revenera

FlexNet Publisher 2023 R3 (11.19.5)

Programming Reference for License File–Based Licensing

Legal Information

Book Name: FlexNet Publisher 2023 R3 (11.19.5) Programming Reference for License File–Based Licensing

Part Number: FNP-11195-PRLF00

Product Release Date: December 2023

Copyright Notice

Copyright © 2023 Flexera Software

This publication contains proprietary and confidential information and creative works owned by Flexera Software and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera Software is strictly prohibited. Except where expressly provided by Flexera Software in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera Software, must display this notice of copyright and ownership in full.

FlexNet Publisher incorporates software developed by others and redistributed according to license agreements. Copyright notices and licenses for these external libraries are provided in a supplementary document that accompanies this one.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see https://www.revenera.com/legal/intellectual-property.html. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Contents

1 Programming Reference for License File–Based Licensing . . . 15

Components in License File–Based Licensing . . . 17

FlexEnabled Applications . . . 17

Flexible API . . . 18

License File . . . 18

License Server . . . 18

License Server Manager . . . 18

Vendor Daemon . . . 19

License Server Utilities (lmutil) . . . 19

lmreporter . . . 19

Debug Log . . . 20

Report Log . . . 20

Options File . . . 20

Message Based Diagnostics . . . 20

Product Support Resources . . . 21

2 Adding the Client API to an Application . . . 23

Adding Function Calls to the Application . . . 23

Flexible API Example . . . 23

Additional Function Calls Needed for Secure Data Types . . . 25

Locating the License File . . . 26

Rules for Locating a License File . . . 27

Setting the Default License Search Path . . . 27

Defining the License Search Path . . . 28

Using the License Search Path to Specify Multiple License Servers . . . 29

3 License Servers . . . 31

License Server Manager . . . 31

lmadmin . . . 31

lmgrd 32
lmgrd Command-Line Syntax 32
Starting lmgrd 34
Upgrading lmgrd 35
Vendor Daemon 35
Building the Vendor Daemon 36
Upgrading the Vendor Daemon 36
Report Log Files 37

Debug Log Files 37
Debug Log's Verbose Information for lmreporter 37
Message Based Diagnostics Metadata 38
Choosing a License Server Configuration 45
Using a License Search Path to Define Redundant License Servers 45
Using Three-Server Redundancy 46
Overview of Three-Server Redundancy 46
Managing License Servers in this Configuration 48
Configuring a Three-Server Redundant Environment 49
Using Other Capabilities with Three-Server Redundancy 51
Troubleshooting Tips and Limitations 53
Comparing License Search Path Redundancy to Three-Server Redundancy 55

4 Understanding Heartbeats 57
Heartbeat Process 57
Automatic vs. Manual Heartbeats 58
Automatic Heartbeats 58
Controlling Automatic Heartbeat Behavior 59
Manual Heartbeats 60
Controlling Manual Heartbeat Behavior 61
Manual Heartbeat Example 61
License Server Considerations 63

5 Customizing the Vendor Daemon 65
Basic Vendor Variables 65
ls_allow_physical_ethernetid_only 67
ls_allow_tz_override 67
ls_allow_vm 68
ls_a_check_baddate 68

ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade 69
ls_daemon_periodic. 69
ls_enable_verbose_dlog 70
ls_flexid9_hasp4_support 70
ls_incallback. 71
ls_infilter. 71
ls_min_lmremove 71
ls_minimum_user_timeout 71

ls_outfilter 72
ls_secure_comms 74
ls_server_borrow_reread_interval. 75
ls_show_vendor_def 75
ls_support_custom_daemon_select_timeout 75
ls_user_init1. 76
ls_user_init2. 76
ls_user_init3. 76
ls_user_init_attributes 76
ls_vd_shutdown 78
ls_vendor_msg 78
ls_vendor_msg_async 78
ls_vendor_msg_list_size 79
Advanced Vendor Variables 79
ls_a_license_case_sensitive 80
ls_allow_updated_feature_borrow 80
ls_borrow_in. 81
ls_borrow_init. 81
ls_borrow_out 83
ls_borrow_return_early 83
ls_checkout_filter_dup_enable 84
ls_client_hostid_callback 84
ls_client_removal_on_reread 85
ls_do_checkroot (UNIX Only) 86
ls_dump_send_data. 86
ls_hud_hostid_case_sensitive. 87
ls_ignore_msg_encryption 87
ls_message_security 88
ls_no_ipaddress_in_server_cache 88
ls_server_override_client_tcp_timeout. 88

ls_server_retry_cnt_after_tcp_timeout 89
ls_server_retry_intvl_after_tcp_timeout 89
ls_prefer_feature_timeout_over_tcp 90
ls_single_xver_signature 90
ls_ts_recovery 90
ls_use_all_feature_lines 91
ls_use_exclusive_commRev4 91
ls_user_lockfile 91

ls_user_lock (Windows only) 92

6 Creating a Common Vendor Daemon 95

Common Vendor Daemon Terminology 95
Primary Vendor Terminology 96
Secondary Vendor Terminology 96
Common Vendor Terminology 97
Planing Your Common Vendor Daemon Configuration 98
Deciding Participating Vendor Daemons 98
Determining Conflicts 99
Configuring the Common Vendor Daemon Toolkit 99
Preparing the Configuration File 99
Configuring the Toolkit 100
Vendor-specific Toolkit Modifications 100
Configuration File Example 101
Building the Common Vendor Daemon 102
Generating License Files for Your Common Vendor Daemon 102
Common Vendor Daemon License File Example 103
Interactions with the SUPERSEDE Keyword 103
Deploying the Common Vendor Daemon 103
Restoring the Original Primary Vendor Toolkit 104

7 Secure Communication 105

Introduction 105
Enabling Ecomm Functionality 105
Generating the Dynamic libfnpssl Library 106
Limitation 108

8 The License File: Overview 109
License File Format. 109
Hostids for Supported Systems 110
License File Types. 111
Simple, Uncounted License 111
Simple, Expiring License 112
Simple Floating (Counted) License 112
Floating with Three-Server Redundancy. 113
License in a Buffer 114
Sorting Licenses During Initialization 114
9 License File Syntax. 117
General Syntax Description 118
Comment Lines 118
Line Continuation. 118
SERVER Lines 118
VENDOR Line 121
USE_SERVER Line 122
FEATURE and INCREMENT Lines 122
License Pools 125
FEATURE and INCREMENT Example 126
Counted vs. Uncounted 126
Feature Name. 127
Vendor Daemon Name. 127
Feature Version 127
Expiration Date. 128
Number of Licenses. 128
BORROW. 129
CAPACITY 129
DUP_GROUP 129
FLOAT_OK. 130
HOST_BASED 130
HOSTID 131
ISSUED 132
ISSUER 132
MINIMUM 132

NOTICE . . . 132
ONE_TS_OK. . . . 132
OVERDRAFT . . . 133
PLATFORMS . . . 133
SIGN . . . 134
SN . . . 134
START . . . 134
SUITE_DUP_GROUP . . . 134

SUPERSEDE . . . 135
SUPERSEDE_SIGN . . . 136
TS_OK . . . 136
TZ . . . 137
USER_BASED . . . 138
VENDOR_STRING . . . 138
VM_PLATFORMS. . . . 138
asset_info. . . . 139
dist_info . . . 139
sort . . . 140
user_info . . . 140
vendor_info . . . 140
UPGRADE Lines . . . 140
License Pool Considerations . . . 141
PACKAGE Lines . . . 141
PACKAGE Line Example without the OPTIONS Keyword . . . 143
PACKAGE Line Example with OPTIONS=SUITE_RESERVED Keyword. . . . 144
PACKAGE Line Example with the OPTIONS=SUITE Keyword . . . 146

10 License Models. . . . 149
Demo Licensing. . . . 149
Limited Time, Uncounted Demos . . . 149
Minimizing Fraudulent Use of Expiring Versions . . . 150
Limited Functionality Demos. . . . 150
Lenient Licensing: Report Log and OVERDRAFT. . . . 151
Report Log File. . . . 151
OVERDRAFT Detection . . . 152

11 License Signature Configurations . . . 153

Signature Types 153
Signature Configurations 154
Configuration Elements 154
Choosing the Appropriate Configuration 156
Introduction to TRL 156
Choosing a Signature Strength 157
Migrating to Tamper-Resistant Licensing 157

Using the Correct Client Library 158
Enabling TRL in Your FlexEnabled Application 158
Maintaining Backward Compatibility 159
Client Library 159
License File 160
Setting Application Attributes 160

12 Cross-Version License Files 161
Identifying If You Need to Use Cross-Version License Files 161
Implementing Cross-Version License Files 163
Regenerating the License File 164

13 IPv6 Support 167
Capabilities that Support IPv6 167
Testing Considerations 170

14 Hostids 171
Platform-Specific Hostids 171
Using Ethernet Address as a Hostid 173
TPM Hostid 174
Using FlexNet ID Dongles 174
Container ID 175
Special Hostids 175
Virtualization Hostids 177
Cloud-Licensing Hostids 177
Using Composite Hostids 179
Defining a Composite Hostid 180
Initializing a Composite Hostid in Your Application 180

Initializing a Composite Hostid in Your Vendor Daemon . . . 181
Creating a Composite Hostid Utility . . . 182
Providing a Composite Hostid License . . . 183
Multiple Composite Hostids for HOSTID_ETHER . . . 184
Using Vendor-Defined Hostids . . . 184
Defining and Using a Vendor Hostid . . . 185
Test the Vendor-Defined Hostid . . . 186
Additional Steps for Production Use of a Vendor-Defined Hostid Type . . . 187

15 FlexNet Licensing Service . . . 189
Installing the FlexNet Licensing Service on Windows . . . 189
Installing the FlexNet Licensing Service on Linux . . . 190
Installing the FlexNet Licensing Service on OS X . . . 192
Sample Applications . . . 192
installanchorservice.exe . . . 192
uninstallanchorservice.exe . . . 193

16 Software Publisher Utility Programs . . . 195
lmcrypt . . . 195
Error Returns . . . 196

17 License Administration . . . 197
Options File . . . 197
License Administration Tools . . . 198
Efficient Reservation . . . 200

18 Internationalization Support . . . 205
License Files . . . 205
UTF-8 Encoded Files . . . 206
Implementation Considerations . . . 207
UTF-8 Encoding on Windows Platforms . . . 207
Case Folding . . . 207
Options Files . . . 207
UTF-8 Encoded Files . . . 208
Client Library . . . 208
Error Messages . . . 208
Environment Variables . . . 208

Diagnostic Messages . . . 208
Debug Log Output . . . 209
Windows Event Logging . . . 209
Report Log Output . . . 209
License Server Utilities . . . 209
lmreporter . . . 209
LMTOOLS . . . 210

License Finder Runtime Graphical User Interface . . . 210

19 Time Zone Licensing . . . 211
Understanding Time Zone Licensing Basics . . . 211
Implementing Time Zone–Based Licensing . . . 212
Examples . . . 212
Specifying Single and Multiple Time Zones . . . 213
Specifying Time Zone Ranges . . . 213
Specifying Time Zone Ranges and Multiple Time Zones . . . 213
Specifying Half and Quarter Time Zones . . . 213
Accounting for Daylight Saving Time . . . 214
Using the Time Zone of the License Server . . . 214
Using TZ with a Fully Qualified Domain Name (FQDN) . . . 214
Including Cross-Version Signatures . . . 214
Testing Time Zone Licensing . . . 215
Limitations and Interactions with Existing Licensing Functions . . . 216
Three-server Redundancy . . . 216

20 COAVAIL Checkout . . . 219
Introduction . . . 219
Usage Examples . . . 220
Components . . . 220
Preparation . . . 220
Preparing the Vendor Daemon . . . 220
Preparing lmflex_coavail . . . 221
Preparing the License File . . . 221
Example 1: License Checkout Using LM_CO_NOWAIT . . . 222
Example 2: License Checkout Using LM_CO_AVAIL_NOWAIT . . . 223
Error Codes . . . 226

Limitations 226

21 Batch Checkout 229

Introduction 229

Usage Examples 232

Example 1: Simple Batch Checkout 232

Example 2: Batch Checkout with Attributes 233

Example 3: Batch Checkout with 2 Transactions 234

Error Codes 236

Supported Checkout Scenarios 236

Limitation 237

22 Virtualization 239

Understanding Virtualization Features 239

Prohibiting or Allowing Virtualization Support 240

Limitations and Interactions with Existing Licensing Functions 241

Duplicate Grouping using DUP_GROUP 241

Terminal Services Clients: TS_OK and ONE_TS_OK 242

PLATFORMS 242

23 Mobile Licensing 243

Node-Locked to a Laptop Computer 244

Node-Locked to a FlexNet ID Dongle 244

Node-Locked to a FlexNet ID Dongle with FLOAT_OK 244

Using FlexNet ID Dongle with FLOAT_OK Keyword 244

Security Issues for FlexNet ID Dongle with FLOAT_OK 245

Node-locked to a User Name 246

Fulfilled from a Prepaid License Pool 246

License Borrowing using the BORROW Keyword 246

Enabling License Borrowing 247

Support for License Borrowing 247

Security Issues for License Borrowing 248

License Transferring Using the Transferrable Counted Model—Windows Only 249

Enabling License Transferring 249

Support for License Transferring 250

Considerations While Transferring the License 250

Limitation 251

24 FlexNet Publisher Parameter Limits 253

Parameter General Information 253

Internationalization Support 253

Path Name Formats 254

License File Limits 254

Options File Limits 256
lc_set_attr limits 256
Other API Limits. 257
Vendor Daemon Limits 257
Number of Applications Per Vendor Daemon 257
Number of Vendor Daemons Per Node 257
lmadmin (or lmgrd) Limits 258
Subnet, Domain, Wide-Area Network Limits 258
LM_LICENSE_FILE, *VENDOR*_LICENSE_FILE 258

25 Obsolete Features 259
Obsolete Vendor Daemon Variables 259
Obsolete License File Features 260
License Key Length and Start Date 260
Implementing Long License Keys and Embedded Start Dates 261
Compatibility Issues 262
FEATURESET Line 262
Intel Pentium III Hostid (HOSTID_INTEL) 263
lmbind 264
Obsolete Libraries 265
UNIX Libraries for Single-Threaded Applications 265
Non-TRL Libraries 265

Glossary 267

Index 279

1

Programming Reference for License File–Based Licensing

FlexNet Publisher enables software publishers to define and implement licensing strategies for their applications. When implementing a licensing solution, the software publisher must decide how they want to store license rights. FlexNet Publisher allows you to store license rights in either:

- License files
- Trusted storage

This document provides information about how to implement a license model when storing the license rights in license files (called license file–based licensing). The license file contains information that defines the license model for a FlexEnabled application. License files can exist locally with the FlexEnabled application or can reside on a license server that serves licenses to the multiple FlexEnabled applications.

Table 1-1 ▪ *Programming Reference for License File–Based Licensing* Chapter Overview

Section	Content
This section	An overview of the contents of this document.
Adding the Client API to an Application	An overview of the Flexible API, along with coding examples.
License Servers	Describes the systems that run license servers, other utilities and files that make up a license server.
Understanding Heartbeats	Explains the heartbeat mechanism and types of the heartbeat between license server and FlexEnabled applications.
Customizing the Vendor Daemon	How to customize vendor daemon using various vendor variables.
Creating a Common Vendor Daemon	Explains the common vendor daemon terminology, configuration, and deployment.

Table 1-1 ▪ *Programming Reference for License File–Based Licensing* Chapter Overview

Section	Content
Secure Communication	Describes how the secure communication happens via Ecomms.
The License File: Overview	An overview of the license file format and examples of license file types.
License File Syntax	Explains the license file syntax rules, Server information, Package information, and License rights information.
License Models	Defined the types of License models.
License Signature Configurations	Descibes the background of the default license signature mechanism and information on the supported license signature configurations.
Cross-Version License Files	An overview of the Cross-Version license file, its implementation, and regenaration.
IPv6 Support	Describes the support of IPv6 addresses in the license model.
Hostids	An overview of the hostids and their types.
FlexNet Licensing Service	Explains FlexNet Licensing Service and its installation on different platforms.
Software Publisher Utility Programs	Describes about the `lmcrypt` utility.
License Administration	Briefly explains about the option file to customize the software, License administration tools, and efficient reservation for the licence file.
Internationalization Support	Explains the available internationalization support for the areas of FlexNet Publisher involved with end-user interaction.
Time Zone Licensing	Provides detailed information on Time zone licensing basics, implementation, testing, and limitations.
COAVAIL Checkout	Describes the function COAVAIL checkout in detail.
Batch Checkout	Describes the function batch checkout in detail.
Virtualization	An overview of the basics of Virtualization features, its implementation, and limitations.
Mobile Licensing	Explains the various types of mobile licensing.
FlexNet Publisher Parameter Limits	Lists the limitations such as string lengths and unlimited items.
Obsolete Features	Obsolete variables and features are listed here.

Table 1-1 ▪ *Programming Reference for License File–Based Licensing* Chapter Overview

Section	Content
Glossary	Contains a collection of terms and their meanings.

Components in License File–Based Licensing

The following diagram shows the components required to implement license file–based licensing.

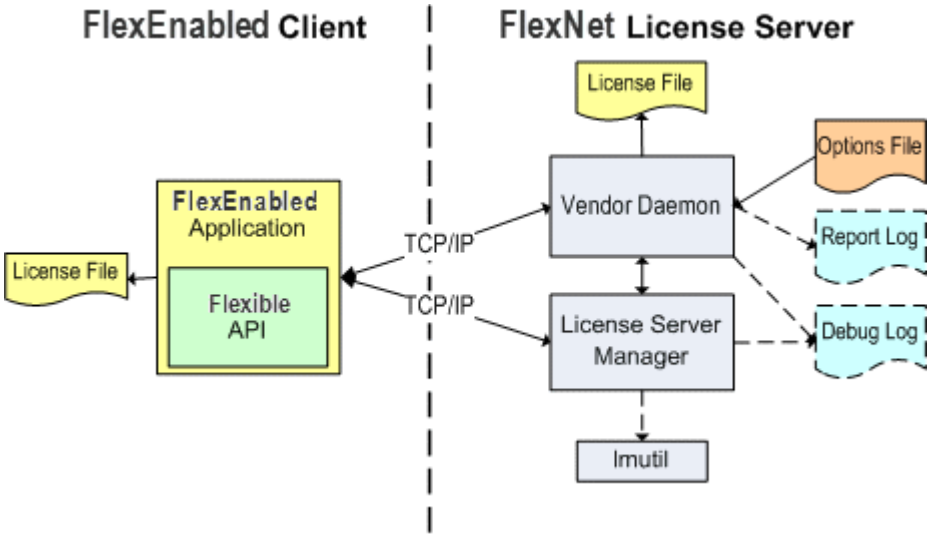


Figure 1-1: Components used in license file–based licensing

FlexEnabled Applications

A FlexEnabled application is a software application that implements the licensing strategy using the FlexNet Publisher components. A FlexEnabled application uses the Flexible API (in the client library) to locate and access license rights for the application.

The application developer uses the function **lc_checkout** to request a feature, which is a discrete unit of product capability. If the feature is available—that is, license rights for that feature are available—the checkout succeeds and the application continues normally. If the feature is unavailable, the application behaves in a publisher-specified way to notify the end user.

Because the application requests licenses at run time (the license rights are contained in the license file and not hard-coded inside your application), you can create a single binary for different editions of your application, and then provide different license files (or define different license models) for various application editions. For example, the premiere-edition license model could enable all application features, while the express-edition license model might enable only a subset of the application features.

Flexible API

The Flexible API is a set of functions available using the FlexNet Publisher Licensing Toolkit client library. These functions check out and check in features in license files or trusted storage. Software publishers add the check-in and check-out functions into the FlexEnabled application so that application functionality is enabled when a feature is successfully checked out.

License File

The license file stores the license rights that a customer has for the application. The license file contains one or more feature definition lines that define the license model for a feature.

The software publisher defines how a feature in the license file maps to the software application. A feature in the license file can represent the entire application or a single capability in the application (for example, a menu item). If the publisher maps a feature in the license file to a subset of capability in the application, they can enable and disable certain capabilities based on the type of license the customer has purchased.

The license file can reside locally with the FlexEnabled application or it can reside on a license server. The software publisher makes this decision when implementing the licensing strategy.

License Server

FlexNet Publisher contains components that enable publishers to build and distribute a license server to their customers. The license server is used when the licensing strategy includes the concurrent license model. The license server manages license rights while FlexEnabled applications connect to the license server to check out and check in features.

A single license server can manage licenses from multiple software publishers. This is because each software publisher builds and ships certain components that are specific to their licensing strategy.

A license server for license file–based licensing can contain the following components:

- License server manager `lmadmin` (or `lmgrd`)
- Vendor daemon
- Debug log
- Report log
- Options file
- License server utilities

The license server must have at least one license server manager (`lmadmin` or `lmgrd`) and vendor daemon to function.

License Server Manager

The license server manager `lmadmin` (or `lmgrd`) is an executable that manages vendor daemons, handles the initial requests from FlexEnabled applications, and handles communication between license servers when they are configured for three-server redundancy.

After the license server manager identifies the appropriate vendor daemon for a FlexEnabled application request, it forwards the request to that vendor daemon. From then on, the vendor daemon and the FlexEnabled application communicate directly.

The license server manager also records messages to the debug log.

Vendor Daemon

The vendor daemon is an executable that is built in such a way that each is specific to a software publisher. It is responsible for managing the license rights for that publisher and handling check-out and check-in requests called from FlexEnabled applications.

The vendor daemon communicates with FlexEnabled applications using the FlexNet communication protocol available in the client library. This communication protocol uses TCP/IP to transmit messages between the vendor daemon and the FlexEnabled application.

When the FlexEnabled application checks out and checks in a feature, the vendor daemon records these actions to the report log. The vendor daemon also writes informational and error messages to the debug log.

License Server Utilities (lmutil)

FlexNet Publisher includes a set of command-line utilities to help license administrators manage licenses and license servers. The utilities are packaged as a single executable called `lmutil`. You run these utilities using the command:

`lmutil <utility name> <options>`

- *<utility name>* is the name of the specific utility you want to execute.
- *<option>* are any command-line options to use with that utility. Each utility takes a different set of options.

The Web-based license server manager available for some platforms, `lmadmin`, includes many of the administrative functions provided by `lmutil`.

For Windows systems, a Windows-based executable called `LMTOOLS` is also available. `LMTOOLS` has the same capability as `lmutil` and also includes additional functionality for Windows-specific administration, such as installing `lmgrd` as a service.

Important ▪ *Always use the newest version of the license server utilities, which are available for download from the Revenera product download site. See the License Administration Guide for information about these utilities*

lmreporter

The lmreporter is an optional user interface that allows to view the feature usage, peaks, and denials. The lmreporter utility uses the debug logs, where the server events are captured, created by the vendor daemon. This functionality must be explicitly enabled by the vendor.

To enable the lmreporter, see the Debug Log's Verbose Information for lmreporter.

For more information on usage of the lmreporter, see the *Using License Administration Tools* chapter in the *License Administration Guide*.

Note ▪ *The lmreporter tool is supported only on the Windows and Linux platforms.*

Debug Log

A debug log file contains status and error messages useful for debugging the license server. A license server always generates debug log output. Some of the debug log output describes events specific to the license server manager and some of the debug log output describes events specific to each vendor daemon.

Report Log

The report log contains information about feature usage and is generated by the vendor daemon. The content of the report log is encrypted and cannot be read by humans. FlexNet Manager, Revenera's license management tool, can read this file so that license administrators can produce license usage reports.

Options File

The options file enables the license administrator to control certain parameters within the limits of a software publisher's licensing policy. The license administrator can:

- Allow or deny features for specific users, hosts, or groups.
- Reserve licenses for specific users, hosts, or groups.
- Restrict the number of licenses available to the enterprise.
- Control the amount of information logged about license usage.
- Enable logging to a report log.

Message Based Diagnostics

The license server emits diagnostic output for every message it receives to a local bound port.

The diagnostics output runs on a separate thread in the VD and is controlled by two mechanisms - both must be enabled for diagnostics to be received:

- By default, the vendor variable `ls_diagnostics_enabled` is enabled.
- A diagnostics listener client (DGLC) - must run on the same machine as the license server and receives diagnostics messages on the locally bound port. Four examples of DGLCs are provided in the `examples/diagnostics` folder:
  - `log2file.c` and `pylog2file.py` demonstrate receiving diagnostic data and writing it to a file.
  - `log2port.c` demonstrates receiving diagnostic data and making it available on a public port - this allows delegation of receipt of diagnostics data to another client on a remote machine.
  - `log2db.c` demonstrates receiving diagnostic data and writing it to a database.

Product Support Resources

The following resources are available to assist you:

- 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 and then making selections on the **Get Support** menu, including **Open New Case** and other options.

2

Adding the Client API to an Application

The FlexEnabled application accesses license rights by calling functions that request (check out) and release (check in) licenses for selected features. These functions are part of the client API.

Two client libraries are available:

- Flexible API—C/ C++ client library
- Java API—Java client library

This chapter provides an overview of the Flexible API, along with coding examples. For information about the Java API, see the *Licensing for Java Programming Reference Guide*.

When you are ready to build the application, link with the client libraries, and test the FlexEnabled application, see "Building Your FlexEnabled Application" in the manual, *Development Environment Guide*.

Adding Function Calls to the Application

This section describes how to modify your application code to call the client functions. The `machind` directory in the toolkit contains a sample FlexEnabled application: `lmflex.c`. This implementation serves as a basis for the examples in this section. The implementation presented produces a minimal, well-behaved FlexEnabled application.

FlexNet Publisher also contains additional examples in the `examples` subdirectory. The `examples` directory contains programs that illustrate how to perform various operations. These programs are intended to be illustrative only and Revenera may not include them in future toolkit releases. Their operation is not covered in this manual.

Flexible API Example

The following is a complete example, using the Flexible API, of the calls needed to produce a minimal, well-behaved FlexEnabled application. This example is derived from `lmflex.c` which is shipped with the toolkit and located in the `machind` directory. Figure 2-1 shows a schematical Flexible API implementation.

