likely succeeded already. This indicates that a previous export operation has already renamed the original database.

Fix

- If the migration file (`flexnetls-h2-1-4-x-migration-source.data`) exists, proceed to import.
- If the migration file is missing, do the following:
  - **On Windows**—Rename the backup file (`flexnetls_licenses.mv-bak.db`) back to `flexnetls_licenses.mv.db`, then rerun the export.
  - **On Linux**—Run the following command:

    ```
    mv /var/opt/flexnetls/<publisher>/flexnetls_licenses.mv-bak.db \
       /var/opt/flexnetls/<publisher>/flexnetls_licenses.mv.db
    ```

This command renames `flexnetls_licenses.mv-bak.db` to `flexnetls_licenses.mv.db`.

Now try rerunning export.

Scenario: No Database Found

Cause

- Server never started.
- Wrong directory configured.
- Files deleted or moved.

Fix

- **On Windows**—Verify the trusted storage directory location and ensure that the license server has been started at least once to create the database.
- **On Linux**—Verify the trusted storage directory location and ensure that the license server has been started at least once to create the database.

Scenario: Export File Already Exists

Cause

A migration file (`flexnetls-h2-1-4-x-migration-source.data`) already exists in the trusted storage directory.

Fix

- **On Windows**—Delete the existing migration file (`flexnetls-h2-1-4-x-migration-source.data`) from the trusted storage directory, then rerun the export command.
- **On Linux**—Run the following command:

  ```
  rm /var/opt/flexnetls/<publisher>/flexnetls-h2-1-4-x-migration-source.data
  ```

  which removes the existing migration data file, and then rerun export using the following command:

  ```
  sudo ./install-systemd.sh --export-database
  ```

Scenario: Rename Warnings (Windows Only)

You may encounter rename warnings on Windows in the following scenarios:

- Case A: Original Database File No Longer Present
- Case B: Rename Fails (Permissions)

Case A: Original Database File No Longer Present

The migration file is still valid even if the original database file (`flexnetls_licenses.mv.db`) is no longer present. No corrective action is required.

Case B: Rename Fails (Permissions)

Cause

The system cannot rename the database file due to insufficient permissions or file locks (Windows only).

Fix

Manually rename the database file (`flexnetls_licenses.mv.db`) to `flexnetls_licenses.mv-bak.db`. Ensure that you have sufficient file system permissions to complete the operation.

Import Scenarios

The following scenarios describe issues that may occur during import on both Windows and Linux systems and how to resolve them.

- Scenario: Import Already Completed
- Scenario: Import Failed - Retry From Scratch
- Scenario: Fresh Machine (Windows Only)

Scenario: Import Already Completed

The import process detects that the target database already contains data and skips execution.

Result

- Import is skipped automatically.
- Service starts normally.

Scenario: Import Failed - Retry From Scratch

Steps

1. Stop the license server service.
   - **On Windows**—Stop the license server service.
   - **On Linux**—Run the following command:

     ```
     sudo systemctl stop flexnetls-<publisher>
     ```
2. Delete the newly created H2 2.x database file (`flexnetls_licenses.mv.db`).
   - **On Windows**—Delete the file manually.
   - **On Linux**—Run the following command:

     ```
     rm /var/opt/flexnetls/<publisher>/flexnetls_licenses.mv.db
     ```
3. Rename `flexnetls_licenses.mv-bak.db` back to `flexnetls_licenses.mv.db`.
   - **On Windows**—Rename the file manually.

- On Linux—Run the following command:

```
mv /var/opt/flexnetls/<publisher>/flexnetls_licenses.mv-bak.db \
   /var/opt/flexnetls/<publisher>/flexnetls_licenses.mv.db
```

4. Re-run the import.

Scenario: Fresh Machine (Windows Only)

Cause

On a new machine, the `flexnetls_licenses.ks` file is not created until the service has run at least once. Because the import requires the same `flexnetls_licenses.ks` file used during export, it must be copied from the original system before running the import.

Fix

Copy the `flexnetls_licenses.ks` file from the old machine to the trusted storage directory before running the import.

Troubleshooting

This section identifies the location of log files and describes common issues that you may encounter and how to resolve them.

- Log Locations
- Common Issues

Log Locations

Log files are written to the logging directory configured in `producer-settings.xml`, typically under the trusted storage directory.

Export Log

`h2-migration-export.log`

Import Log

`h2-2x-migration-import.log`

Common Issues

The following common issues are described:

- Export Authentication Failed
- Import File Not Found
- Service Fails After Import

Export Authentication Failed

Cause

The migration file is encrypted using the publisher identity defined in `producer-settings.xml`. If the file used during import differs from the one used during export, authentication will fail.

Fix

Use the same `producer-settings.xml` file for both export and import.

Import File Not Found

Cause

The import process expects the migration file (`flexnetls-h2-1-4-x-migration-source.data`) to be located in the trusted storage directory. If the file is missing from this location, the import will fail.

Fix

Copy the migration file to the trusted storage directory.

Service Fails After Import

Fix

If the service does not start after import, review the `flexnetls.log` file in the trusted storage directory for configuration or runtime errors.

Export File Corrupted

Cause

The migration file (`flexnetls-h2-1-4-x-migration-source.data`) may have been corrupted during transfer between systems.

Fix

Re-run the export on the source system and copy the file again.

Console Output Looks Incorrect (Windows Encoding)

Explanation

Console output on Windows may display garbled characters due to encoding differences (UTF-8 vs system code page). This does not affect functionality. Always refer to the log file for accurate output.