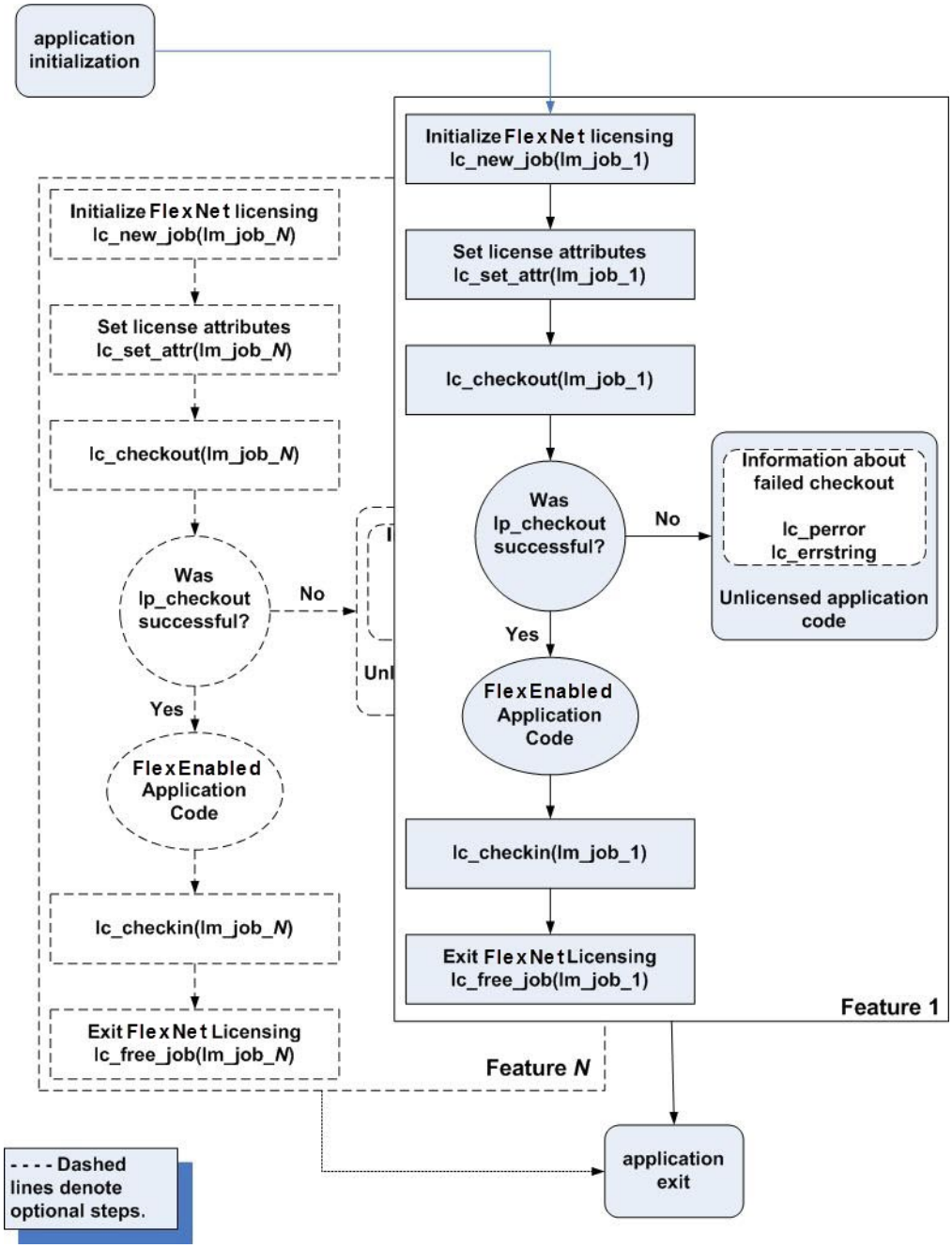


Figure 2-1: Flexible API implementation

Add calls like the following to your application code.

```
#include "lmclient.h"
#include "lm_attr.h"
        /*...*/
        VENDORCODE code;
        LM_HANDLE *lm_job;
        if (lc_new_job(0, lc_new_job_arg2, &code, &lm_job))
        {
            lc_perror(lm_job, "lc_new_job failed");
```

```
            exit(lc_get_errno(lm_job));
        }

/* You define loc for the following call to lc_set_attr() with the expected location
   of the application's license file directory. See Locating the License File for more
   details on locating the license file.*/
        lc_set_attr(lm_job, LM_A_LICENSE_DEFAULT,
            (LM_A_VAL_TYPE)loc);

    if (lc_checkout(lm_job, "myfeature", "1.0", 1,
        LM_CO_NOWAIT, &code, LM_DUP_NONE))
        {
            lc_perror(lm_job, "checkout failed");
            exit (lc_get_errno(lm_job));
        }
/*
  *     Checkout succeeded. Actual application code here
 */

    /*...*/

        lc_checkin(lm_job, "myfeature", 0);
        lc_free_job(lm_job);
```

You need to determine the location of your application's license file directory at run time. Refer to Setting the Default License Search Path for techniques and examples for determining the license file location. See the *C/C++ Function Reference* for a full description of the **lc_checkout** function parameters.

The header file `lmclient.h` contains all the symbolic definitions required for most calls to the API functions. `lm_attr.h` contains the attribute definitions used in calls to **lc_set_attr** and **lc_get_attr**. See the *C/C++ Function Reference* for a complete list of attribute definitions.

Additional Function Calls Needed for Secure Data Types

When implementing Secure Data Types (SDTs) in your application, your application must also call the following initialization functions to load and unload a product-specific run-time component (prepared by `preptool`) that the SDTs require to operate. (For more information about implementing SDTs in your application, see Chapter 9, "Implementing Secure Data Types (SDTs)" in the *Development Environment Guide*.)

Table 2-1 ▪ Initialization Functions

Function Name	Description
lc_flexinit	Loads the prepared run-time component and initializes FlexNet Publisher Licensing Toolkit.
lc_flexinit_cleanup	Unloads the run-time component and performs any cleanup functions required.
lc_flexinit_property_handle_create	Creates a property handle.

Table 2-1 ▪ Initialization Functions

Function Name	Description
lc_flexinit_property_handle_free	Frees an initialization property handle.
lc_flexinit_property_handle_set	Sets a type and value into the initialization property handle.
lc_flexinit_property_handle_get	Returns the value for a specified property.

The use of these routines must be coordinated with the following:

- Secure Data Types (SDTs)—You must call **lc_flexinit** before any SDTs are used in the code.
- Job management—You must call **lc_flexinit** before the call to **lc_new_job** and call **lc_flexinit_cleanup** after the call to **lc_free_job**.

In multi-threaded environments you must ensure synchronization between threads. The following are areas where particular care is needed:

- Executable exit logic—Your code must explicitly check in all licenses, using **lc_checkin**, and free all jobs, using **lm_free_job**, before calling **lc_flexinit_cleanup** and exiting. Do not rely on the automatic check-in mechanism to perform proper cleanup upon exit.
- Lost connection to the license server—When a concurrent license is in use and the connection to the license server breaks, the code must explicitly check in all licenses before calling **lc_flexinit_cleanup**.

Locating the License File

The *most important* part of integrating FlexNet Publisher into your application is correctly setting and finding the location of your application's license file relative to the application's installation location. This location should not be hard-coded, but should be determined at run time, relative to your application's installation location.

In order for a FlexEnabled application to check out a feature, it must first locate the license file that contains the license model for that feature. Specifying a license file directory (containing license files with a `.lic` extension) with **lc_set_attr** and the attribute `LM_A_LICENSE_DEFAULT` is the recommended way for your application to find one or more valid license files. This enables you to specify more than one license file without setting the license search path as an environment variable. See example UNIX and Windows functions to locate your application's license file directory, based on the application's installation location, in Setting the Default License Search Path.

Important ▪ *Do not choose a location for a license file where the path to the license file contains the @ symbol when installing a license file. Advise your users that they should not locate license files in a folder with an @ symbol in its path.*

Rules for Locating a License File

FlexEnabled applications use the following rules for choosing a license file to use:

1. If a license search path is set using either the *VENDOR*_LICENSE_FILE or LM_LICENSE_FILE (where *VENDOR* is the vendor daemon name) environment variable, it is used. *VENDOR*_LICENSE_FILE is used first. *VENDOR*_LICENSE_FILE is used only by products from *VENDOR* and by lmutil and LMTOOLS.
2. The environment variables *VENDOR*_LICENSE_FILE and/or LM_LICENSE_FILE can also be set in the Windows registry or, on UNIX, in $HOME/.flexlmrc. The Windows registry is in HKEY_CURRENT_USER\SOFTWARE\FLEXlm License Manager. *VENDOR*_LICENSE_FILE is also automatically set upon successful checkout.
3. License server utilities accept the argument -c license_file_list, which specifies the license paths. If this is set, the environment variables are ignored.
4. If none of the other locations is specified, then the following default location is used. Relying on this default location is not recommended.
   - On UNIX, the default location is /usr/local/flexlm/licenses/license.dat.
   - On Windows, the default location is C:\flexlm\license.dat.

Important ▪ *In practice, use of the default location is discouraged because it is not searched if the application specifies a different default, or if the user has a license-path environment variable set. For example, the default location, which may have successfully worked for an application, fails once the environment variable LM_LICENSE_FILE is set (unless LM_LICENSE_FILE includes the default location).*

Setting the Default License Search Path

The default license search path is set using the license_file_list parameter of the FlexEnabled application's call to **lc_set_attr** with the LM_A_LICENSE_DEFAULT attribute.

Specifying a directory (containing one or more license files with a .lic extension) as the license search path is the recommended way for your application to find one or more valid license files. This allows the application to internally specify more than one license file without defining external configuration variables.

The following UNIX function is an example that are useful to help you write a function to locate your application's license file path, based on the application's installation location.

UNIX Example

```
char *unix_example_get_dir_loc(void)
{
  char *prodhome;

/*      Substitute PRODUCT_HOME with the environment variable the application uses to indicate
 *      the directory where the application was actually installed. If you don't use an environment
 *      variable like this, then see examples/advanced/get_dir.c
 */
    if (prodhome = getenv("PRODUCT_HOME"))
        return prodhome;
    else return 0; /* see get_dir.c for another approach */
}
```

Defining the License Search Path

A *license search path* specifies one or more files, directories, or license servers where the FlexEnabled application can search for the feature used in the checkout call. FlexEnabled applications process the license files specified in a license search path in a specified order. For example:

`setenv VENDOR_LICENSE_FILE file1:file2:..:dir1:..:filen`

- If the FlexEnabled application runs on UNIX or Mac, separate each entry in the license search path by colons.
- If the FlexEnabled application runs on Windows, separate each entry by semicolons.

The FlexEnabled application tries *file1* and if it fails, it looks in *file2*, and so on. Directories are automatically expanded to use all files matching `*.lic` in that directory as part of the list.

The license search path is stored in the following location:

- Windows registry: `HKEY_CURRENT_USER\SOFTWARE\FLEXlm License Manager`
- UNIX file path: `$HOME/.flexlmrc`

The license search path can also define three servers configured for three-server redundancy. See Using Three-Server Redundancy for more information about this capability.

The license search path can consist of the following:

- A directory, `dir`, where files `dir/*.lic` are searched in alphabetical order (according to the rules of the operating system).
  - On Windows, file names are not case sensitive. All file names are converted to uppercase before being searched, so `*.LIC` files are recognized.
  - On UNIX, file names are case sensitive: `*.LIC` files are not recognized.
- A single file, with any extension.
- The port used by the license server. Specify one of the following:
  - *port@host* from the license file's `SERVER` line, where *port* is the port number and *host* is the host name of a license server.
  - *@host* from the license file's `SERVER` line, where *host* is the host name of a license server. No port number is specified, or a default port number (between 27000 and 27009) is being used.
  - `@localhost` if the server is running on the same system as the FlexEnabled application.
  - A list of three *port@host* specifiers denoting a three-server redundant configuration. For example, `port1@host1,port2@host2,port3@host3` specifies the three-server redundant configuration composed of host1, host2, and host3.
- The actual text of the license file, with `START_LICENSE\n` as a prefix, and `\nEND_LICENSE` as suffix, where the embedded new lines are required. See License in a Buffer for more details on this feature.

Using the License Search Path to Specify Multiple License Servers

Aside from being convenient, license search paths enable you to distribute license rights across multiple license servers, which has many advantages over the more formal three-server redundancy. See Choosing a License Server Configuration for more information about using multiple license servers and three-server redundancy.

3

License Servers

A license server includes the license server manager `lmadmin` (or `lmgrd`) and one or more vendor daemon processes. The term *license server* refers to these processes, not the computer on which they run.

If you ship counted or served licenses, your user must run a license server. Therefore, ship your customers a copy of the latest version of `lmadmin` (or `lmgrd`) available from Revenera, along with your customized vendor daemon. This chapter discusses the systems that run license servers and other utilities and files that make up a license server.

License Server Manager

The license server manager `lmadmin` (or `lmgrd`):

- Starts and maintains all the vendor daemons listed in the `VENDOR` lines of the license file.
- Refers a checkout (or other) request to the correct vendor daemon.
- Establishes and maintains communications between a set of license servers configured for three-server redundancy.

lmadmin

The `lmadmin` license server has a Web-based, administrative interface. Using the `lmadmin` user interface, the license administrator can do the following:

- Import existing license files.
- Launch the license server without requiring any command-line options.
- Perform all license server configuration and most administration functions from the interface.
- Keep configuration options persistent—if you change settings and relaunch the tool, the previously set options are overridden.

The current FlexNet Publisher *Release Notes* lists the platforms for which `lmadmin` is available.

The *License Administration Guide* describes how to use `lmadmin`. The `lmadmin` interface also contains online help.

lmgrd

`lmgrd` is an executable with a few options that are set using command-line arguments. The license file location and a few other parameters can be set by the license administrator in this way. These options are described in the following section.

lmgrd Command-Line Syntax

When you invoke `lmgrd`, it looks for a license file that contains information about vendors and features.

Usage

```
lmgrd [-c license_file_list] [-l [+]debug_log_path]
      [-2 -p] [-local] [-x lmdown] [-x lmremove] [-z] [-v] [-datestamp] [-m value] [-help]
```

The following table describes the syntax elements:

Table 3-1 ▪ `lmgrd` Command-Line Syntax

Term	Description
-c *`license_file_list`*	Uses the specified license files.
-l [+]*`debug_log_path`*	Writes debugging information to file *`debug_log_path`*. This option uses the letter `l`, not the numeral 1. Prepending *`debug_log_path`* with the + character appends logging entries. Use the `-l` option before other options to log all debugging information to *`debug_log_path`*. See Debug Log File for more information on this file. ***Note ▪*** *On Windows, it is best practice to set the location of `debug_log_path` to a ProgramData sub-folder, since this location has user-write permission by default, which is needed when the license server runs as a service with LocalService permission.*

Table 3-1 ▪ `lmgrd` Command-Line Syntax

Term	Description
-2 -p	All platforms: `-2 -p` is effective, and supported, only when used together with `-local`. On UNIX systems, `-2 -p` restricts usage of `lmdown`, `lmreread`, and `lmremove`—as well as `lmswitch`, `lmswitchr`, and `lmnewlog`—to a license administrator who is by default root. If there is a UNIX group called `lmadmin`, then use is restricted to only members of that group. If root is not a member of this group, then root does not have permission to use any of the above utilities. On Windows systems, if `lmgrd` is started with `-2 -p -local`, `lmgrd` and the vendor daemon can only interact with the command-line utilities (`lmreread`, `lmnewlog`, `lmdown`, `lmremove`, and `lmswitch`) if these are located on the same machine and if they are run with LOCALSYSTEM privileges. ***Note ▪*** *It is recommended to start your lmgrd with -2 -p -local. This protects FlexNet Publisher from attack that involve remote, unauthenticated access to server commands.*
-local	On UNIX systems, `-local` restricts the `lmdown` and `lmreread` commands to be run only from the same system where `lmgrd` is running. On Windows systems, if `-local` is used on its own, `lmutil` (and variations) can be run by any Local User as long as they are running `lmutil` from the same host as the license server. If `lmgrd` is started with `-2 -p -local`, `lmgrd` and the vendor daemon can only interact with the command-line utilities (`lmreread`, `lmnewlog`, `lmdown`, `lmremove`, and `lmswitch`) if these are located on the same machine and if they are run with LOCALSYSTEM privileges. All platforms: In the case of a three-server setup, any node in the triad is considered to be local.
-x lmdown	Disables the `lmdown` command (no user can run `lmdown`). If `lmdown` is disabled, stop `lmgrd` using `kill pid` (UNIX), or stop the `lmgrd` and vendor daemon processes through the Windows Task Manager or Windows service. On UNIX, be sure the `kill` command does not have a `-9` argument.
-x lmremove	Disables the `lmremove` command (no user can run `lmremove`).
-z	Runs in foreground. The default behavior is to run in the background. If `-l debug_log_path` is present, then no windows are used, but if no `-l` argument specified, separate windows are used for `lmgrd` and each vendor daemon.
-v	Displays the `lmgrd` version number and copyright and exits.

Table 3-1 ▪ `lmgrd` Command-Line Syntax

Term	Description
-d	(OS X/macOS only) Makes `lmgrd` and the vendor daemon OS X/macOS launchd compliant. Use the reference file `demo.plist` (included in the OS X/macOS toolkit) for information on how to instruct launchd.
-reuseaddr	[optional] Allows the server to explicitly bind to a same port, which remains in TIME_WAIT state after the server restart or crash ***Note ▪*** *It is recommended to use the* `-reuseaddr` *option only on non-Windows systems to avoid undefined behavior.*
-datestamp	[optional] Enables writing the date, timestamp, and time zone offset in the debug log files.
-m *`value`*	Enforces obfuscation of communication between client and server. Valid values: ● `disabled`: Message obfuscation is disabled. The license server ignores the obfuscation mechanism for the communication. ● `permissive`: Message obfuscation is enforced only if it is supported by the client. ● `targeted`: Message obfuscation is enforced only for the targeted messages that share the license file information with the clients. Requests sent by clients without message obfuscation are rejected. Communication with utilities must be obfuscated. ● `enforcing`: Message obfuscation is enforced on all new style messages. ***Important ▪*** *If the argument* `targeted` *or* `enforcing` *is set, FlexNet Publisher clients or utilities prior to version 11.19.2 (2022 R3) can not communicate with the license server because they don't support the obfuscation mechanism.*
-help	Displays usage information and exits.

Note ▪ *The license search path can also be specified by setting the environment variable LM_LICENSE_FILE to the file's path name. The -c path specification will override the setting of LM_LICENSE_FILE.*

Starting lmgrd

The most common command line for `lmgrd` is:

```
lmgrd -c license_file_list -l [+]debug_log_path [-2 -p]
[-x lmdown|lmremove]
```

If the `license_file_list` value contains more than one license file, each must be separated by colons on UNIX or semicolons on Windows. If a directory is specified, all `*.lic` files in that directory are read.

On UNIX Platforms

On UNIX systems, it is recommended that `lmgrd` be run as a non-privileged user (not `root`).

On Windows Platforms

The recommended method of running `lmgrd` is to install it and run it as a service.

Two available methods are:

- Use the Web-based application, `LMTOOLS`, to configure `lmgrd` to run as a service after installation.
- Use the command-line utility, `installs.exe`, to install `lmgrd` as a Windows service.

Both are located in the `<platform_dir>` directory of the toolkit installation. Using `LMTOOLS` to install `lmgrd` as a service is the recommended technique. See the *License Administration Guide* for more information.

`lmgrd` can also be started as an application from a Windows command shell. For example:

```
C:\flexlm> lmgrd -c <vendor>.lic
```

The problem with running a license server this way is that it occupies a window on the screen, and may be difficult to start and stop.

Note ▪ *When using* `installs.exe` *to start lmgrd as a Windows service, the* `-c` *parameter accepts a maximum of 30 license files of up to 255 characters in length. It is best practice to set the location of debug_log_path to a ProgramData sub-folder, since this location has user-write permission by default, which is needed when the license server runs as a service with LocalService permission*

Upgrading lmgrd

Recommend to license administrators that they should always run the latest version of `lmgrd`, even if your vendor daemon and FlexEnabled application are not built with the newest version. The current version of `lmgrd` is available from the Revenera product download site.

Vendor Daemon

The vendor daemon is an executable that is unique for each software publisher. It is responsible for responding to check-out and check-in requests from FlexEnabled applications. To build your own vendor daemon, you must supply the following information:

- Your LM_SEEDs
- Your production vendor keys
- The name of your vendor daemon, for example `VENDOR_NAME` in `lm_code.h`

- Your TRL keys
- `LM_STRENGTH` setting

Although normally not required, your vendor daemon may require customization by editing `lsvendor.c`.

The `makefile` creates `lm_new.o` on UNIX (`lm_new.obj` on Windows) and then builds your vendor daemon, `lmflex` (the sample FlexEnabled application), and the license generator (`lmcrypt`). One of the functions of `lm_new.o` is to remove the vendor name and LM_SEEDs from the executables; they are constructed at run time.

See the document titled *Development Environment Guide* for specific information about building the vendor daemon and other required components.

Building the Vendor Daemon

On UNIX Systems

Task ***To build the vendor daemon***

At a command prompt from the `<platform_dir>` directory, enter the following command:

```
make -f makefile
```

The command window displays a list of files being built. The `make` command builds `lm_new.o`, which contains security information from `lm_code.h` and is used to build your application, your vendor daemon, as well as the sample applications. If you have edited only `lm_code.h`, the `make` command rebuilds everything except your application.

On Windows Systems

Task ***To build the vendor daemon***

At a command prompt from the `<platform_dir>` directory, enter the following command:

```
nmake -f makefile
```

The command window displays a list of files being built. The `nmake` command builds `lm_new.o`, which contains security information from `lm_code.h` and is used to build your application, your vendor daemon, as well as the sample applications. If you have edited only `lm_code.h`, the `nmake` command rebuilds everything except your application.

Upgrading the Vendor Daemon

Improvements are made in each release of FlexNet Publisher. Therefore, it is recommended that you distribute an upgraded vendor daemon to your customers soon after each version of FlexNet Publisher is released.

Upgrading your vendor daemon is a simple process. You need only rebuild it using the FlexNet Publisher components. You do not have to rebuild the FlexEnabled application.

To find the current version of your vendor daemon, at a command prompt from the `<platform_dir>` directory type:

```
<vendor> -v
```

or

```
lmutil lmver <vendor>
```

where <*vendor*> is the name of your publisher-specific vendor daemon.

Task ***To rebuild the vendor daemon:***

1. Download and install the latest version of FlexNet Publisher.
2. Open `/machind/lm_code.h` and edit it to match your previous `lm_code.h` settings:
   a. Replace the `VENDOR_KEY`s with the production vendor key lines that you received from Revenera.
   b. Enter `LM_SEED`s (three 32-bit numbers that you make up).
   c. Change the `VENDOR_NAME` to your vendor name.
   d. Add the two `TRL_KEY`s that you received from Revenera and define `LM_STRENGTH` to specify the length of your license signature. For information about `LM_STRENGTH` values, see Choosing a Signature Strength.

   Remember to keep the `lm_code.h` file and the encryption seeds secret.
3. Rebuild the product toolkit with the edited `lm_code.h`. This rebuilds your vendor daemon, as well as other files in the toolkit.
4. Provide a convenient way for your customer to get your new vendor daemon.

Report Log Files

A vendor daemon can write license usage information to a report log. The license server manager does not write data to a report log. The content of the report log output is not human-readable and is used only by FlexNet Manager. You must configure each vendor daemon separately to record data to a report log, it does not write report log output by default.

The *License Administration Guide* provides instructions for how to enable and manage report log files.

Debug Log Files

A license server generates debug log output to a debug log file. Some of the debug log output describes events specific to the license server manager; and some of the debug log output describes events specific to each vendor daemon. The *License Administration Guide* describes how to manage debug log files and provides a description of the messages that are written by the daemons.

Debug Log's Verbose Information for lmreporter

The Verbose information from the debug log files are used by the lmreporter tool. This must be explicitly enabled by the vendor.

Task ***To enable verbose information in the debug log files:***

1. Download and install the latest version of FlexNet Publisher.
2. Open `/machine/lsvendor.c` and set the vendor variable:
   a. Set `ls_enable_verbose_dlog` to 1.
   b. Change `int ls_enable_verbose_dlog = 0` to `int ls_enable_verbose_dlog = 1.`
3. Rebuild the product toolkit with the edited `lsvendor.c`. This rebuilds your vendor daemon, as well as other files in the toolkit.
4. Provide a convenient way for your customer to get your new vendor daemon.

Note ▪ *For more information on the `ls_enable_verbose_dlog` vendor variable, see the ls_enable_verbose_dlog.*

Message Based Diagnostics Metadata

Message-based diagnostics emits a row of comma-delimited diagnostics data for every message received by the vendor daemon. The fields emitted for each row are:

Table 3-2 ▪ Message Based Diagnostic Metadata

Diagnostics Field	Description	Always Present
Arrival Time	UTC timestamp.	Yes
Message Type	FNP protocol message name.	Yes
Latency	Time taken, in milliseconds, for the VD to process the message.	Yes
User	Username sent in the message.	Yes
HostId	HostID sent in the message.	`LM_CHECKOUT` messages may have this info.
Display	Display info sent in the message.	Messages generated from an `lc_checkout` call will typically have this info.
VendorDef	Vendor defined data from custom dup grouping.	`LM_CHECKOUT` messages will have this info if custom dup grouping is being used.
Source	IP address of the message source	Yes

Diagnostics Field	Description	Always Present
Process ID	Process ID of the process that sent the message.	Yes
Session ID	Session ID of the client.	Only from clients and utilities with version >= 11.17.0.
Component	Value is one of the following: ● Name of an `lc_*` API call that is responsible for the message type, when `lc_diagnostics_start/end` does not wrap that lc_* call. ● Any `lc_*` API calls wrapped by `lc_diagnostics_start/end` will have Component=USER. ● Name of a FNP standalone component - currently one of LMGRD or LMSTAT.	Only from clients and utilities with version >= 11.17.0.
Info	Info provides additional information about the Component. If an `lc_*` API call, Info is a unique sequence number. If `lc_diagnostics_start` (unsigned int value) is used, the hexadecimal of value. If lmstat is used, Info is currently one of the following: ● `Default` - None of those below ● `ALL_WITH_USER_INFO` - lmstat -a ● `ALL_NO_USER_INFO` - lmstat -a --no-userinfo ● `ONLY_ONE_FEAT_WITH_USER_INFO` - lmstat -f feature_name ● `ONLY_ONE_FEAT_NO_USER_INFO` - lmstat - f feature_name --no-userinfo	Only from clients and utilities with version >= 11.17.0

Note ▪ *The diagnostics metadata may change in the future. If any changes to diagnostics metadata are done, those will be notified via release notes.*

Message based diagnostics examples

lmstat

`lmstat -a -counted.lic`, followed by `lmstat -a --no-userinfo -c counted.lic`, from a remote Windows machine:

Arrival Time	Message Type	Latency	User	Host ID	Display	Vendor Ref	Source	Process ID	Session ID	Component	Info
2020-03-06T15:26:12.377161Z	LM_HELLO	0.081	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:54837	51848	0	-	-
2020-03-06T15:26:12.378524Z	FLEX_MSG_SERVER_VERSION_REQUEST	0.061	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:54837	51848	40800002	LMSTAT	ALL_WITH_USER_INFO
2020-03-06T15:26:12.379596Z	FLEX_MSG_VD_GET_FLIST_TYPE	0.066	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:54837	51848	40800002	LMSTAT	ALL_WITH_USER_INFO
2020-03-06T15:26:12.380453Z	FLEX_MSG_FULFILL_STATUS	0.068	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:54837	51848	40800002	LMSTAT	ALL_WITH_USER_INFO
2020-03-06T15:26:12.381551Z	FLEX_MSG_LIST	0.08	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:54837	51848	40800002	LMSTAT	ALL_WITH_USER_INFO
2020-03-06T15:29:45.226692Z	LM_HELLO	0.071	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:56401	55272	0	-	-

Arrival Time	Message Type	Latency	User	Host ID	Display	Vendor Ref	Source	Process ID	Session ID	Component	Info
2020-03-06T15:29:45.227783Z	FLEX_MSG_VD_GET_FLIST_TYPE	0.056	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:56401	55272	40800003	LMSTAT	ALL_NO_USER_INFO
2020-03-06T15:29:45.228644Z	FLEX_MSG_FULFILL_STATUS	0.058	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:56401	55272	40800003	LMSTAT	ALL_NO_USER_INFO
2020-03-06T15:29:45.229492Z	FLEX_MSG_VD_FEATINFO_HANDLE_TYPE	0.072	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:56401	55272	40800003	LMSTAT	ALL_NO_USER_INFO

lc_checkout - Linux client

Arrival Time	Message Type	Latency	User	Host ID	Display	Vendor Ref	Source	Process ID	Session ID	Component	Info
2020-02-28T18:28:46.322645Z	LM_HELLO	0.075	Test User	-	/dev/pts/2	-	::ffff:10.48.150.134:51956	21141	0	-	-
2020-02-28T18:28:46.322795Z	FLEX_MSG_HEARTBEAT	0.079	Test User	-	/dev/pts/2	-	::ffff:10.48.150.134:51956	21141	83800003	lc_checkout	3
2020-02-28T18:28:46.322919Z	FLEX_MSG_COMMREV4_HANDSHAKE	0.068	Test User	-	/dev/pts/2	-	::ffff:10.48.150.134:51956	21141	83800003	lc_checkout	3
2020-02-28T18:28:46.323137Z	LM_CHECKOUT	0.136	Test User	a41f727cc29e	/dev/pts/2	-	::ffff:10.48.150.134:51956	21141	83800003	lc_checkout	3
2020-02-28T18:28:46.323515Z	FLEX_MSG_VD_VERIFY_KEY_V1	0.082	Test User	a41f727cc29e	/dev/pts/2	-	::ffff:10.48.150.134:51956	21141	83800003	lc_checkout	3

DUP GROUP checkout - Windows client

Bulk custom dup group checkout and incremental checkin (note SessionID is different for each `lc_*` call)

```
lc_set_attr(lm_job, LM_A_MULTIPLE_CHECKOUT_DATA, "abc\fxyz");

lc_checkout(lm_job, FEATURE, "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR)

lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "abc");

lc_checkin(lm_job, FEATURE, 0);

lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "xyz");
```

```
lc_checkin(lm_job, FEATURE, 0);
```

Arrival Time	Message Type	Latency	User	Host ID	Display	Vendor Ref	Source	Process ID	Session ID	Component	Info
2020-03-06T16:57:56.192099Z	FLEX_MSG_HEARTBEAT	0.053	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:63497	42336	83800007	lc_checkout	7
2020-03-06T16:57:56.192888Z	FLEX_MSG_COMMREV4_HANDSHAKE	0.079	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:63497	42336	83800007	lc_checkout	7
2020-03-06T16:57:56.193855Z	LM_CHECKOUT	0.16	Test User	-	MDH-NT-211897	abc\014xyz	::ffff:10.48.41.236:63497	42336	83800007	lc_checkout	7
2020-03-06T16:57:56.194780Z	FLEX_MSG_VD_VERIFY_KEY_V1	0.043	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:63497	42336	83800007	lc_checkout	7
2020-03-06T16:57:56.243617Z	LM_CHECKIN	0.07	Test User	-	MDH-NT-211897	abc	::ffff:10.48.41.236:63497	42336	83800009	lc_checkin	9
2020-03-06T16:57:56.244458Z	LM_CHECKIN	0.072	Test User	-	MDH-NT-211897	xyz	::ffff:10.48.41.236:63497	42336	83800009	lc_checkin	00000b

User-defined Info and SessionID

```
lc_diagnostics_start(10);
```

```
lc_checkout(lm_job, FEATURE, "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE));
```

```
lc_checkin(lm_job, FEATURE, 0);
```

```
lc_diagnostics_end();
```

Arrival Time	Message Type	Latency	User	Host ID	Display	Vendor Ref	Source	Process ID	Session ID	Component	Info
2020-03-06T16:44:16.157427Z	FLEX_MSG_HEARTBEAT	0.056	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:61397	107552	2000000a	USER	0000000a
2020-03-06T16:44:16.158241Z	FLEX_MSG_COMMREV4_HANDSHAKE	0.068	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:61397	107552	2000000a	USER	0000000a
2020-03-06T16:44:16.159395Z	LM_CHECKOUT	0.122	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:61397	107552	2000000a	USER	0000000a
2020-03-06T16:44:16.160963Z	FLEX_MSG_VD_VERIFY_KEY_V1	0.08	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:61397	107552	2000000a	USER	0000000a
2020-03-06T16:44:16.170815Z	LM_CHECKIN	0.089	Test User	-	MDH-NT-211897	-	::ffff:10.48.41.236:61397	107552	2000000a	USER	0000000a

Diagnostics Filtering

Filtering can occur pre or post message-based diagnostics is persisted. The former kind of filtering might typically occur in order to save disk space, whereas the latter would typically occur when searching for the source of capacity overload in the server.

An example of pre-persistence filtering is available in the `log2db.c` sample. The command line `./log2db -i -d diag.db COMPONENT=^lc_check` will only write records (aka diagnostic message rows) to the database that match the `COMPONENT=^lc_check` regular expression - that is, only rows corresponding to messages generated from **lc_checkout** or **lc_checkin** API calls.

Filtering can occur on any fields or on any combination of fields.

Choosing a License Server Configuration

FlexNet Publisher supports serving license rights from:

- -A single license server
- Multiple license servers each serving a subset of the total available licenses.
- Multiple license servers configured for three-server redundancy.

If license files are stored on a network file server (separate from the license server), do not implement three-server redundancy. You lose the failover protection because the file server is a potential point of failure. If you choose this configuration, it is recommended that you use a single license server to manage all vendor daemons.

If the licenses are split among two or more license servers and the license administrator wants licenses to be available when a license server goes down or is taken off of the network, then multiple license servers can be deployed.

In all cases, an effort should be made to select stable systems. Do not pick systems that are frequently rebooted or shut down. It is not required that each system be the same architecture or operating system.

Using Multiple License Servers

A license administrator can configure license servers in such a way to allow FlexEnabled applications to continue to check out licenses if one of the license servers goes down. There are two mechanisms available to accomplish this.

- **Redundancy using the license search path**: Configure and maintain multiple independent license servers, each with a subset of the total licenses available to the enterprise. Configure the FlexEnabled client with the license servers in the license search path. This provides load balancing capabilities and limited failover protection. License administrators must manage different versions of the license rights on each license server.
- **Three-server redundancy**: Configure and maintain a set of three license servers on separate systems configured specifically for three-server redundancy. This provides failover protection only. License administrators manage only one version of the license file and vendor daemon on all three license servers.

Using a License Search Path to Define Redundant License Servers

It is recommended that you use the function **lc_set_attr** to set the `LM_A_LICENSE_DEFAULT` attribute to a license file directory (containing license files with a `.lic` extension). The FlexEnabled application should call this function to define the location of valid license files. The license files are tried in alphabetical order. This method allows you to specify more than one license file without setting the license search path to an environment variable.

You can define multiple license servers in the license search path so that if a FlexEnabled application cannot connect to one of the license servers, it will try the next one in the list. This is implemented by setting the `LM_LICENSE_FILE` or *`VENDOR`*`_LICENSE_FILE` environment variable.

This type of license server redundancy is best explained by example. If ten licenses are desired for both features f1 and f2, you would issue your customer two license files with a count of 5 for each of f1 and f2, each keyed to a different license server. The license servers can be physically distant.

The license files would look like this:

- License 1 for “chicago”

```
SERVER chicago 17007ea8 1700
VENDOR xyzd /etc/mydaemon
FEATURE f1 xyzd 1.000 01-jan-2020 5 SIGN=”...“
FEATURE f2 xyzd 1.000 01-jan-2020 5 SIGN=”...”
```

- License 2 for “tokyo”

```
SERVER tokyo 17a07e08 1700
VENDOR xyzd /etc/mydaemon
FEATURE f1 xyzd 1.000 01-jan-2020 5 SIGN=16BE40E1DAEE
FEATURE f2 xyzd 1.000 01-jan-2020 5 SIGN=6DB6F3E40E61
```

The user in Chicago could set `LM_LICENSE_FILE` or *`VENDOR`*`_LICENSE_FILE` to this:

```
1700@chicago:1700@tokyo
```

The user in Tokyo could set `LM_LICENSE_FILE` or *`VENDOR`*`_LICENSE_FILE` to this:

```
1700@tokyo:1700@chicago
```

The FlexEnabled application attempts to check out licenses from the first license server in the license search path. If that fails for any reason, it tries the next license server. This type of redundancy can be used to balance the usage of licenses across several license servers.

Using Three-Server Redundancy

Software publishers can enable their enterprise software customers to configure three-server redundancy for failover protection. Three-server redundancy is a specific capability available in FlexNet Publisher Licensing Toolkit that provides failover protection while preventing license rights from being improperly replicated.

In this configuration, only one license server supplies licenses to FlexEnabled applications. The other two are available in case that one fails. When at least two of the three license servers are running and communicating, the system serves licenses to FlexEnabled applications.

In order to implement three-server redundancy, both the software publisher and the license administrator must perform certain steps.

Note ▪ *Three-server redundancy is supported with license file–based licensing only. It is not supported with trusted storage–based licensing.*

See the FlexNet Publisher *Release Notes* for information about the supported third-party software allowed for license servers.

Overview of Three-Server Redundancy

Using the three-server redundancy capability in FlexNet Publisher, all three license servers operate to form a triad. The license servers send periodic messages to each other to ensure that at least two servers are running and communicating with each other. A *quorum* is formed when at least two of the three license servers are running and communicating with each other.

The license servers are identified as primary, secondary, or tertiary. One license server is also designated as the master and is responsible for:

- Serving licenses to FlexEnabled applications
- Recording information to the debug log.
- Recording information to the report log.

If the master fails, then another license server becomes the master.

In the following figure, the primary license server is the master [m]. When a FlexEnabled application sends a checkout request for a license, the master responds and then serves the license to the FlexEnabled application.

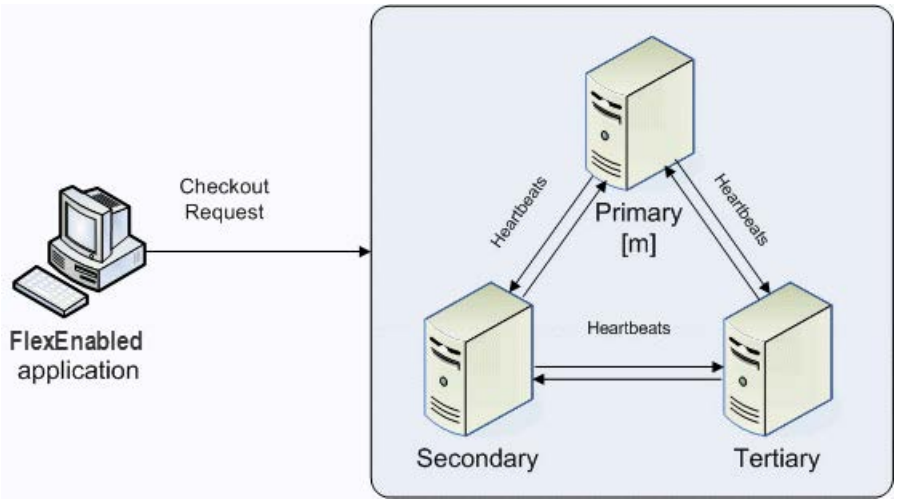


Figure 3-1: Three-server redundancy overview

If the master license server fails, then the secondary license server becomes the master (as shown in Figure 3-2) and will serve licenses to FlexEnabled applications. The tertiary license server can never be the master. If both the primary and secondary license servers go down, licenses are no longer served to FlexEnabled applications. The master will not serve licenses unless there is a quorum of at least two license servers in the triad running and communicating.

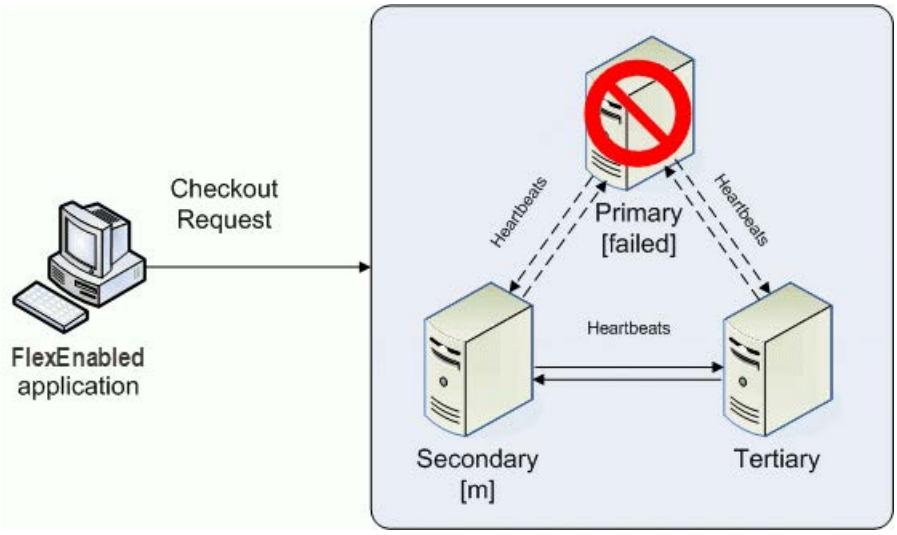


Figure 3-2: Three-server redundancy backup failover

Understanding How License Servers Communicate

When started, each license server reads the license file and checks that it can communicate with the other license servers. Until each license server establishes this first connection with the others, it will continue to establish a connection by sending messages periodically.

When the initial communication has been established, each license server periodically sends a heartbeat to the others. Heartbeats are messages sent over TCP/IP. Each license server sends a heartbeat and waits for a response from the other license servers. If a license server does not receive a response, it shuts down the vendor daemon so that it cannot serve licenses. A publisher or license administrator can configure the amount of time a license server waits to receive a heartbeat using the `HEARTBEAT_INTERVAL` property.

Poor network communication causes system performance to slow.Slow network communication can also cause a delay in the transmission of heartbeats between license servers.

Understanding Why Three License Servers Are Required

When configured for three-server redundancy, the license server manager `lmadmin` (or `lmgrd`) requires that the configuration include three license servers and that at least two of them communicate with each other. Ensure that the license servers are in the same subnet so that they can communicate. If the master cannot communicate with at least one other license server, it will stop serving licenses.

This prevents an enterprise from putting each license server on a different subnet and having different users connect to each so that they can use more licenses than they have been issued.

Managing License Servers in this Configuration

Using the lmstat Utility

The output message generated by the `lmstat` utility identifies which license server is the master. In the following example `lmstat` output, the secondary license server is the master.

```
[Detecting lmgrd processes...]
```

```
License server status: 30000@RMD-PRIMARY,30000@RMD-SECONDARY,
30000@RMD-TERTIARY
License file(s) on RMD-PRIMARY: C:\server\3.lic:
RMD-PRIMARY: license server UP v11.4
RMD-SECONDARY: license server UP (MASTER) v11.4
RMD-TERTIARY: license server UP v11.4
```

Starting and Stopping License Servers

To start the entire system, you must start each license server manager (`lmadmin` or `lmgrd`). Generally, it is good practice to start the primary license server before the secondary or tertiary license server. This allows the primary license server to become the master before the others start. If you start the secondary and tertiary before the primary, then the secondary will establish itself as master.

If you do not set the `PRIMARY_IS_MASTER` keyword for the primary license server, then the order in which you start the license servers is important. If you do not set this property, when you start the primary license server after the secondary license server, control will not transfer to the primary license server. By setting the `PRIMARY_IS_MASTER` keyword, you ensure that when the primary license server is running, it is always the master.

In three-server scenarios, a system administrator may wish to shut down one of the servers to perform maintenance on it, while the other two servers maintain a quorum and continue to serve licenses. Previously, shutting down a license server in a triad through `lmdown` (or `lmtools`) would shut down the entire triad. Now, a system admin can run `lmdown -c 3server-license.lic` from one of the machines in the triad. `lmdown` will read the 3-server license file and provide a prompt asking which of the three listed servers to shut down.

Running lmadmin or lmgrd as a Service on Windows

There are no dependencies or known issues related to running `lmadmin` or `lmgrd` as a service in this configuration.

Logging and the Debug Log

When using three-server redundancy, the master records information to its local debug log and report log (and the Windows event log if this is configured). If this system fails, another license server becomes the master and records information to its local debug log and report log. Subsequently, there may be different versions of the debug log and report log on the primary and secondary license server which each contain different information.

Configuring a Three-Server Redundant Environment

Both the software publisher and the license administrator must perform certain configuration steps. This section describes the steps that each must perform.

Special Note About hostid Types

Best practice is to require the license administrator to obtain a hostid *of the same type* for each of the three machines. Issues can arise if different hostid types are used in the configuration. If license administrator plans to obtain the hostid using the default hostid type for a given machine, be aware that the default type can vary between operating-system platforms.

Configuration for Software Publishers

A publisher enables three-server redundancy by defining three `SERVER` lines in the license file, one for each of the license servers. Each license server is identified as primary, secondary, or tertiary by the order of the `SERVER` lines.

- The first SERVER line defines the primary license server.
- The second SERVER line defines the secondary license server.
- The third SERVER line defines the tertiary license server.

When the publisher signs the license file, the hostid values in the SERVER lines are computed into the signature of each feature definition line. The master will serve only those features with signatures that encode those hostid values.

The steps to configure this capability are as follows:

1. Create the license file and include three SERVER lines, one for each of the license servers. Each line contains the following:
   - Host name of the system
   - Hostid of the system (see Special Note About hostid Types)
   - Port number that the license server manager uses to listen for communication (the license administrator can also change this)
   - PRIMARY_IS_MASTER keyword, optionally (the license administrator can also change this)
   - HEARTBEAT_INTERVAL property, optionally (the license administrator can also change this)
2. Define the remainder of the license file including the VENDOR line and any FEATURE, INCREMENT, UPGRADE, or PACKAGE lines.
3. Sign the license file using the lmcrypt utility. The SIGN value for each line binds the feature to the three hostid values. The three license servers will serve these licenses to FlexEnabled applications per the license model defined in this file.

For more information about these properties, see the section Using License File Keywords.

Configuration for License Administrators

The license administrators should perform the following steps:

1. Before obtaining the license server package, identify and set up the three systems. When selecting systems, make sure they are stable. Do not use systems that are frequently rebooted or shut down.
2. Send the publisher the hostname and hostid values for these systems. Ask the publisher what system identifier they need for the hostid. For example, this could be an Ethernet address or disk serial number. The publisher will create license server components specifically for these systems. (See Special Note About hostid Types.)
3. After receiving the license server package from the publisher, change the following SERVER line properties in the license file if necessary:
   - Port number the license server uses to listen for communication
   - PRIMARY_IS_MASTER keyword
   - HEARTBEAT_INTERVAL property

Important ▪ *Do not change the hostid values. If the hostid changes at any time, the license administrator must work with the software publisher to obtain a new license file.*

4. Perform any additional configuration as required by the software publisher.
5. Copy or install the license server software package to each of the three systems.
6. Start the license servers in the following order: primary, secondary, and then tertiary.

An Example License File

The following is an example of a license file that is configured for three-server redundancy.

```
SERVER pat 17003456 2837 PRIMARY_IS_MASTER
SERVER lee 17004355 2837
SERVER terry 17007ea8 2837
VENDOR demo
FEATURE f1 demo 1.0 1-jan-2018 10 SIGN=”...”
FEATURE f2 demo 1.0 1-jan-2018 10 SIGN=”...”
```

The following portions of the license file directly affect the three-server redundant configuration:

- `SERVER` lines: These three lines define each of the systems involved.
  - The host values are: pat, lee, and terry.
  - The hostid values are: 17003456, 17004355, and 17007ea8. This example uses the value returned by the `lmhostid` utility default hostid type.

    Note ▪ *Keep in mind that the default hostid type can vary between operating-system platforms. Best practice is to use the same hostid type for all three servers. (See Special Note About hostid Types.)*
- The TCP/IP ports: All servers use the same port (2837, in the example) to listen for communication.

The following properties of the license file do not affect the three-server redundant configuration directly, but are used to define license rights or configure the license server.

- `VENDOR` line: this is required and references the publisher’s vendor daemon.
- `FEATURE` lines: The two features, f1 and f2, define the license rights. The `SIGN` value for each `FEATURE` line encodes the license server hostid values.

Using Other Capabilities with Three-Server Redundancy

The following section describe other capabilities available in FlexNet Publisher Licensing Toolkit and how they interact with three-server redundancy.

Configuring the FlexEnabled Application

This configuration can be performed by either the software publisher or the license administrator. Before a FlexEnabled application can check out a license, it must know where to locate the license rights. The license search path identifies the location of license rights.

When connecting to a license server configured for three-server redundancy, the FlexEnabled application must use the port@host convention (and not a license file location) in the license search path.

The license search path should list the license servers in the same order that they appear in the license file. This helps shorten the amount of time it takes to identify the master server and respond to the checkout request. The configuration will work if you include only one of the license servers in the license search path. This may lengthen the amount of time it takes for the license server to respond to the checkout request because the license server must identify the other license servers and also identify the master.

You must also separate each port@host entry with a comma and not a semicolon (Windows), colon (UNIX/Mac), or ampersand (Java). The comma indicates that the license servers are configured for three-server redundancy.

Using the previous license file as an example, the license search path should be:

```
2837@pat,2837@lee,2837@terry
```

The FlexEnabled application will try to connect to each of the license servers in the order listed, until it makes a connection. This helps ensure that the FlexEnabled application can connect to the quorum.

Using License File Keywords

The following keywords and properties for the SERVER line enable you to modify the configuration.

- host—The hostname of the system. The publisher should know this information when generating the license file. This value can be changed after the license file has been signed.
- hostid—This value uniquely identifies the system. You can use different types of system information to identify the hostid, such as disk serial number. After the license file has been signed, you cannot change this value without invalidating the signature.
- port—The port number that the license server uses to listen for communication. Unlike with single license servers, each SERVER line must include a port number. This can be any number in the range 1 to 65535 that is not used by another process running on the system. On UNIX, choose a port >1024, since those <1024 are privileged port numbers. If not specified, the license server will automatically use the next available port number in the range 27000 to 27009.

  If you specify a port number greater than 65535, the client fails to establish a connection with the license server manager (lmgrd or lmadmin).

  ***Note** ▪ Those producers concerned about potential DoS attacks based on knowledge of common license server ports may want to consider specifying a server port outside the range 27000 to 27009.*

  This value can be changed after the license file has been signed. For more information on port numbers on the SERVER line, see Table 9-2 on page 118 in Chapter 9, License File Syntax.

  If you are using lmadmin, you do not need to edit the license file; you can configure the port number using the interface. The port number specified in lmadmin takes precedence over any port set in the license file, either before or after the port is specified using lmadmin. (See the online help for more details.) Editing the port number using lmadmin is only recommended if no port number is specified in the license file. If a port number is already specified in the license file and the port is subsequently changed in lmadmin, during the next restart the vendor daemon will exit with a “port mismatch” error.

  To make it easier to administer the license server, it is strongly recommended that you define the same port number for each SERVER line.

- `PRIMARY_IS_MASTER`—This keyword ensures that the primary server is the master whenever it is running and communicating with one of the other license servers.
  - If this is set and the primary server goes down, when the primary server comes back up again, it will always become the master.
  - If this is not set and the primary server goes down, the secondary server becomes the master and remains the master even when the primary server comes back up. The primary can become the master again only if the secondary license server fails.

  This parameter is optional and should be placed on the first `SERVER` line. This value can be changed after the license file has been signed. The license server must be running a version 10.8 or later vendor daemon to use this keyword.

- `HEARTBEAT_INTERVAL=seconds`—This indicates how long the license servers wait to receive a heartbeat from another license server before shutting down the vendor daemon. This value is used in the following equation to calculate the actual timeout value:

  ```
  timeout = (3 * seconds) + (seconds – 1)
  ```

  Valid timeout value is 0-120. The default value is 20, which equates to an actual timeout of 79 seconds. Valid seconds values are 0 through 30. This parameter is optional and should be placed on the first `SERVER` line in the license file. This value can be changed after the license file has been signed. The license server must be running a version 10.8 or later vendor daemon to use this keyword.

  Note ▪ *When the seconds value exceeds 30, `Lmadmin` displays the Heartbeat Interval value as -1 along with the an error message "`Invalid three server redundancy configuration, valid timeout values are 0 - 120`".*

Using Vendor Variables

None of the vendor daemon variables is used specifically for three-server redundancy.

Using the Flexible API

None of the FlexNet Publisher APIs is used specifically for three-server redundancy.

Using Options File Keywords

None of the keywords in the options file affects three-server redundancy.

Troubleshooting Tips and Limitations

Three-server redundancy configurations require all three servers use the same platform type. You can use any of the following platform types in the configuration, but each server must use the same platform type: VM_UUID or PHY_*.

Using the BORROW keyword

The BORROW keyword capabilities are not compatible with three-server redundancy. This capability requires that license state data be maintained on the license server (this is called the borrow data cache). The information that identifies the state of a borrowed license can persist only on the license server that provided the borrowed licenses. This state information cannot be transferred to another license server when a failure occurs.

Using Trusted Storage–based Licensing

You cannot use three-server redundancy if you use trusted storage to store license rights.

Separating the Contents of a License File

Because the hostid values in the SERVER lines are computed into the signature of each FEATURE and INCREMENT line, make sure you keep SERVER lines together with any feature definition lines as they were generated. This means that if you move a feature definition line to another file, you must also move the respective SERVER lines and VENDOR line.

Putting the License File on a Network File Server

Do not put the license file on a network file server. If you do this, you lose the advantages of having failover protection because the file server becomes a single point of failure.

Using License Servers in Heavy Network Traffic

On a network with excessive traffic, the license servers may miss heartbeats which causes them to shut down the vendor daemon. The master may then stop serving licenses. If heavy network traffic causes this to occur, set the HEARTBEAT_INTERVAL to a larger value.

Enterprises can experience a performance issue when there is slow network communication or if FlexEnabled clients are using a dial-up link to connect to the network.

Using Multiple Vendor Daemons

The license server manager (`lmadmin` or `lmgrd`) cannot start vendor daemons from other software publishers when configured for three-server redundancy. The license server manager can manage only one vendor daemon. If one of the systems runs more that one vendor daemon, then the license administrator must run separate instances of the license server on that system to support the other vendor daemons. Make sure the port numbers do not clash.

Switching Between Three-Server Redundancy and Single-Server Configuration

While running, the license server manager (`lmadmin` or `lmgrd`) is not designed for switching from three-server redundancy to a single-server configuration (and vice-versa). To switch configurations, you need to do the following:

1. Shut down the license server manager in the single-server configuration or, in a three-server redundancy, all the license server managers currently running.
2. Do one of the following:
    - For `lmadmin`:

a. From the `lmadmin` installation directory on a given machine, import the required license files for the new configuration to which you are switching (either the three-server-redundant or single-server configuration):

```
lmadmin -import <new_license_file_list> -force
```

b. Restart `lmadmin`.

- For `lmgrd`:

  To restart `lmgrd` on a given machine in the configuration to which you are switching (either the three-server-redundant or single-server configuration), enter the following from the directory in which `lmgrd` is located:

```
lmgrd -c <new_license_file_list>
```

Comparing License Search Path Redundancy to Three-Server Redundancy

Are there any drawbacks to using the license search path for redundancy?

Yes. By default, once a *license job* has successfully checked out a license from a specific host, all subsequent checkouts must be satisfied from the same license server. If the application requires more than one license, this could result in a license denial when the license is available on another server. An application can bypass this restriction if it is coded with the use of multiple license jobs. Therefore, your customers may need to know whether your application is programmed in this manner.

If the application supports license queueing, all licenses are queued only from the first host in the license search path.

Finally, if one license server in the list becomes unavailable, some licenses are unavailable.

When is it recommended to use a license search path for redundancy rather than three-server redundancy?

- When there is less system administration available to monitor license servers.
- When load balancing is needed for FlexEnabled applications located far apart—for example, in London and Tokyo. You can make servers available locally with remote servers available as backup.
- License search path is more forgiving if one of the license servers goes down.
- This type of redundancy is not limited to three servers: It will work with any number of servers. For wide-area networks, local servers can be specified first, with remote servers available as backup.
- FlexEnabled applications do not require reliable networking to a single node with a license search path, so this is recommended where networking itself requires redundancy.

4

Understanding Heartbeats

The license server uses a TCP/IP port to communicate to FlexEnabled applications. The server can detect when a connection to the FlexEnabled application is gone and frees its licenses automatically. However, an application needs to communicate regularly with the license server to detect that the server is still running. This communication is effected using *heartbeats*.

A heartbeat is a message initiated by the application and acknowledged by the license server; this exchange assures each that the other is still up and running. By default, heartbeats are sent, and acknowledgments are received automatically by the FlexEnabled application at a predetermined interval.

If the license server is shut down and restarted, the heartbeat mechanism in the application automatically reconnects to the server and checks out the complement of licenses that is currently being used. If your application does not send heartbeats to the license server, the application cannot determine that the license server has been shut down and restarted. If the license server is restarted without the application's knowledge, current copies of the application continue running and a new, full complement of licenses becomes available for the license server, making license over-usage possible.

Heartbeat Process

Every time the license server receives a heartbeat, it immediately sends an acknowledgment message back to the application. The application does not pick up the message until it is time for the next heartbeat; this latency is, by default, 120 seconds. This means the FlexEnabled application does not detect that the license server is down until the time has passed for two heartbeats to be sent.

If the license server is shut down, the next heartbeat succeeds, because the acknowledgment processed is from the previous heartbeat. If a heartbeat is not acknowledged by the license server, the application can decide what action to take: continue, warn, or terminate.

Automatic vs. Manual Heartbeats

Deciding how the heartbeats occur and what action takes place when the license server is not running is an important part of incorporating FlexNet Publisher in an application. On both Windows and UNIX, automatic heartbeats are recommended and are automatically implemented as a separate thread in the FlexEnabled application.

Heartbeats can be automatic or manual. The distinction is how the timing of heartbeat messages is managed and how reconnection attempts are recognized and handled. Table 4-1 summarizes these distinctions.

Table 4-1 ▪ Heartbeat Summary

	Automatic Heartbeats	Manual Heartbeats
Heartbeat Timing	Control of heartbeat timing is transparent via a dedicated thread created on behalf of the application.	Control of heartbeat timing is regulated by the frequency with which the application invokes a particular function.
Reconnection Management	Reconnection to license server is handled automatically and transparently.	Application detects individual reconnection attempts to the license server and determines how to respond.
Considerations	• Requires the application be multithreaded. • A dedicated heartbeat thread is automatically interrupted rather than the application.	• No multithreaded requirement. • The application has total control of the timing of heartbeat messages.

Automatic Heartbeats

By default, heartbeats are automatically initiated when a license is checked out. Heartbeats are managed via a timer running as a separate thread that is created on behalf of the application. There are four aspects to automatic heartbeat behavior, all of which can be configured.

By default, the behavior of automatic heartbeats is implemented using:

- **Heartbeat thread management**—Heartbeats are managed by a dedicated thread in the application.
- **Heartbeat exchange interval**—Heartbeats are sent to and acknowledgements received from the license server at a default interval of 120 seconds.
- **License server reconnection strategy**—Reconnecting to a lost license server is attempted every 60 seconds for up to five attempts. There are three possible outcomes with automatic heartbeats when reconnection is attempted after the connection to the license server is lost:
  - Reconnection to the license server succeeded.
  - Reconnection to the license server was attempted and failed, but more attempts will be made.
  - Reconnection to the license server was unsuccessful after the retry limit was reached.

- **Recovery from a lost license server**—If connection is not re-established after five attempts, the application, by default, exits with the error message: `Lost License, cannot reconnect` (UNIX: stderr, Windows: dialog box).

Controlling Automatic Heartbeat Behavior

Various aspects of automatic heartbeat default behavior can be overridden by setting the appropriate Flexible API attributes using the function **lc_set_attr**. Table 4-2 lists the attributes and the effect they have on automatic heartbeat behavior.

Table 4-2 ▪ Automatic Heartbeat Control

Attribute	Description
Thread Management	
LM_A_MT_HEARTBEAT	Determines whether a dedicated thread in the application is used to manage heartbeats. Default is to use a dedicated thread. UNIX platforms only.
Exchange Management	
LM_A_CHECK_INTERVAL	Sets the interval at which heartbeats are exchanged. Default is 120 seconds.
Reconnection Strategy	
LM_A_RETRY_COUNT	Specifies the number of reconnect attempts. If –1, the application retries forever. Default is 5 attempts.
LM_A_RETRY_INTERVAL	Specifies the interval between retry attempts. Default interval is 60 seconds.
LM_A_USER_RECONNECT **LM_A_USER_RECONNECT_EX**	Sets a pre-reconnection callback to a vendor-defined routine. Default is no handler.
LM_A_USER_RECONNECT_DONE **LM_A_USER_RECONNECT_DONE_EX**	Sets a post-reconnection callback to a vendor-defined routine. Default is no handler.
Lost Server Recovery	
LM_A_USER_EXITCALL **LM_A_USER_EXITCALL_EX**	Sets a callback to a publisher-defined exit handler. Default is to call the `exit` system call on behalf of the application.

See Also
The following sections in the *C/C++ Function Reference*
LM_A_CHECK_INTERVAL
LM_A_MT_HEARTBEAT
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL
LM_A_USER_EXITCAL, LM_A_USER_EXITCALL_EX
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX

Manual Heartbeats

Manual heartbeats are used instead of automatic heartbeats if:

- The publisher wants the application to be in control of the precise timing of heartbeats.
- The application cannot tolerate the extra thread introduced by automatic heartbeats.
- The application cannot be multithreaded.
- The application can be interrupted only at specific times.

Some applications have critical sections of code that cannot be interrupted. Manual heartbeats provide a way for the application to control when the heartbeat thread executes, thereby not perturbing the critical code sections.

Task ***To set up manual heartbeats:***

1. Turn off the automatic heartbeat timer before a license is checked out. Set both the `LM_A_CHECK_INTERVAL` and `LM_A_RETRY_INTERVAL` attributes to `-1`, as in the following:

```
lc_set_attr(lm_job, LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE) -1);
lc_set_attr(lm_job, LM_A_RETRY_INTERVAL, (LM_A_VAL_TYPE) -1);
```

2. Invoke **lc_heartbeat** at whatever interval is deemed appropriate. You decide the timing mechanism used to control the interval. The following is an example code sequence using the Flexible API:

```
status = lc_heartbeat(lm_job, &num_reconnects, num_minutes);
if (status != 0)
{
    /* Monitor status and num_reconnects, take appropriate action */
}
```

   The heartbeat routine returns the number of reconnects over a given time period and the state of the connection to the license server. Monitor these return values and take appropriate action.

The heartbeat routine performs the following services on behalf of the application:

- It attempts to read the acknowledgement from the previous heartbeat.
- If there is no acknowledgement, it makes one attempt to reconnect to the license server.
- If there is acknowledgement, it sends the next heartbeat message.
- It returns with a status: 0 if there was an acknowledgment or the number of unacknowledged heartbeats. This number is incremented for each consecutive time a heartbeat was not acknowledged.

- After the fifth (default) consecutive unacknowledged heartbeat (five calls to the heartbeat routine), it causes the application to exit (via the standard **exit** system call) with the error message `Lost License, cannot reconnect` (UNIX: stderr, WINDOWS: dialog box).

See Also
The **lc_heartbeat** section in the *C/C++ Function Reference*

Controlling Manual Heartbeat Behavior

Various aspects of manual heartbeat default behavior can be overridden by setting the appropriate Flexible API attributes using **lc_set_attr**. Table 4-3 lists the attributes and the effect they have on manual heartbeat behavior.

Table 4-3 ▪ Manual Heartbeat Control

Attribute	Description
Reconnection Strategy	
LM_A_RETRY_COUNT	Used to adjust the number of consecutive reconnect attempts (each call to the heartbeat routine makes one attempt) to a lost license server before the heartbeat routine causes the application to exit. If –1, exit is never called. Default is five attempts.
LM_A_USER_RECONNECT **LM_A_USER_RECONNECT_EX**	Sets a pre-reconnection callback to a publisher-defined routine. Default is no handler.
LM_A_USER_RECONNECT_DONE **LM_A_USER_RECONNECT_DONE_EX**	Sets a post-reconnection callback to a publisher-defined routine. Default is no handler.
Lost Server Recovery	
LM_A_USER_EXITCALL **LM_A_USER_EXITCALL_EX**	Sets a callback to a publisher-defined exit handler. Default is for application exit.

See Also
The following sections in the *C/C++ Function Reference*:
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX

Manual Heartbeat Example

The following code sequence demonstrates one way to implement manual heartbeats using the Flexible API. This example incorporates the manual heartbeat timing loop within the main routine of the application, with the intention that application code be included within this timing loop.

```
#include "lmclient.h"
#include "lm_attr.h"
```

```
VENDORCODE code;
LM_HANDLE *lm_job;

int num_reconnects; /* parameter to lc_heartbeat() */
int num_minutes;    /* parameter to lc_heartbeat() */
int hbstat;         /* return status for lc_heartbeat() */
int exitcall(char *);     /* exit handler callback */

void
main(int argc, char * argv[])
{
    /* Initialize the job */
    if (lc_new_job(0, lc_new_job_arg2, &code, &lm_job))
    {
        lc_perror(lm_job, "lc_new_job failed");
        exit(lc_get_errno(lm_job));
    }

    /* Turn off automatic heartbeat timing. */
    lc_set_attr(lm_job, LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE) -1);
    lc_set_attr(lm_job, LM_A_RETRY_INTERVAL, (LM_A_VAL_TYPE) -1);

    /* Set retry count to 3 (default is 5); lc_heartbeat() makes LM_A_RETRY_COUNT number
        of reconnection attempts after license server connection is lost before calling the
        exit handler (defined below).*/
    lc_set_attr(lm_job, LM_A_RETRY_COUNT, (LM_A_VAL_TYPE) 3);

    /* Register the exit handler callback, "exitcall." The default is the system’s
        exit routine. */
    lc_set_attr(lm_job, LM_A_USER_EXITCALL, (LM_A_VAL_TYPE) exitcall);

    /* Check out the feature, "f1." */
    if (lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT,
            &code, LM_DUP_NONE))
    {
        lc_perror(lm_job, "checkout failed");
        exit (lc_get_errno(lm_job));
    }

    /* Start the manual heartbeat timing loop. This loop sends a heartbeat message with
    every iteration. */
    while (1)
    {
        /* When the following call to lc_heartbeat() returns, num_reconnects will contain the
            number of successful license server reconnects in the last num_minutes. */

        if (hbstat = lc_heartbeat(lm_job, &num_reconnects, num_minutes))
        {
        /* Connection to license server is lost and an attempt to reconnect unsuccessful.
            Monitor hbstat and respond as appropriate to your license policy. After
            LM_A_RETRY_COUNT number of reconnect attempts (3 in this example), control
            automatically transfers to the exit handler, exitcall */
        }
        /* Loop timing and application code for this license job occur here. */
    } /* end of heartbeat timing loop */
```

```
    lc_checkin(lm_job, feature ,0);
    lc_free_job(lm_job);
    exit(0);
}

/* Exit handler callback. This is invoked automatically after three consecutive reconnection
    attempts are made to a lost license server. */
int
exitcall(char * feature)
{
    printf("I am in the exit callback for feature %s\n", feature);
    exit(1);
}
```

License Server Considerations

By default, the license server waits two hours to hear from the FlexEnabled application before checking in its licenses.

- The `LM_A_TCP_TIMEOUT` attribute is used to adjust default timeout value for the TCP/IP port. Zero (0) means no TCP/IP port timeout. See the *C/C++ Function Reference* for more details.
- `ls_minimum_user_timeout` in `lsvendor.c` adjusts general timeout (not just TCP/IP related). The license administrator can adjust this timeout value further with the `TIMEOUT` and `TIMEOUTALL` options file keywords. See the *License Administration Guide* for more details.

5

Customizing the Vendor Daemon

When you build the production toolkit, the publisher-specific vendor daemon *(vendor* or `vendor.exe`) is built also. You can customize the publisher-specific vendor daemon using vendor variables in `machind/lsvendor.c`, but changes to this file are normally neither suggested nor required.

Task ***To customize the vendor daemon using vendor variables***

1. Configure the licensing toolkit as described in the *Development Environment Guide*.
2. Edit `lsvendor.c`: ensure that you are using the latest version from the current toolkit.
3. Build the production toolkit as described in the *Development Environment Guide*.

The build process generates the intermediate file, `lsrvend.c`, from `lsvendor.c`. Any errors in `lsvendor.c` show up as compile-time errors in `lsrvend.c`.

Basic Vendor Variables

If you need to customize your vendor daemon, you can edit the vendor variables in `lsvendor.c`. Usually, this file should be left as is. Most of the variables in this file appear for historic and compatibility reasons and should not be used except where required for compatibility. This chapter provides information for the most popular vendor variables.

Table 5-1 ▪ Basic Vendor Variables

Variable	Description
ls_allow_physical_ethernetid_only	Controls whether only hostids from physical ethernet adapters are authenticated.
ls_allow_tz_override	Enables overriding of the license server time zone and time settings.

Table 5-1 ▪ Basic Vendor Variables

Variable	Description
ls_allow_vm	Specifies whether the vendor daemon is restricted to run only in a virtual environment or is restricted to run only on a physical machine.
ls_a_check_baddate	Checks whether the system date has been set back.
ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade	Controls whether an `INCREMENT` line will require the vendor string to match in order to pool its licenses. Controls whether an UPGRADE line will require the vendor string to match in order to upgrade another license.
ls_daemon_periodic	Ensures that a specific function will be called periodically in the vendor daemon's main processing loop.
ls_enable_verbose_dlog	Controls whether the vendor daemon generates the debug log messages in a new format.
ls_flexid9_hasp4_support	Enables the borrow functionality of licenses for the end user machine.
ls_flexid9_hasp4_support	Turns the server-side HASP4 detection on or off.
ls_incallback	Installs a vendor-defined checkin callback routine.
ls_infilter	Installs a vendor-defined checkin filtering routine.
ls_min_lmremove	Makes the `lmremove` utility ineffective for a specified period of time.
ls_minimum_user_timeout	Specifies the minimum value allowed for a feature's TIMEOUT value.
ls_outfilter	Installs a vendor-defined checkout filtering routine.
ls_secure_comms	Enables secure communication between client and vendor daemon in addition to the existing plain communication.
ls_server_borrow_reread_interval	Enables the remote license server to reread the changes in transfer cache on the remote machine at regular specified interval.
ls_show_vendor_def	Enables display of vendor-defined checkout string using `lmstat`.
ls_support_custom_daemon_select_timeout	Controls the use of the `DAEMON_SELECT_TIMEOUT` keyword in the vendor-daemon options file.
ls_user_init1	Installs an initialization routine that runs before normal vendor daemon initialization.
ls_user_init2	Installs an initialization routine that runs after normal vendor daemon initialization.

Table 5-1 ▪ Basic Vendor Variables

Variable	Description
ls_user_init3	Installs an initialization routine that runs after the license file is read and after each `lmreread`.
ls_user_init_attributes	Installs an initialization callback routine that can be used to set attributes in your vendor daemon.
ls_vd_shutdown	Enables you to define a callback routine that is invoked when the license server is shut down via the `lmdown` utility.
ls_vendor_msg	Enables you to add support for sending messages from your FlexEnabled application to the daemon.
ls_vendor_msg_async	Enables asynchronous processing of **lc_vsend** messages.
ls_vendor_msg_list_size	Controls the maximum number of outstanding **lc_vsend** messages allowed in the Vendor Daemon.

See Also
Obsolete Vendor Daemon Variables

ls_allow_physical_ethernetid_only

```
int ls_allow_physical_ethernetid_only = 0;
```

This variable operates on Windows platforms only. If set to 1, then the vendor daemon authenticates only licenses that use physical ethernet IDs: hostids generated from virtual adapters cannot be used in feature definition lines as they will fail the hostid checks. The default operation of the vendor daemon when this variable is not set or it is set to 0 (zero) is that the hostids from both physical and virtual ethernet adapters will be authenticated. This behavior matches the behavior of vendor daemons created with toolkits prior to version 11.6.1. This vendor variable was introduced in version 11.6.1.

Caution ▪ *Setting this variable to 1 will cause previously issued licenses that use hostids from virtual ethernet adapters to fail.*

ls_allow_tz_override

```
unsigned int ls_allow_tz_override = 0;
```

To enable override of the license server time zone and time settings, set `ls_allow_tz_override` to 1. When the variable is set to 1, the vendor daemon reads the values of the environment variables `LM_SET_SERVER_TIMEZONE` and `LM_SET_SERVER_TIME`. Use these environment variables to test scenarios that simulate vendor daemons running in different time zones.

Note ▪ *It is recommended that you set the `ls_allow_tz_override` variable to 0 and rebuild your toolkit before distributing to your customers, if you have set the variable to 1 for testing.*

See Also
Testing Time Zone Licensing

ls_allow_vm

```
FLEX_VM_TYPE ls_allow_vm = PHYSICAL | VM_ONLY | VM_ALL
```

Specifies whether the vendor daemon is restricted to run only in a virtual environment or is restricted to run only on a physical machine. By default, the vendor daemon can operate in a virtual environment or on a physical machine.

To use the non-default values for the `ls_allow_vm` variable, you must have a set of vendor keys that provides for virtualization support.

The following values are supported. Specify one and only one value.

Table 5-2 ▪ ls_allow_vm Values

Value	Description
PHYSICAL	Restricts the license server to run only on a physical machine; the license server cannot be run in a virtual environment.
VM_ONLY	Restricts the license server to run only in a supported virtual environment; the license server cannot be run on a physical machine. See the current FlexNet Publisher *Release Notes* for information about supported VM platforms.
VM_ALL	Allows the license server to run on both physical and virtual environments.

See Also
Virtualization

ls_a_check_baddate

```
int ls_a_check_baddate = 0;
```

If set to 1, and the license that would authorize a checkout is expiring, a check is made to see if the system date has been set back. If the failure is due to detection of system date tampering, the checkout error will be `LM_BADSYSDATE`.

See Also
LM_A_CHECK_BADDATE in the *C/C++ Function Reference*
Limited Functionality Demos

ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade

```
int ls_compare_vendor_on_increment = 0;
int ls_compare_vendor_on_upgrade = 0;
```

If VENDOR_STRING is used in your license files, then the value of these two variables may need to be modified. You cannot use only one variable: If you set one, you must set both.

ls_compare_vendor_on_increment gives you control over whether an INCREMENT line will require the vendor string to match in order to pool its licenses. If set to a non-zero value, then the string is included in the pooling decision; if set to 0, then the string is not involved in pooling.

ls_compare_vendor_on_upgrade gives you control over whether an UPGRADE line will require the vendor string to match in order to upgrade another license. If set to a non-zero value, then the string is included in UPGRADE line matching decision; if set to 0, then it is not included.

INCREMENT lines add licenses to a prior FEATURE or INCREMENT line and are combined into one pool if all of the following are true:

- Feature names match
- Feature versions match
- Any node-locked hostid, if present, matches
- USER_BASED and HOST_BASED fields match
- PLATFORM fields match
- DUP_GROUP fields match
- FLOAT_OK fields match
- VENDOR_STRING fields match

See Also
VENDOR_STRING
UPGRADE Lines

ls_daemon_periodic

```
void (*ls_daemon_periodic)() = 0;
```

If you set the function pointer ls_daemon_periodic in lsvendor.c to one of your functions, this function will be called approximately once per minute in the vendor daemon's main processing loop. You must ensure that the object file (.o/.obj) for this routine is linked into your vendor daemon.

ls_get_status_is_master

```
int stat = 0;
stat = ls_get_status_is_master();
```

This function, declared in `ls_attr.h`, can be used from the vendor daemon's `ls_daemon_periodic` callback (and only from that callback) within `lsvendor.c` to determine if that vendor daemon is master. `ls_get_status_is_master` returns a non-zero integer if the vendor daemon is master. This function cannot be called from `ls_user_initx callbacks` because this state is unknown at that point.

A limitation is that `ls_get_status_is_master` works only from vendor daemons started with `lmgrd`. All of a triad's vendor daemons started with `lmadmin` will report a non-zero value from `ls_get_status_is_master`.

ls_enable_verbose_dlog

```
int ls_enable_verbose_dlog = 0;
```

This vendor variable controls whether the vendor daemon generates the debug log messages in a new format. This vendor variable affects the following debug log message types:

- OUT
- IN
- QUEUED
- DEQUEUED
- DENIED

Table 5-3 ▪ ls_enable_verbose_dlog Values

Variable Setting	Desciption
0	Allows the vendor varaible to generates the debug log messages in an earlier format (default format). It is the default value.
1	Allows the vendor variable to generates the debug log messages in a new format.

Note ▪ *Note the following difference in the debug log messages after setting this variable:*

- *Setting the variable value to 0 generates the `8:41:54 (demo) OUT: "f1" ADas@BLR-LT-104276` debug log message.*
- *Setting the variable value to 1 generates the `8:41:54 (demo) OUT: "f1" ADas@BLR-LT-104276 ^^^ 1.0 ADas 127.0.0.1 29865 0x12CE 1 3 N/A 11 19 8-8-2023 8:41:56 YES ^^^` debug log message.*

ls_flexid9_hasp4_support

```
unsigned int ls_flexid9_hasp4_support = 0;
```

Use this variable to turn the server-side HASP4 detection on or off.

When set to `1`, `ls_flexid9_hasp4_support` executes all HASP4 Safenet API calls. When set to `0`, no HASP4 APIs are executed.

Note ▪ *The HASP4 keys are not supported from FlexNet Publisher 11.18.2.*

See Also

`LM_A_FLEXID9_HASP4_SUPPORT` in the *C/C++ Function Reference*

ls_incallback

```
int (*ls_incallback)() = 0;
```

To install a vendor-defined checkin callback routine, initialize `ls_incallback` with a pointer to your routine. The checkin callback is called with no parameters, and the return value is unused. The checkin callback routine is called after the checkin is performed.

To obtain the parameters of the current checkin call, use the **ls_get_attr** call described in ls_outfilter.

ls_infilter

```
int (*ls_infilter)() = 0;
```

To install a vendor-defined checkin filtering routine, initialize `ls_infilter` with a pointer to your routine. The checkin filter is called with no parameters. If it returns 0, the current checkin is aborted; a return of 1 allows the current checkin to continue. If the filter aborts the operation (returns 0), then it should set the error code, using **lc_perror**, appropriately.

To obtain the parameters of the current checkin call, use the **ls_get_attr** call described in ls_outfilter.

ls_min_lmremove

```
int ls_min_lmremove = 120;
```

The `lmremove` utility could be used to bypass the license count for a feature if a license administrator were to run `lmremove` on each user as soon as they had checked out a license. `ls_min_lmremove` makes the `lmremove` utility ineffective for a specified period of time after a user connects to the daemon (120 seconds by default).

ls_minimum_user_timeout

```
int ls_minimum_user_timeout = 900;
```

This is the minimum value (in seconds) allowed for a feature's `TIMEOUT` value. An attempt to set a timeout value less than `ls_minimum_user_timeout` will result in the maximum value being set. If `ls_minimum_user_timeout` is set to <=0, then the user `TIMEOUT` option is disabled.

ls_outfilter

```
int (*ls_outfilter)() = 0;
```

Note ▪ *Contact Revenera Technical Support for assistance before using `ls_outfilter`. Callbacks in this area are rarely needed.*

To install a vendor-defined checkout filtering routine, initialize `ls_outfilter` with a pointer to your routine. The checkout filter is called with no parameters. If it returns 0, your routine has either checked out the feature, or rejected the checkout request. If it returns 1, then the normal server checkout occurs.

If 0 is returned and the checkout fails, set the error appropriately using **lc_perror**.

ls_get_attr

To obtain the parameters of the current checkout call, use the **ls_get_attr** call. This is for use only in the `ls_outfilter` callback.

`ls_get_attr(attr, &value)`

where:

Table 5-4 ▪ ls_get_attr Parameters

Parameter	Description
attr	An attribute specified in `ls_attr.h`. See *LS_ATTR_LAST_REPORT_LOG_ROTATION* and *LS_ATTR_REPORT_LOG_FILENAME* for a description of attributes only used with ls_outfilter.
value	Value of the attribute.

ls_get_attr operates in the same manner as **lc_get_attr**. **ls_get_attr** allows you to retrieve the values of the feature name, user, host, display, and so on for use in your filtering function.

LS_ATTR_LAST_REPORT_LOG_ROTATION

Type

`time_t`

Default

`NULL`

This attribute is used with **ls_get_attr** to get the last date and time that the report log was rotated. **ls_get_attr** is used with a server-side checkout filter to add custom business rules that determine whether a license checkout request is granted.

See Also

LS_ATTR_REPORT_LOG_FILENAME

LS_ATTR_REPORT_LOG_FILENAME

Type

`const char *`

Default

`NULL`

LS_ATTR_REPORT_LOG_FILENAME is used to get the report log file name and to checkout filter to add custom business rules that determine whether a license checkout request is granted.

ls_set_attr

To set the parameters of the current checkout call, use the `ls_set_attr` call. This is for use only in the `ls_outfilter` callback.

`ls_set_attr(`*`attr`*`, &`*`value`*`)`

where:

Table 5-5 ▪ ls_set_attr Parameters

Parameter	Description
attr	An attribute specified in `ls_attr.h`. See *LS_ATTR_CHECKOUT_DATA* for more information.
value	Value of the attribute.

ls_set_attr operates in the same manner as **ls_get_attr**. **ls_get_attr** allows you to retrieve the values of the feature name, user, host, display, and so on for use in your filtering function.

LS_ATTR_CHECKOUT_DATA

Type

`char *`

Default

`NULL`

LS_ATTR_CHECKOUT_DATA is used to set and get the checkout data.

ls_checkout

The **ls_checkout** vendor daemon routine is only for use in `ls_outfilter` callbacks:

```
ls_checkout(feature, num_lic, wait, who, version, dup_sel, linger, sign, 0,1);
```

Parameters

The **ls_checkout** function uses the following parameters:

Table 5-6 ▪ ls_checkout Parameters

Parameter	Description
feature	Feature desired.
num_lic	Number of licenses.
wait	“Wait until available” flag. If set to 1, the request is queued if a license is not available.
who	The user.
version	Version number of feature.
dup_sel	Duplicate license selection criteria.
linger	How long the license is to linger.
sign	Signature in the feature definition line.

Note **▪ ls_get_attr** *can be used to retrieve all the parameters that* **ls_checkout** *requires.*

Return

The **ls_checkout** function returns the following:

Table 5-7 ▪ ls_checkout Return Values

Return Value	Description
0	Success, license checked out
<> 0	Error.

ls_secure_comms

```
int ls_secure_comms = 0;
```

When set to 1, the vendor daemon can support secure communication with clients, provided the dynamic library `libfnpssl` is present in the current working directory.

By default, this variable is set to 0, which means secure communication with clients will not be enabled.

ls_server_borrow_reread_interval

```
int ls_server_borrow_reread_interval = 180;
```

This vendor variable enables the remote license server to reread the changes in transfer cache on the remote machine at regular specified interval.

Table 5-8 ▪ Variable Settings

Variable Setting	Description
180	It is the default value, the remote license server is enabled to reread the changes in transfer cache at every 180 seconds.
0	The remote license server is disabled to reread the changes in transfer cache.

ls_show_vendor_def

```
int ls_show_vendor_def = 0;
```

Your FlexEnabled application can send a vendor-defined checkout string to the daemon on each checkout request. If `ls_show_vendor_def` is non-zero, this data will appear in **lc_userlist** calls, and hence, in `lmstat` output. If you use this vendor-defined checkout data and want your users to be able to view it with `lmstat`, then set `ls_show_vendor_def` to 1.

ls_support_custom_daemon_select_timeout

```
unsigned int ls_support_custom_daemon_select_timeout = 0;
```

This variable controls the use of the `DAEMON_SELECT_TIMEOUT` keyword in the vendor-daemon options file. `DAEMON_SELECT_TIMEOUT` allows a license administrator to define a threshold for the amount of time that can elapse before a vendor-daemon connection experiences a timeout.

- If this variable is set to **0** (the default), the `DAEMON_SELECT_TIMEOUT` keyword is disabled, prohibiting customization of the timeout threshold.
- If this variable is set to **1**, the license administrator can use `DAEMON_SELECT_TIMEOUT` to customize the timeout threshold.

When `DAEMON_SELECT_TIMEOUT` is disabled (or not defined) in the options file, a vendor-daemon connection will time out after 1 second, by default.

Important ▪ *Best practice is to set this variable to* **1** *only for environments known to have limited bandwidth.*

ls_user_init1

```
void (*ls_user_init1)() = 0;
```

To install an initialization routine that runs before normal vendor daemon initialization, initialize `ls_user_init1` with a pointer to your routine and make sure an object file with this function is linked with your vendor daemon.

ls_user_init2

```
void (*ls_user_init2)() = 0;
```

To install an initialization routine that runs after normal vendor daemon initialization, initialize `ls_user_init2` with a pointer to your routine and make sure an object file with this function is linked with your vendor daemon.

ls_user_init3

```
void (*ls_user_init3)() = 0;
```

To install an initialization routine that runs after the license file is read and after each `lmreread`, initialize `ls_user_init3` with a pointer to your routine and make sure an object file with this function is linked with your vendor daemon.

ls_user_init_attributes

```
void (*ls_user_init_attributes)() = 0;
```

To install an initialization callback routine that can be used to set attributes in your vendor daemon, initialize `ls_user_init_attributes` with a pointer to your routine. An object file that includes this initialization function must be linked to your vendor daemon. The callback routine executes after the vendor daemon is initialized but before any license files are read.

The following functions can be used within your intialization function to customize your vendor daemon:

- **lc_set_attr**—Sets attributes in a single vendor daemon or for all vendors defined in a Common Vendor Daemon.
- **lc_set_attr_vendor**—Sets attributes for a single vendor in a Common Vendor Daemon.
- **ls_set_log_vendor_enable**—Defines which debug log messages are output by the vendor daemon.

See the *C/C++ Function Reference* for a description of **lc_set_attr** and **lc_set_attr_vendor,** and the attributes that can be set using them.

ls_set_log_vendor_enable

The **ls_set_log_vendor_enable** function is for use only in `ls_user_init_attributes` callbacks.

```
int ls_set_log_vendor_enable(
    char * string,
    int enable);
```

Parameters

The **ls_set_log_vendor_enable** function uses the following parameters.

Table 5-9 ▪ ls_set_log_vendor_enable Parameters

Parameter	Description
string	Specifies the type of debug log message whose output is to be controlled. The following are valid values: ● "IN" = details of checked-in licenses ● "OUT" = details of checked-out licenses ● "QUEUED" = details of license requests that have been queued ● "DENIED" = details of license requests that were denied ● "UNSUPPORTED" = details of license requests that were for unsupported features
enable	Specifies whether debug log messages of the specified type are output. The following are valid values: ● 1 = enable logging (the default) ● 0 = disable logging

Return

This function returns the following values.

Table 5-10 ▪ ls_set_log_vendor_enable Return Values

Return Value	Description
0	Success.
1	Error, an invalid parameter has been input.

Discussion

The default behavior of the vendor daemon is to output all debug log messages. This function disables the output of the specified type of debug log message.

The output of debug log messages is also controlled by the setting of the `NOLOG` option in the options file. This setting is overriden by the `NOLOG` parameter in the options file. See the *License Administration Guide* for details about the options file.

ls_vd_shutdown

```
void (*ls_vd_shutdown)() = 0;
```

You can define a callback routine that is invoked when the license server is shut down via the `lmdown` utility. (For example, you might need a routine that provides customize cleanup operations.) To implement this callback, initialize `ls_vd_shutdown` with a pointer to the routine.

Note ▪ *The additional time needed for callback processing might cause the vendor daemon to time out and terminate before the callback can complete.*

ls_vendor_msg

```
char * (*ls_vendor_msg)() = 0;
```

To add support for sending messages from your FlexEnabled application to the daemon (with **lc_vsend**), initialize `ls_vendor_msg` with a pointer to your routine which processes the message and creates the reply for the FlexEnabled application. `ls_vendor_msg` is called with a single parameter, the character string sent by the FlexEnabled application. It should create a reply message and return a pointer to it. The message string will be unused the next time that `ls_vendor_msg` is called, so the use of a single static char array in `ls_vendor_msg` is appropriate. Make sure an object file with this routine is linked with your vendor daemon.

When ls_vendor_msg_async is set to 1, the ls_vendor_msg callback is processed asynchronously. In most cases, this will be all that needs to be done. However, when setting this vendor attribute producers should also review their ls_vendor_msg implementation for the following two scenario:

1. If ls_vendor_msg uses the internal lm_job handle and makes lc_* calls to the same VD, then this should be updated to use a separate job handle in order to prevent race conditions on the lm_job handle. The vf_nextconf.c example shows how to safely create a new job handle in an asynchronous callback in a VD. You have to note that:
   - this method of creating a new job should not be used in synchronous callbacks in the VD when subsequently calling lc_* APIs to the same VD since this will result in a deadlock in the message processing queue.
   - this step is not required (i.e. is optional) when lc_* calls are made to a different VD.
2. A number of undocumented ls_* APIs have historically been used in VD callbacks. When processing asynchronously, any calls to these APIs should be protected by calls to the new functions ls_api_mutex_lock(<timeout in ms>) and ls_api_mutex_unlock(). Example usage can be found in vf_prime.c.

See Also
`lc_vsend` in the *C/C++ Function Reference*

ls_vendor_msg_async

```
ls_vendor_msg_async = 0;
```

This variable enables the asynchronous processing of lc_vsend messages when it is set to 1. That is, the ls_vendor_msg callback is processed asynchronously.

ls_vendor_msg_list_size

```
ls_vendor_msg_list_size = 64;
```

The variable controls the maximum number of outstanding lc_vsend messages allowed in the Vendor Daemon. It has no effect unless ls_vendor_msg_async is set to 1.

Advanced Vendor Variables

The vendor variables in this section provide advanced capabilities beyond what is provided by the basic variables in Customizing the Vendor Daemon.

Table 5-11 ▪ Advanced Vendor Variables

Variable	Description
ls_a_license_case_sensitive	Controls license file case sensitivity in the vendor daemon.
ls_allow_updated_feature_borrow	Activates the validation for the server borrow cache entries.
ls_borrow_in	Callback that removes the given borrow-data record from a borrow-data cache.
ls_borrow_init	Callback that returns all borrow-data records from a borrow-data cache.
ls_borrow_out	Callback that adds the given borrow-data record to a borrow-data cache.
ls_borrow_return_early	Controls ability for borrowed licenses to be returned early.
ls_checkout_filter_dup_enable	Enables the DUP_GROUP functionality while using checkout filter.
ls_client_hostid_callback	Enables the extraction of the client hostid in server callbacks.
ls_client_removal_on_reread	Controls client re-authentication during reread operation.
ls_do_checkroot (UNIX Only)	Controls requirement for the vendor daemon's residence on a real root file system.
ls_dump_send_data	Controls the output of transmitted daemon data.
ls_hud_hostid_case_sensitive	Controls case sensitivity for hostid types `HOSTNAME`, `DISPLAY`, and `USER`.
ls_ignore_msg_encryption	Disables message obfuscation when `ls_message_security` is used.
ls_message_security	Sets the level for the message obfuscation.
ls_no_ipaddress_in_server_cache	Returns borrowed licenses if there is a mismatch between IP address while borrowing and returning the license.

Table 5-11 ▪ Advanced Vendor Variables

Variable	Description
ls_server_override_client_tcp_timeout	Provides an option for the vendor daemon to specify how long(in seconds) the license server waits.
ls_server_retry_cnt_after_tcp_timeout	Defines the number of tcp keepalive probes.
ls_server_retry_intvl_after_tcp_timeout	Defines the time interval between the successive tcp keepalive probes.
ls_prefer_feature_timeout_over_tcp	Enables the feature timeout preference over tcp timeout.
ls_single_xver_signature	customize X-version usage for legacy clients.
ls_ts_recovery	Installs vendor-defined trusted storage automatic recovery callback routine.
ls_use_all_feature_lines	Controls feature definition line succession.
ls_use_exclusive_commRev4	Authenticates and communicates CommRev4 messages.
ls_user_lockfile	Renames the vendor daemon locking mechanism.
ls_user_lock (Windows only)	Renames the vendor daemon locking mechanism.

ls_a_license_case_sensitive

```
int ls_a_license_case_sensitive = 0;
```

This variable controls case sensitivity for feature definition lines in the license file in the context of the vendor daemon. If set to `1`, licenses are case sensitive. Default is `0`, not case sensitive.

If this variable is set to `1`, then the corresponding case-sensitivity attribute, `LM_A_LICENSE_CASE_SENSITIVE`, must be set in the FlexEnabled application and license generator. `ls_a_license_case_sensitive` has no impact while checking out a borrowed license from the system registry as this flag is applicable only during check-out from license server. Check-out from borrow registry is always case-sensitive.

See Also

`LM_A_LICENSE_CASE_SENSITIVE` in the *C/C++ Function Reference*
ls_hud_hostid_case_sensitive

ls_allow_updated_feature_borrow

```
int ls_allow_updated_feature_borrow = 0;
```

This variable enables or disables the validation for the server borrow cache entries.

The default value is 0 and the behavior is that the borrow cache is created when licenses are borrowed from the server. In the event of server restart, if newer lines in certificate are updated or sign value has changed, borrow cache will be deleted.

When this variable is set to 1, the server borrow cache will be retained (not deleted) based on the pooling keywords, exception to this is time-zone of the updated feature lines.

ls_borrow_in

```
void (*ls_borrow_in)(FNP_BORROW_CACHE_ENTRY *borrow) = sBorrowIn;
```

This callback function removes the record that matches `borrowdata` from the borrow-data cache. The vendor daemon calls this function, if set, every time a borrow/lingered license expires or a borrowed license is returned.

By default, this callback is defined with an internal implementation that uses a predefined borrow-data cache. You can override the internal implementation with your own function that accesses your vendor-defined cache. Your implementation needs to do the following:

- Open your vendor-defined cache.
- Iterate through each record in the cache to find the one match that matches `borrow`.

  The initial field of each record (`sizeof(int)`) contains the size, in bytes, of the record. The size of this initial field is included in the count.
- Remove the matching record from the cache.
- Cleanup the cache, as appropriate.

Note ▪ *Do not attempt to free `borrow`. It is allocated on the stack and will get popped on return from the callback.*

This function is one of the borrow-data cache management routines, the others being **ls_borrow_init** and **ls_borrow_out**.

See Also
ls_borrow_init
ls_borrow_out

ls_borrow_init

```
void (*ls_borrow_init)(char ** borrowbuf, int * bufsize) = default_BorrowInit;
```

This callback function

- Allocates a buffer of `bufsize` bytes with **malloc**. The vendor daemon subsequently uses **free** to free the buffer.
- Initializes `borrowbuf` with a pointer to a byte array containing the contents of the borrow-data cache.
- Initializes `bufsize` with the size, in bytes, of the borrow-data cache.

The borrow-data cache contains a record with borrow data for each borrowed license. The initial field of each record (`sizeof(int)`) contains the number of bytes that compose that record. The first byte of the subsequent record immediately follows the last byte of the previous record.

The vendor daemon, at startup time, calls this function if it is set. By default, this callback is defined with an internal implementation that uses an internally defined borrow-data cache. You can override the internal implementation with your own function that accesses your vendor-defined borrow-data cache. The following code is a sample implementation of this function using the file, `myBorrowFile`, as the vendor-defined borrow-data cache:

```
void
myBorrowInit(char **borrowbuf, int *bufsize)
{
    int filesize = 0;
    char filename[MAX_PATH] = "myBorrowFile";
    FILE * fp = NULL;
    *bufsize = 0;
    .
    .
    .
    /* open borrow-data file, "myBorrowFile" */
    .
    .
    .
    (void)fseek(fp, 0, SEEK_END);
    filesize = ftell(fp); /* get size of "myBorrowFile" */

    if(filesize) /* there is borrow data to return*/
    {
        /* alloc buffer, borrowbuf, with filesize number of bytes */
        /* assumes that record are stored such that the first byte of a subsequent record
           immediately follows the last byte of the previous record. */
        *borrowbuf =
            (char *)malloc(sizeof(char) *filesize);
        if(*borrowbuf)
        {
            /*Copy all data into buffer */
            (void)fseek(fp, 0, SEEK_SET);
            fread(*borrowbuf, filesize, 1, fp);
            *bufsize = filesize;
        }
    }
    else /* there is no borrow data to return */
    {
        *borrowbuf = NULL;
        *bufsize = 0;
    }

    /*close the borrow-data file*/
    fclose(fp);
}
```

This function is one of the borrow-data cache management routines; the others being **ls_borrow_in** and **ls_borrow_out**.

See Also

ls_borrow_in
ls_borrow_out

ls_borrow_out

```
void (*ls_borrow_out) (char ** borrowdata, int datasize) = default_BorrowOut;
```

This callback function appends the record, `borrowdata`, to a borrow-data cache. The vendor daemon calls this function, if set, every time a license is borrowed or lingered.

By default, this callback is defined with an internal implementation that uses an internally defined borrow-data cache. You can override the internal implementation with your own function that accesses your vendor-defined cache. Your implementation needs to:

- Open the vendor-defined cache.
- Append the contents of `borrowdata` of size `datasize`.
- Close the vendor-defined cache.

Note ▪ *Do not attempt to free the `borrowdata` buffer itself. It is allocated on the stack and will get popped on return from the callback.*

This function is one of the borrow-data cache management routines, the others being **ls_borrow_in** and **ls_borrow_init**.

See Also
ls_borrow_in
ls_borrow_init

ls_borrow_return_early

```
int ls_borrow_return_early = 0;
```

This variable controls whether the vendor daemon allows a borrowed license to be returned before the borrow period is over.

Table 5-12 ▪ Variable Settings

Variable Setting	Description
`0`	Do not allow borrowed licenses to be returned early. This is the default setting.
`1`	Allow licenses to be returned early.

See Also
ls_borrow_in
ls_borrow_init
ls_borrow_out

ls_checkout_filter_dup_enable

```
int ls_checkout_filter_dup_enable = 0;
```

This variable enables the DUP_GROUP functionality with checkout filter enabled on client side. The filter function on the client side will receive FEATURE/INCREMENT line, which fulfills the DUP_GROUP criteria as first priority.

Table 5-13 ▪ Variable Settings

Variable Setting	Description
0	Do not enable the DUP_GROUP functionality. This is the default setting.
1	Enable the DUP_GROUP functionality.

ls_client_hostid_callback

```
void (*ls_client_hostid_callback) () = 0;
```

The vendor-defined hostid (VDH) and client MAC address (aka ETHER) hostid can be extracted from the vendor daemon's checkout filter (`ls_outfilter`) and other callbacks (`ls_outod_callback`, `ls_inod_callback`, `ls_infilter`, `ls_incallback`, `ls_vd-shutdown`), by means of the server-side attributes: `LS_ATTR_CLIENT_HOSTID_VENDOR` and `LS_ATTR_CLIENT_HOSTID_ETHER` (defined in file `machind\ls_attr.h`).

The `ls_co_ex.c` advanced toolkit example demonstrates the extraction of the MAC address(es) and the vendor-defined hostid of the client requesting the feature checkout, via the **s_get_client_hostids** sample function.

This feature is designed to work with old FlexNet Publisher clients (the oldest client tested was 11.12.1). A consequence of this design decision is that the vendor daemon callback, `ls_client_hostid_callback`, must be implemented in a vendor daemon where client hostids are extracted in other callbacks such as `ls_outfilter`. `ls_client_hostid_callback` specifies which client hostids are required by calling the new server API `ls_attr_init_client_hostids` (see the example implementation `ls_client_hostid_callback_example` in `ls_co_ex.c`). This ensures that additional message exchanges that are required to retrieve hostids from legacy clients occur before `ls_outfilter` is executed.

Producers should be aware that using `ls_client_hostid_callback` necessarily increases the number of messages between client and vendor daemon during checkout. Further, since a client may have multiple MAC address hostids, the server will retrieve up to five hostids from each client. Therefore, if both hostids are specified, a maximum of six additional round-trips occur between the client and the vendor daemon on a client's first checkout. This negatively affects both vendor daemon scalability and client first-checkout performance. Producers are therefore advised to perform their own performance and scalability testing when using this feature.

Clients performing an early return from a borrowed license will not have their hostids available in checkin callbacks `ls_infilter` and `ls_incallback`.

ls_client_removal_on_reread

```
int ls_client_removal_on_reread = 1;
```

Significant changes are introduced in the license server reread functionality from FNP version 11.14.1.0 onwards. The reread functionality has been enhanced to resolve the license leakage on the license server. The changes were introduced to disconnect the clients whose feature license are no longer valid after license server has performed reread operation. Previously, clients always stay in connected state and keep using feature license, which is no longer valid on license server after reread has been performed.

The changes were introduced in the following phases:

- First phase, in FNP version 11.14.1.0, has introduced the change, which disallowed clients to further use the feature license if newly deployed license file no longer supports it. It can be because of any reason mentioned below occurred on the license pool used for feature checkout:

  1. The license pool no longer exists.

     For example:

     - CLIENT1 and CLIENT2 have checked out licenses from INCREMENT line having `VENDOR_STRING as "abc".`

       ```
       INCREMENT f1 demo 1.0 1-jan-2020 8 VENDOR_STRING="abc" SIGN=9BFAC03164ED
       ```

     - License file is edited with VENDOR_STRING changed from "abc" to "def". CLIENT1 and CLIENT2 are no longer allowed to further use the same license.

       ```
       INCREMENT f1 demo 1.0 1-jan-2020 8 VENDOR_STRING="def" SIGN=afb303162056
       ```

  2. The number of licenses are decreased from same license pool.

     For example:

     - Each license of the below INCREMENT line is checked out by CLIENT1 and CLIENT2 sequentially.

       ```
       INCREMENT f1 demo 1.0 1-jan-2020 2 SIGN=eda701122006
       ```

     - In edited license file the license count is decreased by 1. As a reason, CLIENT2 is no longer allowed to use the feature license.

       ```
       INCREMENT f1 demo 1.0 1-jan-2020 1 SIGN= bfc32010132
       ```

- Second phase, in FNP version 11.16.2.0, the scope further extended to options file exclusion lists. If new edited option file disallows the feature license for the client, then after reread server will disallow to use the feature licenses.

  For example:

  - Following was the INCREMENT line and option file used by license server. John has checked out both the license one after another from separate client applications.

    ```
    INCREMENT f1 demo 1.0 1-jan-2020 2 SIGN=eda701122006
    ```

    ```
    Option File: RESERVE 1 f1 USER john
    ```

  - Option File is edited and now reservation is moved to Daniel. Client application, which has checked out the license from reserve pool, will lose the license as reservation is now moved to Daniel. The license for other client application will stay intact.

```
Option File: RESERVE 1 f1 USER Daniel
```

Server reread functionality validates again every client and its related checked out features. If any of the feature license fails authentication then license server drops client. The license server re-authenticates the checkout request for the features, if sent again by client. This complete process may impact server performance if higher number of clients gone through it.

Vendor daemon can be customized to move back pre FNP 11.14.1 reread behavior with vendor variable `ls_client_removal_on_reread`.

The vendor variable can be used to switch server behavior between previous and latest reread behavior.

The following values are supported. Specify any one of the values:

Table 5-14 ▪ Value Setting

Value	Description
1	It is the default value. The license server is enabled with latest reread functionality and clients feature license will be re-authenticated during server reread.
0	License server will use archive reread functionality and clients keep using feature licenses even if no longer valid.

ls_do_checkroot (UNIX Only)

```
int ls_do_checkroot = 0;
```

To require that your vendor daemon be running on a file system that has its root directory as the “real” root directory of the disk, set this option. This prevents a user from cloning part of the UNIX file hierarchy and executing the daemon with a `chroot` command. If this were done, the vendor daemon locking would be bypassed and the user could run as many copies of your vendor daemon as desired.

Note ▪ *Theft by using `chroot` is an obscure, difficult kind of theft. This is because the user must have root permission, and setting up a phony root directory is a non-trivial task. It requires that the necessary parts of the OS from `/etc`, `/dev`, `/bin`, and so on be copied into this phony root directory and is an ongoing administrative hassle.*

The check performed by **ls_do_checkroot** will fail on a diskless system. This prevents diskless systems from acting as vendor daemon. Running license daemons on diskless nodes is not recommended, but if you choose to support this, you need to set **ls_do_checkroot** to `0`.

For improved security, set **ls_do_checkroot** to `1`. To minimize confusion and support calls when your users are running on diskless nodes, set **ls_do_checkroot** to `0`.

ls_dump_send_data

```
int ls_dump_send_data = 0;
```

This variable controls the debug output of transmitted daemon data. It should normally be left set to 0.

ls_hud_hostid_case_sensitive

```
int ls_hud_hostid_case_sensitive = 0;
```

This variable controls the case sensitivity for hostid types HOSTNAME, DISPLAY, and USER in the context of a served, node-locked license during hostid authentication. When **ls_hud_hostid_case_sensitive** is set to 0, the values for HOSTNAME, DISPLAY, and USER hostids are treated as case insensitive. When set to 1, these values are treated as case sensitive.

The setting of this variable does not control case sensitivity in the context of verifying host names, displays, or user names for the purposes of duplicate grouping.

The default setting is 0 (case insensitive).

ls_ignore_msg_encryption

```
unsigned int ls_ignore_msg_encryption = LM_IGNORE_MSG_ENCRYPT_DISABLED;
```

Use ls_ignore_msg_encryption to disable message obfuscation for certain APIs and attributes (listed in the table below) that share license file contents. This vendor variable enables backward compatibility for older clients (before FlexNet Publisher 11.19.2) that do not support message obfuscation. It will be applied to all the clients, if set.

The variable ls_ignore_msg_encryption is applied only if message obfuscation is set to targeted with the lmgrd command line argument -m or the vendor variable ls_message_security.

Table 5-15 ▪ Variable Settings

Variable Setting	Description
LM_IGNORE_MSG_ENCRYPT_DISABLED	Message obfuscation enabled for all messages (default)
LM_IGNORE_FEAT_LIST_API	Message obfuscation disabled for lc_feat_list
LM_IGNORE_NEXT_CONF_API	Message obfuscation disabled for lc_next_conf
LM_IGNORE_GET_CONFIG_API	Message obfuscation disabled for lc_get_config
LM_IGNORE_USERLIST_API	Message obfuscation disabled for lc_userlist
LM_IGNORE_CHECKOUTFILTER_ATTR	Message obfuscation disabled for LM_A_CHECKOUTFILTER
LM_IGNORE_CHECKOUTFILTER_EX_ATTR	Message obfuscation disabled for LM_A_CHECKOUTFILTER_EX
LM_IGNORE_CHECKOUTFILTERLAST_EX_ATTR	Message obfuscation disabled for LM_A_CHECKOUTFILTERLAST_EX

Example

The following setting disables message obfuscation for lc_feat_list and LM_A_CHECKOUTFILTER:

```
ls_ignore_msg_encryption = (LM_IGNORE_FEAT_LIST_API | LM_IGNORE_CHECKOUTFILTER_ATTR) ;
```

See Also
ls_message_security

ls_message_security

```
LM_MSG_SEC ls_message_security = LM_MSG_SEC_NONE;
```

This variable sets the level for the message obfuscation on the vendor daemon side. This corresponds to the `lmgrd` command-line parameter `-m`.

Table 5-16 ▪ Variable Settings

Variable Setting	Description
none	The security level is set by `lmgrd` or, failing that, disabled.
disabled	No message body obfuscation and sequencing checks are performed (default).
permissive	Message body obfuscation and sequencing is enforced if supported.
targeted	Message body obfuscation and sequencing is enforced for certain messages.
enforcing	Message body obfuscation and sequencing is enforced.

At runtime, the vendor daemon sets the value equivalent to the `lmgrd` command line argument `-m`, if set for `lmgrd`. Otherwise, the variable is set to `disabled` (default value). Only the value `none` can be overwritten by the `-m` argument.

See Also
ls_ignore_msg_encryption

ls_no_ipaddress_in_server_cache

```
ls_no_ipaddress_in_server_cache = 0;
```

This vendor variable returns borrowed licenses if there is a mismatch between IP address between borrowing and returning the license. When this variable is set to 1 then the license server ignores IP address comparison while returning the borrowed licenses.

The default setting is 0.

ls_server_override_client_tcp_timeout

```
ls_server_override_client_tcp_timeout = 0;
```

This vendor variable provides an option for the vendor daemon to specify how long (in seconds) the license server waits since it received the last heartbeat before it determines that the client is disconnected and checks in all licenses associated with that client. This option overrides the **LM_A_TCP_TIMEOUT** attribute's value set in the client application.

Valid values (in seconds) that can be set for the `ls_server_override_client_tcp_timeout` are from 60 to 15300.

If the `ls_prefer_feature_timeout_over_tcp` is set, or if the **LM_A_TCP_TIMEOUT** `attribute` is turned off in the client application, then the `ls_server_override_client_tcp_timeout` is ineffective.

Note ▪ *If the connection is still alive after the `ls_server_override_client_tcp_timeout` value, then the `ls_server_retry_cnt_after_tcp_timeout` value (number of TCP keepalive probe retries) are sent, with an interval between each retry probe attempt defined by the `ls_server_retry_intvl_after_tcp_timeout` before closing the connection. If the connection is lost or dead even before the `ls_server_override_client_tcp_timeout` value, no probe retries are send. It is benificial to configure the `ls_server_override_client_tcp_timeout` value more than the Heartbeat Interval value.*

ls_server_retry_cnt_after_tcp_timeout

```
ls_server_retry_cnt_after_tcp_timeout = 2;
```

This varaible defines the number of tcp keepalive probes that should be sent before considering the connection as lost or dead. This vendor variable is equivalent to the TCP keepalive probe count. It is effective only, if the `ls_server_override_client_tcp_timeout` is set.

Valid values that can be set for the `ls_server_retry_cnt_after_tcp_timeout` are from 1 to 30. In Windows, the operating system sets this value to 10 as a fixed, non-configurable setting.

If the `ls_prefer_feature_timeout_over_tcp` is set, then the `ls_server_retry_cnt_after_tcp_timeout` is ineffective.

Caution ▪ *Improper configuration of the TCP keepalive settings can lead to issue like network congestion due to excessive traffic and inefficient utilization of server resources. Hence, precise configuration of these parameters are essential.*

ls_server_retry_intvl_after_tcp_timeout

```
ls_server_retry_intvl_after_tcp_timeout = 15;
```

This variable defines the time interval (in seconds) between the successive tcp keepalive probes on an active network connection to ensure it's still functional. This vendor variable is equivalent to the TCP keepalive probe interval. It is effective only, if the `ls_server_override_client_tcp_timeout` is set.

Valid values (in seconds) that can be set for the `ls_server_retry_cnt_after_tcp_timeout` are from 5 to 1200.

If the `ls_prefer_feature_timeout_over_tcp` is set, then the `ls_server_retry_intvl_after_tcp_timeout` is ineffective.

Caution ▪ *Improper configuration of the TCP keepalive settings can lead to issue like network congestion due to excessive traffic and inefficient utilization of server resources. Hence, precise configuration of these parameters are essential.*

ls_prefer_feature_timeout_over_tcp

```
int ls_prefer_feature_timeout_over_tcp = 0;
```

This variable enables the feature timeout preference over tcp timeout when feature timeout and tcp timeout are given. Feature timeout can be set with OPTIONS file keywords TIMEOUT/TIMEOUTALL. The tcp timeout can be customized with client side attribute **LM_A_TCP_TIMEOUT**. The behavior of the vendor variable is:

- When the variable is set to 0, the TIMEOUT/TIMEOUTALL or tcp timeout whichever value is smaller takes preference over others.
- When the variable is set to 1, the TIMEOUT/TIMEOUTALL takes preference over tcp timeout irrespective of its value.

Note ▪ *Value of the tcp timeout can be calculated by the following:*

*tcp timeout = `ls_server_override_client_tcp_timeout` + (`ls_server_retry_intvl_after_tcp_timeout` * `ls_server_retry_cnt_after_tcp_timeout`).*

ls_single_xver_signature

```
int ls_single_xver_signature = 0;
```

The X-version signature is used to customize X-version usage for legacy clients by serving all the checkout requests from legacy clients. The behavior of the vendor variable is described in the following:

- When the variable is set to 0, multiple versions of the FlexEnabled application, each based on a different version of FlexNet Publisher, take a designated version mentioned in the feature line and can be supported with one license file. The default value is 0.

  Example: To support four separate version signatures (current, v7.1, v8.1, and v11.4), the feature definition line looks something like this:

  ```
  FEATURE f1 demo 1.0 permanent 5 TZ=SERVERTZ SIGN=0 V7.1_SIGN=0 V8.1_SIGN=0 V11.4_SIGN=0.
  ```

- When the variable is set to 1, the cross-version mentioned in the `FEATURE` line serves all the checkout requests from legacy clients irrespective of the client version.

  Example: To support four separate version signatures (current, v7.1, v8.1, and v11.4), the feature definition line looks something like this:

  ```
  FEATURE f1 demo 1.0 permanent 5 TZ=SERVERTZ SIGN=0 V11.4_SIGN=0.
  ```

ls_ts_recovery

```
void (*ls_ts_recovery)() = 0;
```

To install a vendor-defined automatic recovery mechanism callback routine, initialize `ls_ts_recovery` with a pointer to your routine. The automatic recovery callback is called with no parameters. The callback may be called when vendor daemon reads trusted storage and all the fulfillment records including the fulfillments activated from Backoffice server are untrusted.

Note ▪ *Follow the Best Practices for Recovering Trusted Storage in White Paper, which provides the details of configuration and usage of ls_ts_recovery.*

See Also

White Paper: Best Practices for Recovering Trusted Storage

ls_use_all_feature_lines

```
int ls_use_all_feature_lines = 0;
```

The variable causes your vendor daemon to process every `FEATURE` line in the license file as an `INCREMENT` line.

With **ls_use_all_feature_lines** set to a non-zero value, any old `FEATURE` lines which you may have shipped will now be "legal," so, for example, if you had shipped a `FEATURE` line with a count of 5, then upgraded it with a new line with a count of 7, your users would now be able to use 12 licenses.

Note ▪ *License borrowing depends on the `INCREMENT` line, so if you use* **ls_use_all_feature_lines***, then license borrowing will not be available to you.*

See Also
FEATURE and INCREMENT Lines

ls_use_exclusive_commRev4

```
int ls_use_exclusive_commRev4 = 0;
```

This variable can be set to 1 for vendor daemon to authenticate and communicate with commRev4 messages only.

ls_user_lockfile

```
char * ls_user_lockfile = (char *)NULL;
```

By default, only one instance of a vendor daemon with the same name can run on the same system at one time; a lock file is used to prevent multiple copies from executing at the same time.

The default lock file names are—

- **UNIX**—`/var/tmp/lock`*vendor*. On some systems, including DEC Alpha, the location is `/var/tmp/.flexlm/.lock`*vendor*.

  Note ▪ *It is not recommended to have /var/tmp on a shared network drive for FlexNet Publisher licensing.*

- **Windows**—`C:\flexlm\`*vendor*

—where *vendor* is the vendor daemon name, as on the `VENDOR` line in the license file.

To change the location of the lock file, set **ls_user_lockfile** to the new location. If **ls_user_lockfile** is NULL, the default lock file will be used. Express the lock file name as the file's full path name.

Usage Guidelines for Vendor Daemon Synchronization

Windows Platforms

- If one or more vendor daemons involved in the synchronization are earlier than version 8.2, use this mechanism for all daemons in the group.
- If all vendor daemons involved in the synchronization are version 8.2 or later, see ls_user_lock (Windows only).

UNIX Platforms

- Use this mechanism regardless of vendor daemon version.

ls_user_lock (Windows only)

```
char *(*ls_user_lock)() = NULL;
```

By default, only one instance of a vendor daemon with the same name can run on the same system at one time. To prevent multiple vendor daemons that have different vendor names from running simultaneously, use this callback mechanism.

Initialize **ls_user_lock** with a pointer to your routine that returns a string containing either one lock name or a list of space delimited lock names. The lock names are used in a lock-out mechanism to prevent multiple copies of the vendor daemon from running on the same system. Lock names have the following restrictions:

- The maximum number of lock names supported per vendor daemon is 30.
- Each lock name can contain a maximum of 179 characters and any ASCII character except "/", "\", or space.
- Lock names are case sensitive.

Any other vendor daemon using one or more of these lock names is prevented from running on the same system at the same time. The default value includes one lock name: the name of the vendor daemon. If **ls_user_lock** is NULL, then this default lock name is used.

This mechanism is provided as a callback so that the set of lock names can be constructed at run time, rather than appearing as a literal string in the binary for a hacker to find and change.

Example

The following code is an example of a vendor-defined callback function. This example returns a string of three lock names: TOM, DICK, and HARRY.

```
char* MyCallBackFunction()
{
/* Return a string constructed dynamically at runtime.
   For the purposes of this example, a static string is returned.
*/
    return ("TOM DICK HARRY");
}
```

This callback is registered using the **ls_user_lock** identifier:

```
char *(*ls_user_lock) () = MyCallBackFunction;
```

The callback function can be included directly in `machind\lsvendor.c` or can be linked in from another object file during the vendor daemon build process.

Usage Guidelines for Vendor Daemon Synchronization

Windows Platforms

- If all vendor daemons involved in the synchronization are version 8.2 or later, use this mechanism for all daemons in the group.
- If one or more vendor daemons involved in the synchronization are earlier than version 8.2, see ls_user_lockfile.

UNIX Platforms

This mechanism is not applicable; see ls_user_lockfile.

6

Creating a Common Vendor Daemon

This chapter is for publishers who have purchased the Common Vendor Daemon option and who want to deploy a single vendor daemon that serves features previously authenticated by one or more vendor daemons. This scenario typically applies to publishers who have acquired companies, where the acquired companies have previously deployed FlexEnabled applications with their own vendor daemons.

Benefits of a common vendor daemon include:

- The publisher has one vendor daemon to maintain that serves features with different vendor names and different encryption seeds.
- The publisher can form a single corporate identity without having to redeploy the acquired company's vendor daemons.
- The publisher can combine all authentication methods configured in currently deployed vendor daemons. Going forward, the common vendor daemon supports the sum of all currently deployed authentication methods without the need to redeploy FlexEnabled applications or license files.
- The user has one vendor daemon to manage.
- Feature names from both the acquiring and acquired company can be components in a single PACKAGE line.
- A common vendor daemon can serve currently deployed FlexEnabled applications with existing license files and with various combinations of authentication methods.

When vendor daemons use trusted storage for license rights, they cannot be combined into a common vendor daemon. A common vendor daemon does not support the use of trusted storage.

Common Vendor Daemon Terminology

For the discussion in this chapter, it is helpful to use the following terminology when referring to common vendor daemon configuration and deployment. These terms are presented from an acquiring company scenario.

Primary Vendor Terminology

In general, the term *primary* is used to denote the vendor capabilities that will represent the combined company. In a common vendor daemon configuration, there is one primary vendor.

Table 6-1 ▪ Primary Vendor Terminology

Term	Definition
primary vendor name	The vendor name, as registered with Revenera, for the company that has acquired one or more companies each having deployed FlexEnabled applications.
primary feature name	One or more feature names supported by the acquiring company.
primary package name	One or more package names supported by the acquiring company, each of which is composed solely of components with primary feature names.
primary authentication method	The authentication method or methods employed by the primary vendor, which include: ● License key—12- or 20-character ● Non-TRL digital signature ● TRL signature—including 113-, 169-, and 239-bit strengths
primary vendor daemon	The vendor daemon that serves primary licenses to primary applications only. This vendor daemon is from the acquiring company.
primary vendor daemon configuration	Configuration settings in the primary vendor's `lsvendor.c` consisting of both global variable settings and any registered callback functions.

Secondary Vendor Terminology

In general, the term *secondary* is used to denote the vendor capabilities they were identified previous to the merger. There can be one or more secondary vendors in a common vendor daemon configuration.

Table 6-2 ▪ Secondary Vendor Terminology

Term	Definition
secondary vendor name	The vendor name, as registered with Revenera, for a specific acquired company, having deployed FlexEnabled applications.
secondary feature name	One or more feature names supported by a specific acquired company.
secondary package name	One or more package names supported by a specific acquired company, each of which is composed solely of components with secondary feature names.

Table 6-2 ▪ Secondary Vendor Terminology

Term	Definition
secondary authentication method	The authentication method or methods employed by a specific secondary vendor, which include ● License key—12- or 20-character ● Non-TRL digital signature ● TRL signature—including 113-, 169-, and 239-bit strengths
secondary vendor daemon	The vendor daemon that serves secondary licenses to secondary applications, only. This vendor daemon is from a specific acquired company.
secondary vendor daemon configuration	Configuration settings in a specific secondary vendor's `lsvendor.c` consisting of both global variable settings and any registered callback functions.

Common Vendor Terminology

The term *common* is used to denote FlexNet Publisher components that contain both primary and secondary information.

Table 6-3 ▪ Common Vendor Terminology

Term	Definition
common package name	A package name whose `PACKAGE` line contains both primary and secondary feature names as components.
common package license certificate	A `PACKAGE` line containing both primary and secondary feature names as components and signed with corresponding primary and secondary signatures.
common license generator	A license generator configured with both primary and secondary seeds. It signs primary license files with primary signatures, signs secondary license files with secondary signatures, and signs common package license files with both primary and secondary signatures.
common vendor daemon	The vendor daemon of the acquiring company that serves primary licenses to primary applications, secondary licenses to secondary applications, and common package licenses to both primary and secondary applications. It subsumes the primary vendor name.
common vendor daemon configuration	The changes made in `lsvendor.c` by the acquiring company to merge the primary and all secondary vendor daemon configurations.
common license server manager	The license server manager `lmadmin` (or `lmgrd`) that dispatches requests from both primary and secondary applications to a common vendor daemon.

Table 6-3 ▪ Common Vendor Terminology

Term	Definition
cvd_platform **cvd_machind**	The `<platform_dir>` and `machind` directories for the resulting common vendor daemon toolkit.

Planing Your Common Vendor Daemon Configuration

Deploying a common vendor daemon is summarized in the following steps, each of which is described in subsequent sections of this chapter.

1. Decide which vendor daemons you want to participate in the common vendor daemon configuration. Choose which is the primary and which are the secondary. See Deciding Participating Vendor Daemons for considerations.
2. Note the current location in your enterprise where each participating vendor daemon is configured and built.
3. Migrate the primary daemon to the current release of the FlexNet Publisher. See "Rebuilding the Production Licensing Toolkit" in the *Development Environment Guide* for details.
4. Configure the common vendor daemon toolkit using the `cvdconfig` utility. See Configuring the Common Vendor Daemon Toolkit for details.
5. Build the common vendor daemon and associated FlexNet Publisher components based on the configuration in the previous step.
6. Regenerate your license files, as needed, to account for the new common vendor daemon. See Building the Common Vendor Daemon for further information.
7. Deploy the following to your user:
   - Your common vendor daemon
   - New license files
   - FlexNet Publisher utilities (version 10.8 or later)
   - `lmadmin` (or `lmgrd` version 10.8 or later)
   - Appropriate installation instructions

Your user must use an appropriate version of the license server manager and must disable all older vendor daemons corresponding to those participating in the common vendor daemon.

Deciding Participating Vendor Daemons

Survey the set of candidate vendor daemons that you want to participate in the common vendor daemon. For each candidate vendor daemon, determine the following:

- The feature names and package names that it serves.

- Its vendor daemon settings found in `lsvendor.c`.
- All `ENCRYPTION_SEEDs` and any `LM_SEEDs`, as applicable.

Decide which vendor daemon is the primary and which is the secondary.

Determining Conflicts

You need to determine if there are conflicts between any participating vendor daemon's configuration and resolve them before proceeding. Check the following categories for conflicts:

- Duplicate feature names between vendor daemons
- Duplicate package names between vendor daemons
- Conflicting settings for the following vendor daemon configuration variables in `lsvendor.c`:
  - `(*ls_user_init1)()`
  - `(*ls_user_init2)()`
  - `(*ls_user_init3)()`
  - `ls_dump_send_data`
  - `ls_minimum_user_timeout`
  - `ls_min_lmremove`
  - `ls_do_checkroot`
  - `(*ls_borrow_out)()`
  - `(*ls_borrow_in)()`
  - `(*ls_borrow_init)()`

Configuring the Common Vendor Daemon Toolkit

The `cvdconfig` utility, provided in the `<platform_dir>` directory, is used to produce a production toolkit configured for the common vendor daemon. Driven by a configuration file, it is run from the `<platform_dir>` directory of the primary vendor daemon and reconfigures its toolkit, combining configuration data from each of the secondary vendor daemons. The result is one `makefile` that is used to build the common vendor daemon and all corresponding FlexNet Publisher utilities and applications.

Preparing the Configuration File

The configuration file consists of a set of vendor daemon specifications: one for each secondary daemon. A specification has the following form:

```
VENDOR=vendor_daemon
{
    ALIAS=alias_name
```

```
    CODEFILE=path_to_lm_code.h
    LMSEEDS=path_to_lmseeds.h
    LSVENDOR=path_to_lsvendor.c
}
```

where:

Table 6-4 ▪ Vendor Daemon Specifications

Specification	Description
vendor_daemon	The vendor daemon name for which this specification applies.
alias_name	Another name you can assign to the vendor daemon. This is useful in cases where you want to rebrand or otherwise add your own identity to a vendor daemon name from an acquired company.
path_to_lm_code.h	The full path to the vendor daemon's `lm_code.h` file.
path_to_lmseeds.h	The full path to the vendor daemon's `lmseeds.h` file
path_to_lsvendor.c	The full path to the vendor daemon's `lsvendor.c` file

Configuring the Toolkit

After you have built the configuration file, you can use the `cvdconfig` utility to build the common vendor daemon toolkit. From the primary vendor daemon's `<platform_dir>` directory, type:

`cvdconfig -f` ***`my_config_file`***

This command does the following:

- In the `machind` directory, the original `lm_code.h`, `lmseeds.h`, and `lsvendor.c` files for the primary vendor daemon are copied to corresponding files with `.save` appended to the name.
- New, all-inclusive versions of `lm_code.h`, `lmseeds.h`, and `lsvendor.c` are created in `machind` that contain toolkit configuration details for all participating vendor daemons. These new files are prepended with the primary vendor daemon name.
- The toolkit `makefile` is configured to make the common vendor daemon. Intermediate files created during the configuration process are prepended with alias names for participating vendor daemons.

Vendor-specific Toolkit Modifications

The `cvdconfig` utility configures the standard toolkit as included in the FlexNet Publisher as shipped from Revenera. Specific changes and additions you have made to the toolkit are not copied to the new, common vendor daemon toolkit. Consider the following types of modifications you may have made to any of your secondary vendor daemon toolkits. You need to manually copy these modifications to the newly created common vendor daemon components.

- **Changes and additions to the makefile in the <platform_dir> directory**—Make corresponding changes to the new common `makefile` keeping in mind naming conventions for secondary vendor daemon intermediate files.
- **Callback routines added to lsvendor.c**—Copy over existing callback routines to the new `lsvendor.c`. Keep in mind any conflicts that may exist for the same variable as detailed in Determining Conflicts. If conflicts exist, you need to merge the code from all callback routines into one routine for use by all secondary vendor daemons.
- **Vendor-specific files**—Any custom files you have added to the secondary vendor toolkits must be copied to the new common toolkit area.

Configuration File Example

Consider the following scenario:

- Company ABC acquires Company X and Company Y.
- Company ABC's currently deployed vendor daemon is named `vendor_abc` and is considered the primary daemon. Its corresponding toolkit is located at `c:\Vendor ABC`.
- Company X's currently deployed vendor daemon is named `vendor_x` and is one of two secondary daemons. Its corresponding toolkit is located at a remote location within the enterprise and mapped to `x:\Vendor X`.
- Company Y's currently deployed vendor daemon is named `vendor_y` and is one of two secondary daemons. Its corresponding toolkit components is located at a remote location within the enterprise and mapped to `y:\Vendor Y`.

The configuration file looks something like:

```
VENDOR=vendor_x
{
    ALIAS=alias_x
    CODEFILE="x:\Vendor X\i86_n3\lm_code.h"
    LMSEEDS="x:\Vendor X\i86_n3\lmseeds.h"
    LSVENDOR="x:\Vendor X\i86_n3\lsvendor.c"
}
VENDOR=vendor_y
{
    ALIAS=alias_y
    CODEFILE="y:\Vendor Y\i86_n3\lm_code.h"
    LMSEEDS="y:\Vendor Y\i86_n3\lmseeds.h"
    LSVENDOR="y:\Vendor Y\i86_n3\lsvendor.c"
}
```

Note ▪ *Note the following:*

- *Path names with embedded spaces must be enclosed in quotation marks.*
- *All path names must conform to formats supported by the platform on which `cvdconfig` is run. In this case, the configuration is for Windows, regardless of the location that `x:\Vendor X` and `y:\Vendor Y` represent.*

Building the Common Vendor Daemon

After you have finished the configuration, you need to build the common vendor daemon and associated FlexNet Publisher components. Proceed according to your established procedures. See "Rebuilding the Production Licensing Toolkit" in the *Development Environment Guide* for further details.

Generating License Files for Your Common Vendor Daemon

In general, the common vendor daemon is able to serve features for currently deployed FlexEnabled applications, previously authenticated by one or more vendor daemons, without the need for issuing new license files. However, the following scenarios require you to issue updated license files:

- You want to create packages composed of components from more than one of your secondary vendor daemons
- You want to replace references to vendor daemons of your acquired companies with a single name, thereby establishing one brand identity for all your FlexEnabled applications.

For the above scenarios, a single `PACKAGE` or `FEATURE` line can be constructed to support the common vendor daemon and all its secondary vendor daemons. Replace existing license key and `SIGN=` keywords in your license files with a single `AUTH` keyword construct according to the following syntax:

```
AUTH={ vendor_name_1=(signature= ...) \
    vendor_name_2=(signature= ...) \
    ... \
    }
```

where:

Table 6-5 ▪ Vendor Daemon Keyword Constructs

Keyword Construct	Description
vendor_name_n=	The names of all vendor daemons that participate in the common vendor daemon configuration. The first one must be the name of the common vendor daemon.
signature=	The particular authentication methods used in the original license file for `vendor_name_n`. Include all appropriate methods, where `signature=` can be: • `SIGN=` —License signature, both TRL and non-TRL and their version-specific variants. • `LK=` —License key, including all version-specific variants.

Use `lmcrypt` from the common vendor daemon toolkit to sign the license files. Use your existing methods for reissuing license files.

Common Vendor Daemon License File Example

Consider the scenario where Company ABC acquires Company X and Company Y and wants to create a package consisting of feature `f1_abc`, feature `f2_x` from Company X, and feature `f2_y` from Company Y. The existing license files for each of the three companies are as follows:

- Company ABC, using `SIGN=` authentication:

  ```
  FEATURE f1_abc vendor_abc 1.5 permanent 9 SIGN=0
  ```

- Company X, using both license key and `SIGN=` authentication:

  ```
  FEATURE f2_x vendor_x 2.0 permanent 5 0 SIGN=0
  ```

- Company Y, using license key authentication:

  ```
  FEATURE f2_y vendor_x 1.0 permanent 15 0
  ```

The license file for the desired package, for the common vendor daemon, looks something like:

```
PACKAGE pkg_abc vendor_abc 1.0 COMPONENTS="f1_abc f2_x f2_y" \
        AUTH={ \
            vendor_abc=(SIGN=0) \
            alias_x=(LK=0 SIGN=0) \
            alias_y=(LK=0) \
            } \
FEATURE pkg_abc vendor_abc 1.0 permanent 15 SIGN=0
```

Note ▪ *Note the following:*

- *Vendor alias names (`alias_x` and `alias_y`) are used with the `AUTH` keyword in this example to hide legacy branding of acquired companies. The use of vendor alias names is optional.*
- *Currently deployed FlexEnabled applications from Company ABC, Company X, and Company Y do not have to be redeployed.*
- *The user must use common vendor daemon, `vendor_abc`, in place of vendor daemons for Company ABC, Company X, and Company Y.*

Interactions with the SUPERSEDE Keyword

If you issue `FEATURE` or `INCREMENT` lines with the `SUPERSEDE` keyword, consider the following:

- A line containing the `AUTH=` keyword supersedes a line that does not have the `AUTH=` keyword as long as one of the vendor daemon names listed with the `AUTH=` keyword matches the vendor daemon name in the line to be superseded. This applies to the `SUPERSEDE` keyword only.
- Given two lines, both with the `SUPERSEDE` and `AUTH=` keywords, there must be at least one vendor daemon name in common in each `AUTH=` keyword for the supersede operation to have effect.

Deploying the Common Vendor Daemon

Deploy the common vendor daemon and associated FlexNet Publisher components according to your established procedures with the following considerations:

- Your user must use `lmadmin` (or `lmgrd` version 10.8 or later).
- All constituent secondary vendor daemons must be disabled. The common vendor daemon executes in their place.
- In license files, the `VENDOR` line corresponding to the common vendor daemon must be located before the feature definition lines. `VENDOR` lines corresponding to secondary vendor daemons must be removed from license files.
- The user may maintain separate options files, one for each secondary vendor daemon. However, if there are any conflicts, such as the debug log file path name specification, the last options file read takes precedence. Alternatively, all options files may be combined.
- The common vendor daemon generates report logs for the primary vendor name. Usage for both primary and secondary feature names is logged.
- The common vendor daemon generates debug logs for the primary vendor name. Information for both primary and secondary feature names is logged.

Restoring the Original Primary Vendor Toolkit

The common vendor toolkit is configured in the `<platform_dir>` directory of the primary vendor daemon. You can revert the common vendor toolkit back to the original, primary vendor toolkit. This operation reverts all toolkit files to their original state; the common vendor toolkit is no longer available. From the primary vendor daemon's `<platform_dir>` directory, type:

```
cvdconfig -revert
```

This command does the following:

- In the `machind` directory, the original `lm_code.h`, `lmseeds.h`, and `lsvendor.c` files are restored.
- In the `<platform_dir>` directory, the original, primary toolkit `makefile` is restored.

7

Secure Communication

The secured communication is an encrypted communication between the client and vendor daemon using FlexNet Publisher protocol over TCP. The FlexNet Publisher has introduced new Encrypted Communication (Ecomms) layer at application level based on TLS 1.3. The Ecomms helps communicating data in ciphertext between client and vendor daemon.

The Ecomms layer links OpenSSL or other native SSL libraries for encrypting the communication. The use of Ecomms layer is an optional feature. Producers can continue building their FlexEnabled clients and vendor daemons, without any direct linkage to OpenSSL or other encrypted communication libraries.

This section describes how the secure communication happens via Ecomms.

Introduction

The default communication between the client and vendor daemon using FlexNet Publisher protocol over TCP is with plaintext. FlexNet Publisher provides the Ecomms-enabled vendor daemon for secured communication. The existing client can decide whether to use ciphertext communication or continue with plaintext communication based on the attribute set.

Enabling Ecomm Functionality

The pre-requisites when a producer chooses to allow encrypted communication are:

- the vendor daemon has to be built with Vendor Variable *`ls_secure_comms`* set to 1 (default 0).
- FlexEnabled client is required to be built with the attribute **LM_A_SECURE_COMMS** to 1 (default 0). This indicates that the client wants to use encrypted communication.

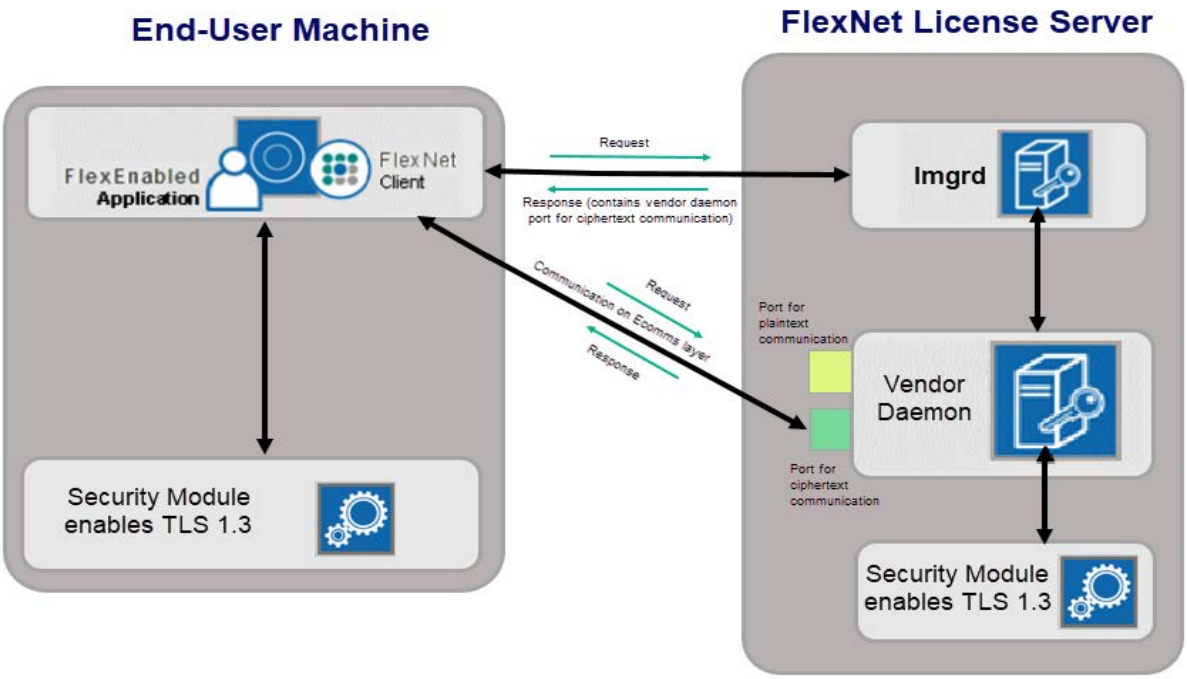
FlexEnabled client sends connection request that contains **LM_A_SECURE_COMMS** value. Depending on the shared value, lmgrd directs the client to the port for ciphertext or plaintext communications (as shown in Figure 7-1). A license administrator can specify the port for the ciphertext communication with keyword *`eport`* in vendor line mentioned in the license file.

After the port discovery for ciphertext communication, the client re-initiates secured connection and start communicating with vendor daemon available port.

Vendor daemon enabled with secured communication can also communicate with the existing client on plaintext communication.

The error codes, `LM_ECOMMS_ERROR` and `LM_NO_ECOMMS_SUPP` have been introduced to indicate a secure communication failure. Refer *C/C++ Function Reference guide* for more details.

Figure 7-1: Block Diagram for the Secured Communication



Generating the Dynamic libfnpssl Library

For secure communication, the FlexNet Publisher toolkit provides a flexibility to generate a dynamically linked libfnpssl library that can dynamically link to OpenSSL binaries.

By default, the FlexNet Publisher toolkit includes a statically and tightly linked OpenSSL binaries with libfnpssl library. If you want to generate a dynamic version of libfnpssl, perform the following steps based on your platform:

- Steps to Generate the Dynamic libfnpssl on Windows Platform
- Steps to Generate the Dynamic libfnpssl on Linux (Intel) Platform
- Steps to Generate the Dynamic libfnpssl on Solaris Platforms

Steps to Generate the Dynamic libfnpssl on Windows Platform

1. Go to the FlexNet Publisher toolkit platform directory.

2. Execute the following command to generate a dynamically linked `libfnpssl.dll` that will search for dependent libraries and/or DLLs (libssl.lib, libcrypto.lib) in the standard system directories:
   `$ nmake ssl_library_local`

Note ▪ *The statically linked `libfnpssl will be removed` once dynamic version of libfnpssl has been generated. If you need statically linked `libfnpssl, you can regenerate` by executing the command "`$ nmake ssl_library`".*

Steps to Generate the Dynamic libfnpssl on Linux (Intel) Platform

1. Go to the FlexNet Publisher toolkit platform directory.
2. Execute the following command to generate a dynamically linked `libfnpssl.so` that will search for dependent Shared object files(`libssl`.so and `libcrypto.so`) in standard system directories:
   `$ make DYNSSL=1 ssl_library`
3. Execute the following command to generate a dynamically linked `libfnpssl.so` that will search for `libssl.so` and `libcrypto.so` in `/path/to/lib`:
   `$ make DYNSSL=1 DYNSSLDIR=/path/to/lib ssl_library`
4. Execute the following command to regenerate a statically linked `libfnpssl.so`:
   `$ make ssl_library`

Note ▪ *The statically linked `libfnpssl will be removed` once dynamic version of libfnpssl has been generated. If you need statically linked `libfnpssl, you can regenerate` by executing the command "`$ nmake ssl_library`".*

Steps to Generate the Dynamic libfnpssl on Solaris Platforms

1. Go to the FlexNet Publisher toolkit platform directory.
2. Execute the following command to generate a dynamically linked `libfnpssl.so` that will search for dependent Shared object files (libssl.so, libcrypto.so) in standard system directories:
   `$ make ssl_library_local`

Note ▪ *The statically linked `libfnpssl will be removed` once dynamic version of libfnpssl has been generated. If you need statically linked `libfnpssl, you can regenerate` by executing the command "`$ nmake ssl_library`".*

While starting vendor daemon with secure communication, if the following error messages observed in the vendor log:

- Ecomms: SSL/TLS initialization failed
- Ecomms: Startup failed, res=-279

Then, make sure the `libfnpssl.dll/so` file exists in FlexNet Publisher kit platform directory and the dependent ssl/crypto libs (libssl.lib, libcrypto.lib) and dlls/so are present in the desired locations/path.

The desired location/path would be any one of the following:

- Standard system directories.
- Current working directory where vendor daemon executable present.

- The path set in environment variables and library paths.

See Also
LM_A_SECURE_COMMS attribute in *C/C++ Function Reference*
ls_secure_comms vendor variable in *Programming Reference for License File–Based Licensing*
eport using in Options file in *Programming Reference for License File–Based Licensing*
LM_ECOMMS_ERROR and LM_NO_ECOMMS_SUPP in *C/C++ Function Reference*

Limitation

- Not supported on the macOS, Linux (ARM), AIX, and HP-UX platforms. The behavior will be undefined if the feature is used on unsupported platform.
- This feature is not supported with lmadmin server. The support will be introduced in the future release.

8

The License File: Overview

This chapter provides an overview of the license file format and examples of license file types. When you have an idea of the license model for your FlexEnabled application, go to the section titled License File Syntax for specific syntactical information.

License File Format

A license file consists of the following sections:

- **SERVER line**—This line appears in the license file if a license server is used. The `SERVER` lines contain information about the systems where the license server manager is running.
- **VENDOR line**—This line appears in the license file if a license server is used. The `VENDOR` line contains information about the vendor daemon that runs on the license server.
- **USE_SERVER line**—A `USE_SERVER` line, if used, usually follows the `SERVER` line and indicates that a FlexEnabled application should not process the rest of the license file itself, but should check out the license directly from the license server.
- **PACKAGE line**—This section defines how individual FlexEnabled components are grouped into packages.
- **Feature definition lines**—This section includes information that entitles feature usage, whether the features are used individually or as components in a package. This information is required in a license file read directly by the license server or otherwise does not contain a `USE_SERVER` line. The following types of feature definition lines are described in this section:
  - `FEATURE` line—Defines the individual feature.
  - `INCREMENT` line—Adds additional configuration to a `FEATURE` or `INCREMENT` line with the same feature name.
  - `UPGRADE` line—Upgrades the licenses for an existing feature name.

Vendors and license administrators read the license file to understand how the license behaves; for example, what features are FlexEnabled, the number of licenses, whether these features are node-locked, if the features are demo or regular, and so on.

The SERVER hostids and all keywords in the feature definition line, except the vendor daemon name and lowercase keyword=value pairs, are encrypted into the signature for that feature definition line. If this information is changed, an LM_BADCODE error is returned when the FlexEnabled application tries to check out a license for that feature.

These are the only data items in the license file that can be edited by the license administrator:

- Host on the SERVER line
- Port numbers on the SERVER or VENDOR lines
- Path names on VENDOR lines
- Options file path names on VENDOR lines
- Lowercase keyword=value pairs on feature definition lines

Any amount of white space can separate the components of license file lines. Data can be entered using any plain text editor. Publishers can therefore distribute license data over fax or telephone.

Eight-bit, Latin-based characters are fully supported in license files, options files, log files, and client environments.

See Also
Software Publisher Utility Programs for information on the license generation utility lmcrypt
lc_cryptstr in the *C/C++ Function Reference* for generating licenses with a C function call

Hostids for Supported Systems

A hostid is a means used to uniquely identify a specific system. A hostid is used in the following contexts:

- **License**—The license is bound to one or more hostids. They are used to define which FlexEnabled client can run your application.
- **License server**—A hostid used to define which system is authorized to run a license server that serves licenses to your FlexEnabled application.

Each platform supports one or more methods of determining its hostid. You need to determine which hostid types you accept for each platform you support.

The lmhostid utility prints the default hostid that FlexNet Publisher expects to use on any given platform. FlexNet Publisher supports platform-specific as well as vendor-defined hostids. In addition, there are a number of special hostid types that apply to all platforms. These special hostid types can be used on either a SERVER line or a feature definition line, wherever a hostid is required. Vendor-defined hostids are discussed in Using Vendor-Defined Hostids on page 184.

The decision to use a particular hostid is made in the license file rather than the FlexEnabled application. However, there are occasions when the FlexEnabled application may want to determine a hostid type specified in a particular feature definition line, for example, to detect the usage of special hostid types such as DEMO and ANY. The Flexible API function, **lc_auth_data,** called by the application, provides this information.

See Also
Hostids
SERVER Lines
FEATURE and INCREMENT Lines
Intel Pentium III Hostid (HOSTID_INTEL)
The following Flexible API functions in the *C/C++ Function Reference*:
lc_auth_data
l_new_hostid
lc_free_hostid

License File Types

The information in the license file affects how the contents are interpreted. The following sections show examples of the following types of licenses. See the `/examples/licenses` directory in the toolkit for additional examples.

- Simple, Uncounted License
- Simple, Expiring License
- Simple Floating (Counted) License
- Floating with Three-Server Redundancy

Simple, Uncounted License

The following figure shows a simple, uncounted license.

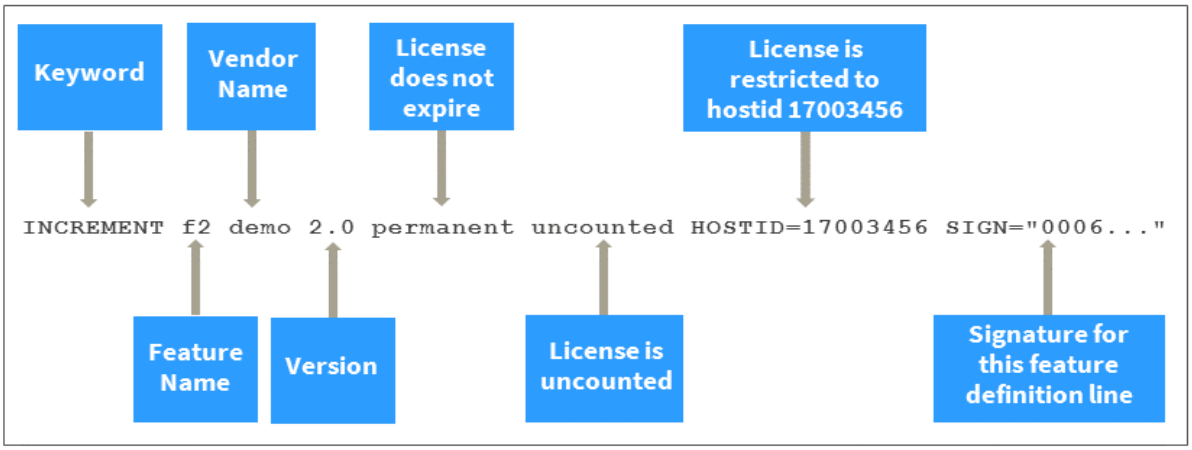


Figure 8-1: Simple, Uncounted License

- Uncounted licenses have unlimited use on the hostid specified and require no server.
- When the expiration date is `permanent` (or if a date is specified with a year of "`0`"), the license never expires.
- This license supports versions 0.0–2.0 (inclusive).

Simple, Expiring License

The following figure shows an expiring demo license.

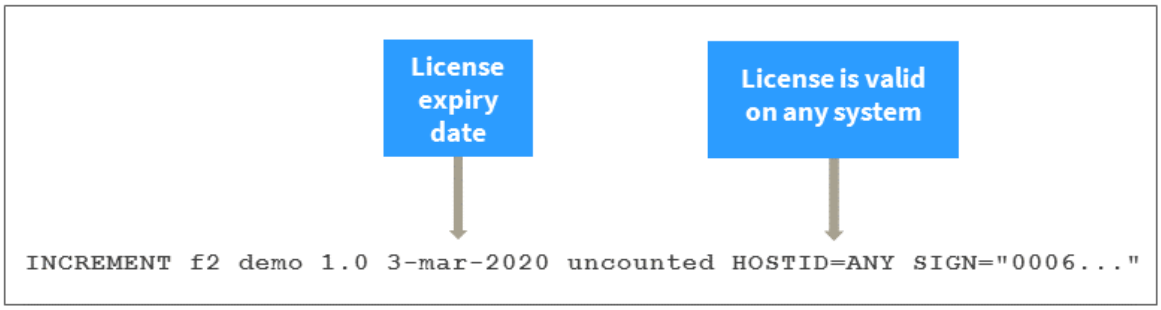


Figure 8-2: Expiring License

- This license expires on March 3, 2020. Specify the year using four digits, as in `1-jan-2018`.
- The `ANY` hostid indicates that this license allows feature `f2` to run on any system.

Simple Floating (Counted) License

The following is an example of a simple, floating (or concurrent) license:

```
SERVER myhost 17003456 2837
VENDOR demo
FEATURE f1 demo 2.0 permanent 9 SIGN=<...>
```

- `SERVER` and `VENDOR` lines are required for counted licenses
- Unexpiring
- Floating, meaning that it runs on any node. There is no hostid defined on the feature definition line.
- Limited to nine concurrent licenses
- Server restricted to hostid 17003456. To remove this restriction, use hostid of `ANY` (for example, `SERVER myhost ANY 2837`).
- A specific TCP/IP port is specified, `2837`, for the license server connection.

The breakdown of the `SERVER` and `VENDOR` lines is illustrated in the following figure.

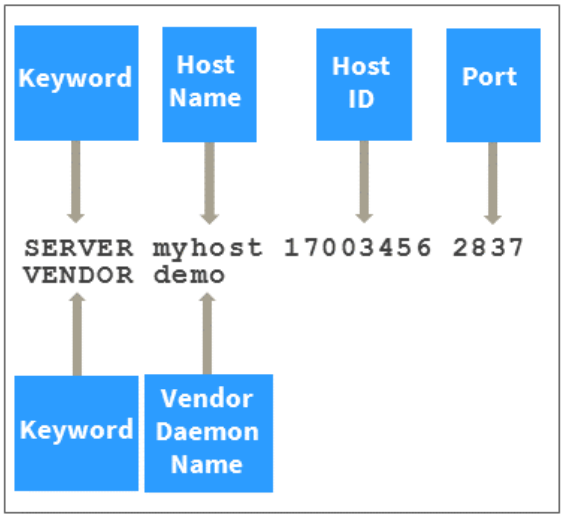


Figure 8-3: SERVER and VENDOR Lines

- The host name can be changed by the license administrator. If host name is `this_host`, clients running on the same node as the server will work. Clients on other nodes fail unless the host name is changed, or the clients use *@host* (or *port@host* if a port number is specified on the SERVER line) to find the server.
- Because a `vendor_daemon_path` is not specified in the VENDOR line, the license server manager `lmadmin` uses the license server root directory (or for `lmgrd`, the $PATH environment variable in its environment) to find the vendor daemon binary.
- Nothing else can be changed on these two lines. Everything else is authenticated by the signature.

Floating with Three-Server Redundancy

The follow is an example of a floating license configured for three-server redundancy.

```
SERVER pat 17003456 2837
SERVER lee 17004355 2837
SERVER terry 17007ea8 2837
VENDOR demo
FEATURE f1 demo 1.0 1-jan-2020 10 SIGN=<...>
FEATURE f2 demo 1.0 1-jan-2020 10 SIGN=<...>
```

A three-server redundant configuration is specified. This is a set of three server nodes, any two of which must be running for the vendor daemon to server licenses.

- Three SERVER lines and one VENDOR line required.
- All three servers communicate using the same TCP/IP port.
- Two features are FlexEnabled: `f1` and `f2`.
- Expires on January 1, 2020.
- Floating, meaning that it runs on any node. No hostid is defined on the feature definition lines.
- Limited to 10 concurrent licenses for each feature.

License in a Buffer

The license rights do not need to be defined in a license file. They can be specified as a literal string anywhere a license file name can be used (such as in a license search path). The actual text of the license file is specified with START_LICENSE\n as a prefix, and \nEND_LICENSE as suffix, where the embedded new lines are required.

In the following license search path, the third component is a literal text string containing the actual feature definition line.

```
setenv VENDOR_LICENSE_FILE file1: file2: START_LICENSE\n\
FEATURE f1 demo 1.0 permanent uncounted HOSTID=ANY \
VENDOR_STRING=\"Acme Inc\" SIGN=50A35101C0F3\n\
END_LICENSE
```

The following example shows this usage in a call to the **lc_set_attr** function inside the application:

```
lc_set_attr(job, LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)
        "START_LICENSE\n\
        FEATURE f1 demo 1.0 permanent \
        uncounted HOSTID=ANY \
        VENDOR_STRING=\”Acme Inc\" SIGN=50A35101C0F3\n\
        END_LICENSE”);
```

If all users for a particular software publisher have the same license file, it is convenient to store it in a character buffer in the program, rather than in a license file, which would require the publisher to distribute an extra file that might get misplaced.

For example, a library may be used to read a particular format file. If the file includes the name of the company that generated the data, a license can guarantee that only files generated by this company can be read by the library, by matching the name in the VENDOR_STRING=“...” field, (in conjunction with using **lc_auth_data** or LM_A_CHECKOUTFILTER).

Limitation

When the characters @ and \\ (double backward slashes) are mentioned in the license search path, the license search path is disqualified.

The following example shows the use of character combination @ and \\ (double backward slashes):

```
lc_set_attr(job, LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)
        "START_LICENSE\n\
        FEATURE f1 demo 1.0 permanent \
        uncounted HOSTID=ANY \
        VENDOR_STRING=\"@C:\\Program Files"
        END_LICENSE”);
```

Sorting Licenses During Initialization

When FEATURE or INCREMENT lines are loaded into memory, they are sorted based on an algorithm that looks at various attributes. During the checkout process, features that are earlier in the sort order are evaluated before those that are later in the sort order.

By default, FEATURE and INCREMENT lines from trusted storage appear at the beginning of the sort order. FEATURE and INCREMENT lines from license files appear later. Within these two groups, features are sorted by the order that each line is listed.

The following table describes the features retrieved during the initialization process.

Table 8-1 ▪ Sample Feature before Loaded into Memory

Features in Trusted Storage	Features in License Files
F1ts	F1lf
F3	F2
F5	F4

When loaded into memory, these features are sorted as follows:

Table 8-2 ▪ Sample Feature After Loaded into Memory

Features in Memory
F1ts
F3
F5
F1lf
F2
F4

You can change this sort order using the following two methods:

- Change the LM_A_SORT_TS_FIRST attribute using the **lc_set_attr** function.
- Define the sort value for one or more FEATURE or INCREMENT lines to define the position in the sort order.

Using the LM_A_SORT_TS_FIRST Attribute

By default, the LM_A_SORT_TS_FIRST attribute is set to 1. When you change this value to zero (0), the sort order rules are as follows:

1. The contents of license files are sorted, first. Features are grouped by the license file that they are read, and then they are sorted within each group. They are not merged into one single group and sorted as a whole.
2. Features are sorted based on the sort attribute for each feature. This attribute is described in the following section.
3. Features are sorted alphabetically by feature name.
4. FEATURE lines are sorted before INCREMENT lines.

5. Uncounted licenses are sorted before counted licenses.
6. Lower feature versions are sorted before higher feature versions.
7. Issued date, in reverse order. More recent dates are sorted before older dates.

   The date is taken from the `ISSUED` attribute or `START` attribute for each `FEATURE` in the license file.
8. The order that the `FEATURE` or `INCREMENT` line appears within the license file is otherwise maintained.

Using the sort Attribute

You can also change the sort order by defining the `sort` attribute for each `FEATURE` and `INCREMENT` line. The value defined for the `sort` attribute identifies the relative position of that feature in the sort order. For example, features with a `sort` value of 25 will be sorted before those with a `sort` value of 50. Unless explicitly defined, the sort value for each feature is set to 100. The value range for the `sort` attribute is 0 through 255. If you specify a value higher than 255, the encryption process converts it to 255.

There is an interdependency between the `sort` attribute and the `LM_A_SORT_TS_FIRST` attribute. When the `LM_A_SORT_TS_FIRST` is set to `1` (the default value), all `FEATURE` and `INCREMENT` lines obtained from trusted storage are sorted as if they had `sort` value of zero (`0`). This ensures that they will appear first in the sort order. If you want `FEATURE` and `INCREMENT` lines from license files to appear before those from trusted storage, you can accomplish this in the following ways:

- Disable the `LM_A_SORT_TS_FIRST` attribute by setting it to `0` and then define the sort attribute to the `FEATURE` and `INCREMENT` lines of interest. See the previous section for the sort order rules when this attribute is set to `0`.
- Leave the `LM_A_SORT_TS_FIRST` attribute at the default value (`1`), and then set the `sort` attribute to `0` for each `FEATURE` and `INCREMENT` lines of interest.

9

License File Syntax

This chapter is a reference for license file syntax. It is divided into sections which correspond to the different license file sections as outlined in License File Format. For an overview of license files, see License File Types.

Table 9-1 lists the license file sections described in this chapter.

Table 9-1 ▪ License File Sections

Section	Synopsis
General Syntax Description	Syntax rules common to all sections.
Server Information	
SERVER Lines	Contains information about the systems running the license server.
VENDOR Line	Contains information about the vendor daemon that runs on the license server.
USE_SERVER Line	Forces the FlexEnabled application to access the license servers in preceding SERVER lines to check out licenses.
Package Information	
PACKAGE Lines	Provides a way to license a set of FlexEnabled components into one package.
License Rights Information	
FEATURE and INCREMENT Lines	Describes the FlexEnabled features available for the specified vendor's products.
UPGRADE Lines	Creates a new version of the license, thereby replacing the older one.

General Syntax Description

This section takes you to the following:

- Comment Lines
- Line Continuation

Comment Lines

It is a convention that comment lines begin with the pound (#) character. However, all lines not beginning with a license file keyword are ignored and are considered comment lines.

Line Continuation

Long lines are broken up by a back-slash (\), the line continuation character. The following example shows use of the line continuation character.

```
FEATURE f1 demo 1.0 permanent 5 HOSTID=adfe2345 \
        SIGN=123456789012
```

SERVER Lines

`SERVER` *`host hostid`* `[`*`port`*`] [PRIMARY_IS_MASTER] [HEARTBEAT_INTERVAL=`*`seconds`*`]`

A `SERVER` line specifies the name and hostid of the license server and, optionally, the TCP/IP port number through which to communicate to the license server manager `lmadmin` (or `lmgrd`). A license file may have one `SERVER` line or three `SERVER` lines (in the case of a three-server–redundant configuration). The `SERVER` node name in the license file can be any network alias for the node.

Table 9-2 ▪ SERVER Line

SERVER Line Input	Description
host	String returned by the UNIX `hostname` or `uname -n` commands, or an IP address. The license administrator can edit this value. The IP address is recommended for sites where NIS or DNS have trouble resolving a host name, or if the server node has multiple network interfaces, and hence multiple host names. `this_host` can be used when the host name is unknown. This enables the license administrator to install the application and start the license server. FlexEnabled applications on the same host as the license server work fine. FlexEnabled applications on other systems need to set `LM_LICENSE_FILE` or `VENDOR_LICENSE_FILE` to *port@host* or *@host* to find the license server, or `this_host` can simply be edited to the real host name. `lminstall` will automatically change `this_host` to the real host name when appropriate.

Table 9-2 ▪ SERVER Line

SERVER Line Input	Description
hostid	Case-insensitive string returned by the `lmhostid` command. Only one hostid is allowed. Alternate special hostids can also be specified here, including `ANY`, `HOSTNAME=host`, and so on. See the Hostids chapter for information about expected and special hostids, and Using Vendor-Defined Hostids for vendor-defined hostids. For virtualization support, refer to Chapter 22, Virtualization. ***Caution ▪*** *If the `INTERNET` hostid is used on the `SERVER` line, wildcards should not be allowed in the IP address. If wildcards are used, the user could easily start license server managers on more than one node and obtain additional licenses.*
port	TCP/IP port number to use. The license administrator can edit this value either directly in the license file or using `lmadmin`. In the latter case, the port number specified in `lmadmin` takes precedence over any port set in the license file, either before or after the port is specified using `lmadmin`. Therefore, editing the port number using `lmadmin` is only recommended if no port number is specified in the license file. If a port number is already specified in the license file and the port is subsequently changed in `lmadmin`, during the next restart the vendor daemon will exit with a “port mismatch” error. A valid number is any unused port number in the range 1 to 65535. On UNIX, choose a port >1024, since those <1024 are privileged port numbers. If you specify a port number greater than 65535, the client fails to establish a connection with the license server manager (`lmadmin` or `lmgrd`). If not specified, the license server will automatically use the next available port number in the range 27000 to 27009. Applications, when connecting to a server, try all numbers in the range 27000 to 27009. SERVER lines that define the license servers in a three-server redundancy configuration must include a port number. ***Note ▪*** *Those producers concerned about potential DoS attacks based on knowledge of common license server ports may want to consider specifying a server port outside the range 27000 to 27009.*

Table 9-2 ▪ SERVER Line

SERVER Line Input	Description
PRIMARY_IS_MASTER	For a three-server redundant configuration, the `PRIMARY_IS_MASTER` keyword indicates how master control is transferred between the primary and secondary servers. ● If this keyword is used and the primary server goes down, the secondary server becomes the master and transfers control back to the primary server when the primary server comes back up. ● If not used and the primary server goes down, the secondary server becomes the master and remains the master even when the primary server comes back up. If both primary and secondary servers go down, licenses are no longer served. In no instance does the tertiary server become the master. This parameter is optional and is placed on the first `SERVER` line in the license file. You must be running a version 10.8 or later vendor daemon to use this parameter.
HEARTBEAT_INTERVAL= *seconds*	For a three-server redundant configuration, this keyword indicates how long a server waits to receive a heartbeat from another server in the configuration before shutting itself down. *seconds* is used in the following equation to calculate the timeout: timeout = (3 x *seconds*) + (*seconds* - 1) If not specified, the default value for seconds is 20, equating to a timeout of 79 seconds. Valid values for seconds are `0` through `120`. This parameter is optional and is placed on the first `SERVER` line in the license file. You must be running a version 10.8 or later vendor daemon to use this parameter.

Note ▪ *The SERVER line must apply to all lines in the license file. It is permitted to combine license files from different vendors, but only if the SERVER hostids are identical in all files that are to be combined. A license search path can be used if hostids are not identical, but refer to the same system.*

See Also
License Administration Guide
HOSTID
Using Three-Server Redundancy
Using Vendor-Defined Hostids

VENDOR Line

`VENDOR vendor [vendor_daemon_path] [[OPTIONS=]options_file_path] [[PORT=]port] [[EPORT=]eport]`

The `VENDOR` line specifies the name and location of a vendor daemon, as well as the location of the options file.

Table 9-3 ▪ VENDOR Line Specifications

VENDOR Line Input	Description
vendor	Name of the vendor daemon used to serve at least some features in the file.
vendor_daemon_path	Path to the executable for this daemon. If not specified and `lmadmin` is used as the license server manager, the vendor daemon locations can be set from the Vendor Daemon Configuration tab of the interface. By default, `lmadmin` will search for vendor daemons from the license server root directory. If not specified and `lmgrd` is used as the license server manager, `lmgrd` uses its own `PATH` environment variable, plus the current directory, to find the vendor daemon.
options_file_path	Path to the options file for this vendor daemon. If unspecified, the vendor daemon will look for a file called `vendor.opt` (where *vendor* is the vendor daemon name) in the same directory where the license file is located. If found, this file is used as the options file.
port	Vendor daemon TCP/IP port number. A valid number is any unused port number in the range 0 to 65535. On UNIX, choose a port >1024, since those <1024 are privileged port numbers. If *port* is not specified or if a port value of 0 is specified, an ephemeral port is chosen by the operating system at run-time. Sites with Internet firewalls need to specify the TCP/IP port number the daemon uses. If a TCP/IP port number is specified on the VENDOR line, there may be a delay restarting the vendor daemon.
eport	Vendor daemon TCP/IP port number for the secured communication. A valid number is any unused port number in the range 0 to 65535. On UNIX, choose a port >1024, since those <1024 are privileged port numbers. If *eport* is not specified or if a port value of 0 is specified, an ephemeral port is chosen by the operating system at run-time. Sites with Internet firewalls need to specify the TCP/IP port number the daemon uses. If a TCP/IP port number is specified on the VENDOR line, there may be a delay restarting the vendor daemon.

UNIX Examples

```
VENDOR acmed
VENDOR acmed /etc/acmed
VENDOR acmed /etc/acmed options=/usr/local/licenses/acmed.opts
```

Windows Examples

```
VENDOR acmed C:\Windows\system\acmed.exe
VENDOR acmed C:\Windows\system\acmed.exe \
        OPTIONS=C:\licenses\acmed.opts
```

USE_SERVER Line

```
USE_SERVER
```

USE_SERVER takes no arguments and has no impact on the server. When the FlexEnabled application sees a USE_SERVER line, it ignores everything in the license file except the preceding SERVER lines. In effect, USE_SERVER forces the application to behave as though LM_LICENSE_FILE were set to *port@host* or *@host*. USE_SERVER is recommended because it improves performance when a license server is used.

The advantages to USE_SERVER are that the application's license file:

- Does not need to match the license file that the server uses.
- Requires only SERVER and USE_SERVER lines.

FEATURE and INCREMENT Lines

```
FEATURE|INCREMENT feature vendor feat_version exp_date num_lic [vendor_keywords] \
            [user_keywords] SIGN=sign
```

The FEATURE, INCREMENT, and UPGRADE lines are collectively called feature definition lines. They describe the license model to use with a FlexEnabled application. Each line is composed of five parts:

- FEATURE or INCREMENT—required keyword
- Positional fields—required
- Vendor keywords—optional
- User keywords—optional
- Signature—required

The optional keywords must appear after all positional fields, but can appear in any order. User keywords are not involved in license authentication. This means they can be modified and the license will remain valid. The signature is required to be last. Table 9-4 summarizes the fields and keywords that you can used on a feature definition line.

Table 9-4 ▪ Feature Definition Line Fields and Keywords

Keyword
Positional Fields — in required order
Feature Name
Vendor Daemon Name

Table 9-4 ▪ Feature Definition Line Fields and Keywords

Keyword
Feature Version
Expiration Date
Number of Licenses
Authenticated Vendor Keywords
BORROW
CAPACITY
DUP_GROUP
FLOAT_OK
HOST_BASED
HOSTID
ISSUED
MINIMUM
ONE_TS_OK
OVERDRAFT
START
SUITE_DUP_GROUP
SUPERSEDE
SUPERSEDE_SIGN
TS_OK
TZ
USER_BASED
VENDOR_STRING
VM_PLATFORMS

Table 9-4 ▪ Feature Definition Line Fields and Keywords

Keyword
Authenticated Informational Vendor Keywords
ISSUER
NOTICE
SN
Unauthenticated Keywords
asset_info
dist_info
user_info
vendor_info
sort
Signature
SIGN

An `INCREMENT` line can be used in place of a `FEATURE` line, as well as to incrementally add licenses to a prior `FEATURE` or `INCREMENT` line in the license file. Only the first `FEATURE` line for a given feature is processed by the vendor daemon. If you want to have additional copies of the same feature (for example, to have multiple node-locked, counted features), use multiple `INCREMENT` lines.

To cause multiple `FEATURE` lines for the same feature to be recognized and processed as `INCREMENT` lines, set *ls_use_all_feature_lines* (in `lsvendor.c`) to a non-zero value. Notify your customers if you set `ls_use_all_feature_lines`.

Important ▪ *Altering the vendor daemon impacts all applications that use that version of your vendor daemon. The original behavior of `FEATURE` line becomes unavailable to all applications that use the vendor daemon.*

Character Limitations in Keyword Values

In places where a keyword value in a `FEATURE` or `INCREMENT` line is a string surrounded with double quotes ("…"), the string can contain any characters except the following:

- Single or double quotes (within the surrounding double quotes)
- The backslash character sequence: `\` (space+backslash+space)
- The double-backslash character sequence: `\\` (space+backslash+backslash+space)

License Pools

INCREMENT lines form license groups, called *license pools*, based on the following fields and keywords:

- Feature name
- Version
- BORROW
- DUP_GROUP
- FLOAT_OK
- HOST_BASED
- HOSTID
- PLATFORMS
- TZ
- USER_BASED
- VENDOR_STRING (if configured by the vendor as a pooling component)
- VM_PLATFORMS

If two lines differ by any of these fields, a new license pool is created in the vendor daemon, and this group is counted independently from other license pools with the same feature name. A FEATURE line does not give an additional number of licenses, whereas an INCREMENT line always gives an additional number of licenses.

Consider the following example that demonstrates license pooling:

```
SERVER speedy 08002b32b161
VENDOR demo
INCREMENT f1 demo 2.0 permanent 1 SIGN=2B8F621C172C
INCREMENT f1 demo 2.0 permanent 2 SIGN=2B9F124C142C
```

- INCREMENT—The server adds the licenses for all lines for the same feature name forming a license pool for feature f1. The concurrent usage limit is 3 (1 + 2).
- The first INCREMENT line could be a FEATURE line and the behavior would be the same.

In contrast, this following example shows separate pooling:

```
SERVER speedy 08002b32b161
VENDOR demo
INCREMENT f1 demo 2.0 permanent 1 HOSTID=80029a3d SIGN=7B9F02AC0645
INCREMENT f1 demo 2.0 permanent 2 HOSTID=778da450 SIGN=6BAFD2BC1C3D
```

- One license is available on hostid 80029a3d.
- Two licenses are available on 778da450.
- The server tracks these licenses independently in separate pools because the HOSTID fields differ.
- Because a license pool is not formed, this independent tracking works only with INCREMENT, not FEATURE, because the server recognizes only the first FEATURE line for a given feature name. Subsequent FEATURE lines are ignored.

A single checkout request cannot span multiple license pools. That is, if a checkout is requesting more licenses than are available in a single license pool, the request is denied. Consider the following example:

```
SERVER speedy 08002b32b161
VENDOR demo
INCREMENT f1 demo 1.0 permanent 3 SIGN=2B8F621C172C
INCREMENT f1 demo 2.0 permanent 4 SIGN=2B9F124C142C
```

These lines form two separate license pools, one with three licenses for version 1.0 and one with four licenses for version 2.0. A request for five licenses for version 1.0 is denied because neither pool has five licenses.

The license checkout behavior is slightly different when feature definition lines are configured such that they create multiple license pools, but the `DUP_GROUP` configuration is the same. For example:

```
INCREMENT f1 demo 1.0 31-dec-2020 1 SIGN=...
INCREMENT f1 demo 1.0 31-dec-2020 2 SIGN=...
```

A checkout request searches all license pools for an existing checkout of the feature made using the same checkout request identifier (in this example, it would search for the same display name). If one is found, that license is used to satisfy the request. If not, a new license is consumed.

FEATURE and INCREMENT Example

To illustrate the difference between `FEATURE` and `INCREMENT`, consider these feature definition lines:

```
FEATURE f1 demo 2.0 permanent 5 ....
FEATURE f1 demo 1.0 permanent 4 ....
```

They result in five licenses for version 2.0. Now, consider:

```
INCREMENT f1 demo 2.0 permanent 5 ....
INCREMENT f1 demo 1.0 permanent 4....
```

- These result in four licenses for version 1.0 *and* five licenses for version 2.0 and below being available, giving a total of nine licenses for `f1`.
- `FEATURE` lines must differ in some way. Otherwise, only one is used.

Counted vs. Uncounted

To contrast counted with uncounted licenses, consider the following `FEATURE` line:

```
FEATURE f1 demo 1.0 1-jan-2020 uncounted HOSTID=DEMO \
        SIGN=123456789012
```

This feature has unlimited usage on any hostid, requires no license server and could, therefore, be a complete license file by itself. It also happens to be an expiring license and will not allow use of the feature after 1-jan-2020.

In contrast, because it is counted, the following feature requires a license server with a vendor daemon named demo:

```
FEATURE f1 demo 1.0 permanent 5 HOSTID=INTERNET=195.186.*.* \
        SIGN=123456789012
```

It limits license usage to five users on any host with an Internet IP address matching 195.186.*.* and it never expires. It must be in a license file with `SERVER` and `VENDOR` lines.

See Also

Using Vendor-Defined Hostids

The following in the *C/C++ Function Reference*

- `ls_use_all_feature_lines`
- `ls_compare_vendor_on_increment`
- `ls_compare_vendor_on_upgrade`
- `ls_a_license_case_sensitive`
- `ls_hud_hostid_case_sensitive`

Feature Name

`feature` is the name given to the feature by the software publisher. Feature names may contain 30 characters or less. The supported set of characters for feature names are: lowercase letters (a-z), uppercase letters (A-Z), digits (0-9), and underscore (_).

Note ▪ *Legacy applications may have used additional characters in the feature name, such as period (.), backslash (\ ), and forward slash (/). For backward compatibility of legacy feature names these characters are supported, but are not recommended for new feature names. The use of hyphen (-) in legacy feature names is partially supported, with the following restrictions:*

- *Passing the feature version when returning a borrowed feature is not supported with feature names containing a hyphen. Clients that borrow such features must set the attribute `LM_A_BORROW_RETURN_BY_VERSION=0`.*
- *The `lmborrowl` utility must be used instead of `lmborrow`.*

Letters in the feature name are case insensitive by default. If case sensitivity is desired, see `LM_A_LICENSE_CASE_SENSITIVE` in the *C/C++ Function Reference*.

Vendor Daemon Name

`vendor` is the vendor daemon name from a `VENDOR` line. It identifies the software publisher from whom this feature is licensed. For a counted license, the *`vendor`* value also identifies the vendor daemon that serves the feature.

Feature Version

The `feat_version` is the latest (highest-numbered) version of this `feature` that is supported by this license file. The version is in floating-point format, with a ten-character maximum.

The feature version defines right-to-use entitlement with versions less than or equal to the value specified in the license certificate. This is FlexNet Licensing's notion of the version and is not necessarily the same as the product version. You assign your own meaning to the version number and test for it within your licensing code. Examples include product version, license version, and date-based version.

Product Version

The license version coincides with your product's version. Entitlement is based on the product's version being less than or equal to that specified in the license certificate.

The advantage of this model is that you maintain only a single version number. However, the disadvantage is that with each product release with a change in the version number, you may need to reissue the license certificate to your customers.

License Version

The license version is independent from the product's version. The notion of the license version is encoded inside the product binary. Entitlement is based on the product's internal license version being less than or equal to that specified in the license certificate. The different numbers allow your customers to run a later version of the product using an older license certificate.

This is the most commonly used model because it gives publishers the flexibility to decide whether the license certificates containing older version numbers will continue to work with new product releases.

Date-Based Version

The license version is based on a date. The version of the product binary is encoded using a date-based scheme. Entitlement is based on the product's internal date–based version being less than or equal to the date specified in the license certificate.

Note ▪ *Date-based versioning pertains to the product version and specifies which version of the product is entitled to run. This modifier is distinct from license start and expiration date modifiers, which specify a date range within which the entitled version is licensed and allowed to operate.*

Using Date-Based Versioning with lmadmin

In order to use date-based versioning and enable your end users to set an alert in `lmadmin` based on the date-based version within their license files, you must use the following format for `feat_version`—

`yyyy.mmdd`

—where the date is composed of numeric characters and *`yyyy`* = year, *`mm`* = month, and *`dd`* = day.

Expiration Date

`exp_date` is the expiration date of the feature in the format:

`{dd-mmm-yyyy | permanent}`

For example, `22-mar-2020`. For no expiration, use `permanent`. Use four digits for the year specification. A date with a year of `0` is equivalent to `permanent`: `1-jan-0`, `1-jan-00`, `1-jan-0000`.

Number of Licenses

Number of licenses of this feature that the customer can use. A value greater than `0` denotes that the license is counted. Use `uncounted` or `0`, for unlimited use of node-locked licenses.

BORROW

`BORROW[=n]`

Optional field. Enables license borrowing for a feature. The value, *n*, is the maximum number of hours for which the license can be borrowed. The default value is 168 (hours), which equates to one week. The maximum borrow period is platform dependent. Typically, a value under 60000 hours works on most platforms.

`BORROW` licenses are susceptible to extra uses if, for example, a user stops and restarts the license server while licenses are borrowed. Specifying a lower value for *n* minimizes the possibility of extra license usage. Issue `BORROW` licenses only to trusted customers.

See Also
License Borrowing using the BORROW Keyword

CAPACITY

`CAPACITY`

Optional keyword. The most common use of the `CAPACITY` keyword is to provide capacity licensing. Capacity could be based on a system's resources. This technique effectively charges more for a more powerful system.

The `CAPACITY` keyword enables a checkout multiplier implemented internally in the FlexEnabled application with the `LM_A_CAPACITY` attribute. `LM_A_CAPACITY` is set using **lc_set_attr**, available in the Flexible API. If **lc_checkout** requests two licenses, and `LM_A_CAPACITY` is set to 3, six licenses will be checked out. Capacity licensing is implemented by adding the `CAPACITY` keyword to the `FEATURE` line and setting `LM_A_CAPACITY` in the application with:

```
lc_set_attr(job,LM_A_CAPACITY,(LM_A_VAL_TYPE)i);
```

Both components must be implemented (the `CAPACITY` keyword on the `FEATURE` line and the `LM_A_CAPACITY` attribute set in the application) for capacity licensing to be enabled. If one component is missing, checkout requests proceed without applying a multiplier to the requested license count. The `LM_A_CAPACITY` attribute must be set before the first connection to the server (usually **lc_checkout**) and cannot be reset once set.

For example, you can license your product based wholly or in part on the number of CPUs on the FlexEnabled client. Processors enabled with Intel's Hyper-Threading Technology, first seen in the Intel Xeon processor family, appear as two logical processors to the operating system. Processors that implement Hyper-Threading Technology provide improvements in performance, resource utilization and processing throughput as compared to single processors without Hyper-Threading Technology, but the performance is not equivalent to two physical processors. System calls (such as **GetSystemInfo**) on these systems return the number of logical processors rather than the number of physical processors present in the system. In order to accurately calculate the number of physical CPUs, and thus, determine an appropriate setting for the `LM_A_CAPACITY` attribute, use code provided by Intel located at http://www.intel.com/cd/ids/developer/asmo-na/eng/index.htm

DUP_GROUP

`DUP_GROUP=NONE|SITE|[UHDV]`

Optional field. If `DUP_GROUP=` is specified in the license, this parameter overrides the `dup_group` parameter in the call to **lc_checkout**. If not specified in the license, the `dup_group` parameter from **lc_checkout** will be used. The syntax is:

```
DUP_GROUP=NONE|SITE|[UHDV]
        U = DUP_USER
        H = DUP_HOST
        D = DUP_DISPLAY
        V = DUP_VENDOR_DEF
```

Any combination of UHDV is allowed. For example DUP_GROUP=UHD means the duplicate grouping is (DUP_USER | DUP_HOST | DUP_DISPLAY), so a user on the same host and display will have additional uses of a feature and not consume additional licenses. This keyword is valid only with counted licenses.

FLOAT_OK

FLOAT_OK[=*server_hostid*]

Optional field. Enables mobile licensing using a FLEXID dongle with FLOAT_OK for a particular feature. The feature definition line must also be node-locked to the dongle using a FLEXID as a hostid.

When FLOAT_OK=server_hostid is specified on a FEATURE line:

- The server_hostid must refer to the same host that appears on the SERVER line of the license file.
- The license server can be run only on the system with the hostid (returned by lmhostid) that is the same as the server_hostid specified by the FLOAT_OK keyword.
- A user can run the license server, but can use only the license being served by the license server, not the node-locked license. Otherwise, an extra use for each FLOAT_OK license could occur.
- The hostid on the FEATURE line containing the FLOAT_OK keyword must be only a single hostid. For multiple FlexNet ID dongles, use individual FEATURE lines for each dongle.

See Also
Node-Locked to a FlexNet ID Dongle with FLOAT_OK

HOST_BASED

HOST_BASED[=*n*]

Optional field. If HOST_BASED appears, then licenses can be used only by hosts INCLUDEd for this feature in the options file. The purpose is to limit the use to particular hosts, but allow the user to determine which hosts.

The licenses in feature definition lines that include this option are available only when there is at least one INCLUDE statement in the options file that specifies the feature. Note that any users specified in INCLUDEALL statements in the options file will also have access to any HOST_BASED features. The include list is created from all the INCLUDEALL and INCLUDE lines in the options file.

If =*n* is specified, then the number of hosts that can be INCLUDEd is limited to *n*. Otherwise, the limit is the num_lic field. If an INCREMENT appears where some licenses are HOST_BASED and some are not, the vendor daemon tracks these in separate license pools.

HOSTID

```
HOSTID="hostid1 hostid2 ... hostidn"
```

A hostid binds the feature to a particular host or hosts. It is a case insensitive string returned by the `lmhostid` utility or by the Flexible API function, **lc_hostid**. A hostid is required for uncounted licenses and is optional for counted licenses. Counted licenses are usually bound to the hostid of the system running the license server, in which case, the hostid is specified on the `SERVER` line of the license file or files loaded by the license server.

A hostid list can be specified. Each hostid is space separated; quotes surround the entire list. For example:

```
HOSTID="12345678 FLEXID=9-876d321a HOSTNAME=joe"
```

If a list of hostids is used, the feature is granted on any one of the hostids in the list.

Using Wildcards in Domain Names

When using `HOSTID` in a `FEATURE` or `INCREMENT` line, you can use a wildcard character in the domain name. For example, the following value allows the feature to be checked out by any hostname in the Revenera domain:

```
HOSTID=HOSTNAME=*.revenera.com
```

When wildcards are used in license files in this way, you must alter your licensing code in your FlexEnabled application. Use the Flexible API function **lc_set_attr** to set `LM_A_USE_FQDN` to 1, as shown in the following example code:

```
(void)lc_set_attr(lm_job,LM_A_USE_FQDN,(LM_A_VAL_TYPE)1);
```

This configures the licensing system to use fully qualified domain names. The default, when `LM_A_USE_FQDN` is not specifically set, is to use relative distinguished names. For details, see “Flexible API Attributes” in the *C/C++ Function Reference*.

Hostid List Considerations

For uncounted licenses, the following line—

```
FEATURE f0 ... uncounted HOSTID="hostid1 hostid2 hostid3"
```

—is equivalent to the set of lines:

```
FEATURE f0 ... uncounted HOSTID=hostid1
FEATURE f0 ... uncounted HOSTID=hostid2
FEATURE f0 ... uncounted HOSTID=hostid3
```

In contrast, for counted licenses, consider the following `FEATURE` line that provides one license, node-locked to a hostid list. The one license can be used at any one time on any one of the specified hostids.

```
FEATURE f0 ... 1 HOSTID="hostid1 hostid2 hostid3"
```

However, providing the following three `FEATURE` lines, each with one license node-locked to one hostid, provides three licenses:

```
FEATURE f0 ... 1 HOSTID=hostid1
FEATURE f0 ... 1 HOSTID=hostid2
FEATURE f0 ... 1 HOSTID=hostid3
```

See Also
Hostids for information about platform-specific and special hostids.
SERVER Lines

ISSUED

`ISSUED=dd-mmm-yyyy`

Optional field. Date that the license was issued; can be used in conjunction with `SUPERSEDE`.

ISSUER

`ISSUER="..."`

Optional field. Issuer of the license. One or more consecutive embedded blank characters are converted to a single blank character after the feature definition line has been signed.

MINIMUM

`MINIMUM=n`

Optional field. If in **lc_checkout(**`...num_lic...`**)**, `num_lic` is less than *n*, then the server will check out *n* licenses.

NOTICE

`NOTICE="..."`

Optional field. A field for intellectual property notices.

ONE_TS_OK

`ONE_TS_OK`

Optional field. Allows a user connected through a remote connection to another system (using Remote Desktop Connection or Terminal Services) to run the FlexEnabled application. This keyword is supported with node-locked, uncounted licenses only. To allow the FlexEnabled application to run using the remote connection, you must define the license model as node-locked to the remote system and add the `ONE_TS_OK` keyword to the license. For example:

```
FEATURE f1 xyzd 1.00 1-jan-2020 uncounted HOSTID=ts_hostid \
        ONE_TS_OK SIGN=<...>
FEATURE f2 xyzd 1.00 1-jan-2020 uncounted HOSTID=ts_hostid \
        ONE_TS_OK SIGN=<...>
```

Without `ONE_TS_OK` keyword on a feature definition line, the user is denied access to the FlexEnabled application. This keyword works similarly to `TS_OK` except that it allows only one remote session to check out a feature at a time. It is supported on all Windows operating systems that allow remote desktop connections.

Consider the scenario where a FlexEnabled application is installed on a system with its license locked to system's hostid. There are two features available: f1 and f2.

- The application runs successfully when accessed from that system directly, regardless of whether the feature definition line contains the `ONE_TS_OK` keyword. The license is checked out as though it was a node-locked license.

- If the feature definition line does not contain the `ONE_TS_OK` keyword and a user tries to run the application by connecting to the system through a remote connection, an error is returned.
- If the feature definition line contains the `ONE_TS_OK` keyword and one user runs the application from the system directly while another user runs the application from a remote connection, both users are successful.
- If the feature definition line contains the `ONE_TS_OK` keyword and more than one user tries to run the application through a remote connection, the first user to check out f1 is successful. Subsequent requests for feature f1 return an error. However, another user can access the system through a remote connection and check out feature f2.

Important ▪ *The limitation is on the number of times a feature is checked out. More than one remote session can run the application, but only one session can check out a feature at a time.*

If both `ONE_TS_OK` and `TS_OK` are defined in a feature definition line, the `ONE_TS_OK` behavior takes precedence. This keyword does not limit users accessing the system using the Fast User Switching feature in Windows.

OVERDRAFT

`OVERDRAFT=`*n*

Optional field. The `OVERDRAFT` policy enables you to specify a number of additional licenses that your user is allowed to use, in addition to the licenses they have purchased. This is useful if you want to allow your users to not be denied service when in a temporary overdraft state. Usage above the licensed limit will be reported by the FlexNet Manager reporting tool.

In addition, you can determine if the user is in an overdraft condition by calling **lc_get_attr(***job*`,` `LM_A_VD_FEATURE_INFO`**`,...`)**. The returned structure has three members of interest related to overdraft: `tot_lic_in_use`, `num_lic`, and `overdraft`. An overdraft state is defined as `tot_lic_in_use` > (`num_lic - overdraft`). If `tot_lic_in_use=num_lic`, no more licenses are available; both the normal allotment and the overdraft allotment have been consumed.

PLATFORMS

`PLATFORMS="`*plat1 ... platn*`"`

Optional field. This enables you to restrict usage to specified hardware platforms. The platforms are defined with FlexNet Publisher platform names and are the same as used to license FlexNet Publisher itself: `sun4_u`, `i86_n`, and so on. The FlexNet Publisher *Release Notes* contain the currently supported platforms and their associated platform names.

The platform name can be overridden with the following:

```
lc_set_attr(job, LM_A_PLATFORM_OVERRIDE, (LM_A_VAL_TYPE)str);
```

The trailing digit in the platform name is ignored and can be optionally left off in the name.

If the platform list differs in any way for two `INCREMENT` lines for the same feature name, they are pooled and counted separately.

Examples

```
FEATURE f1 ... PLATFORMS=sun4_u5
INCREMENT f2 ... 1 PLATFORMS="i86_n hp700_u"
INCREMENT f2 ... 1 PLATFORMS="i86_n"
```

Feature f1 can be used on any Sparc station running Solaris.

Feature f2 can be used on a Windows or HP system. There is one license that can be shared between all Windows and HP systems and one license just for Windows. That is, at most one f2 can be used on the HP systems, and at most two f2s can be used on Windows systems.

If the checkout fails because it is on the wrong platform, the error returned is `LM_PLATNOTLIC`: “This platform not authorized by license.”

See Also
`LM_A_PLATFORM_OVERRIDE` in the *C/C++ Function Reference*
Release Notes

SIGN

`SIGN=`*`signature`*

Required field. Signature for this feature definition line. A *signature* is from 12 to 20 characters long and is produced by **lc_cryptstr** in `lmcrypt` or by a publisher-defined utility that calls **lc_crypstr**. When using `lmcrypt`, put `SIGN=0` at the end of each feature definition line, and `lmcrypt` will replace the 0 with the correct signature.

See Also
Cross-Version License Files

SN

`SN=`*`serial_num`*

Optional field. Useful for differentiating otherwise identical `INCREMENT` lines. Its only use by FlexNet Publisher is in the encrypted signature.

START

`START=`*`dd-mmm-yyyy`*

Optional field. Feature start date. If the license is used before this date, the checkout fails with `LM_TOOEARLY`.

SUITE_DUP_GROUP

`SUITE_DUP_GROUP=NONE|SITE|[UHDV]`

Optional field. Similar to `DUP_GROUP`, but affects only the enabling `FEATURE` line for a package suite.

Note ▪ *If `SUITE_DUP_GROUP` is not specified, the parent will have the same duplicate grouping as the components.*

SUITE_DUP_GROUP limits the total number of users of the package to the number of licenses, and allows the package to be shared among the users that have the SUITE checked out. For example

```
PACKAGE p ... COMPONENTS="A B C" OPTIONS=SUITE
FEATURE p ... 3 ... SUITE_DUP_GROUP=UHD
```

In this example, SUITE_DUP_GROUP limits the number of component users to 3, and, separately, limits the number of uses of each component to 3. This keyword is valid only with counted licenses.

See Also
DUP_GROUP
PACKAGE Lines

SUPERSEDE

SUPERSEDE[="*feat1 ... featn*"]

Optional field. Replaces existing lines in a license file. Without the optional list of features, allows publishers to sum up a set of INCREMENT lines in a single, new FEATURE (or INCREMENT) line, which supersedes the following:

- All INCREMENT lines for the same feature name with START or ISSUED dates earlier than the SUPERSEDE ISSUED date
- All INCREMENT lines for this same feature for which no ISSUED dates are specified

With the optional list of features, it replaces all previously issued lines for *feat1* through *featn*.

Specifying the start date with the ISSUED keyword makes this date explicit (ISSUED=1-jan-2018). If the ISSUED date is set, then SUPERSEDE uses it; otherwise it uses the START date.

In the following two examples, the second line supersedes the first, and the first line is ignored:

```
INCREMENT f1 ... 1 ... ISSUED=1-jan-2017
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-2018
```

or

```
INCREMENT f1 ... 1
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-2018
```

In the next example, f3 supersedes f1 and f2 and causes the license file to support only f3:

```
FEATURE f1 ... 1 ... ISSUED=1-jan-2017
FEATURE f2 ... 1 ... ISSUED=1-jan-2017
FEATURE f3 ... 4 ... SUPERSEDE="f1 f2" ISSUED=2-jan-2018
```

Multiple INCREMENT lines for the same feature—each specifying SUPERSEDE with the same ISSUED date—will be pooled together rather than superseding one another. The resulting license pool will collectively supersede the older feature. For example, the following INCREMENT lines result in a license pool containing seven licenses for f1, collectively superseding the first INCREMENT line:

```
INCREMENT f1 ... 1 ... ISSUED=1-jan-2017
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-2018
INCREMENT f1 ... 3 ... SUPERSEDE ISSUED=1-jan-2018
```

Note ▪ *If you use this keyword on the feature definition line, you cannot use the* SUPERSEDE_SIGN *keyword.*

SUPERSEDE_SIGN

```
SUPERSEDE_SIGN={"<feature name>:<sign>" "<feature name>:<sign>" …}
SUPERSEDE_SIGN={"<package_name>:<pkg_sign>" "<package_name>:<pkg_sign>"}
```

Optional field. Allows you to override the license models of all feature definition lines defined by the SUPERSEDE_SIGN keyword. This keyword behaves similarly to the SUPERSEDE keyword except that it requires you to explicitly define which feature definition lines to override.

If this is used in a FEATURE or INCREMENT line, you must specify the feature name and signature value that will be overridden. If this is used in a PACKAGE line, you must specify the package name and package signature value that will be overridden.

If multiple feature definition lines use the SUPERSEDE_SIGN keyword to specify the same feature name and sign value, then the first SUPERSEDE_SIGN instance overrides that feature's license model. The other SUPERSEDE_SIGN instances are ignored. This rule applies when using this keyword in PACKAGE lines also.

Important ▪ *You cannot use both the SUPERSEDE and SUPERSEDE_SIGN keywords on the same feature definition line (or package line). Do not use both the SUPERSEDE keyword and the SUPERSEDE_SIGN to override the license model for the same feature name or package name. This may produce unexpected results.*

In the following example, five feature definition lines are defined. Feature f3 contains the SUPERSEDE_SIGN keyword and overrides the second f1 feature definition line and the second f2 feature definition line.

```
INCREMENT f1 <...> 2 ISSUED=1-jan-2017 SIGN="0002 <...>”
INCREMENT f1 <...> 4 ISSUED=2-jan-2017 SIGN="00C4 <...>"
INCREMENT f2 <...> 4 ISSUED=1-jan-2017 SIGN="0056 <...>"
INCREMENT f2 <...> 4 ISSUED=2-jan-2017 SIGN="00C8 <...>"
INCREMENT f3 <...> 1 SUPERSEDE_SIGN={"f1:00C4 <...>" "f2:00C8 <...>"} ISSUED=15-feb-2017 \
SIGN="0082 7DCF 1473 1555 E614 2990 4219 D100 4328 1F26 C8AA 6E52 A428 CEC0 1CDF"
```

TS_OK

```
TS_OK
```

Optional field. Allows a user running on a Windows Terminal Server Client system to run the FlexEnabled application using a local license. For a node-locked uncounted license, lock the license to the HOSTID of the Terminal Server system.

For example:

```
FEATURE f1 xyzd 1.00 1-jan-2020 uncounted HOSTID=ts_hostid TS_OK SIGN=sign (node-locked)
FEATURE f1 xyzd 1.00 1-jan-2020 uncounted TS_OK SIGN=sign (not node-locked
```

Without the TS_OK keyword on a FEATURE line, the user is denied access to the FlexEnabled application. This keyword allows any number of remote sessions to run the FlexEnabled application on a Terminal Server.

Consider the scenario where a FlexEnabled application is installed on a Terminal Server system with its license locked to Terminal Server system's hostid.

- The application successfully executes directly on the Terminal Server system, regardless of whether the feature definition line contains the TS_OK keyword. The license is checked out as though it was a node-locked license.

- If the feature definition line does not contain the `TS_OK` keyword and a user tries to run the application from a Terminal Server Client window, an error is triggered.
- If the feature definition line contains the `TS_OK` keyword and one or more users try to run the application from a Terminal Server Client window, the license is checked out and the application can run. FlexNet Publisher does not limit the number of Terminal Server Clients that can use this node-locked license.

Counted licenses behave on a Terminal Server Client as they do on any other system on a network.

TZ

```
TZ=[SERVERTZ | <[+-]hh<.30|.45><:[+-]hh<.30|.45>>>]
```

Optional field. Enables you to enforce license usage for a feature relative to a time zone; where the time zone is specified and measured relative to Greenwich Mean Time (GMT). Use `TZ` on a `FEATURE` or `INCREMENT` line to specify that the computer system on which the FlexEnabled application is running must be within a specified time zone or range of time zones, or in the same time zone as the license server (using the value `SERVERTZ`). You can also specify half (.30) and quarter (.45) time zones, if applicable.

Security

The implementation for served licenses uses both the time and time zone information of the FlexEnabled application for time zone validation. The FlexEnabled client sends the time and time zone information to the license server on every checkout request. The client's time and time zone are normalized to GMT, and on success, the client's time zone is validated against the time zone information specified in the license file. If the time zone information matches, the license is granted to the requesting client application. However, during validation of time zone information in the license server, 15 minutes offset time is allowed between FlexEnabled client and the license server.

- Time zone values must be preceded by a + (plus) or – (minus) sign. Every + or – sign must be followed by two decimal digit hours *h*. The only cases in which you would specify values for minutes would be if specifying a half (.30) or quarter (.45) time zone.
- Valid range for the `TZ` keyword is –12 to +13. To specify GMT as the time zone value, use either `TZ=+00` or `TZ=–00`.
- To specify multiple time zones, use a space as a delimiter and enclose the values in quotes. If `TZ` keyword has a single value, quotes are ignored when the license file is encrypted. For example, `TZ="-06 -08"` identifies the GMT-06:00 and GMT-08:00 time zones.
- To specify a time zone range, use a colon between time zone values. The *from* and *to* values for the range must use the same + or – sign. For example, `TZ=-06:-08` identifies a time zone range that includes all the time zones from GMT-06:00 to GMT-08:00.
- If the time zone range includes GMT, you must use two range specifications. For example, `TZ="–05:–00 +00:+01"` specifies the range from GMT–05:00 to GMT+01:00, inclusive.

***The `TZ` keyword is not supported with decimal-format licenses.

- `SERVERTZ` is a case-sensitive value that can be used with the `TZ` keyword to specify that the license server should serve licenses only if the time zone in which the client resides matches the time zone in which the license server resides. `SERVERTZ` is valid only for use with a served license model, and cannot be combined with any other values.

Note ▪ *The time zone value is not validated in the license file, as long as the syntax is correct. For example, specifying `TZ=-07.30` will not trigger an encryption-time error, although GMT-07:30 is an invalid time zone.*

If the time zone value differs in any way between two `INCREMENT` lines for the same feature name, they are pooled and counted separately.

See Also
Time Zone Licensing

USER_BASED

`USER_BASED[=n]`

Optional field. If `USER_BASED` appears, then licenses can be used only by users `INCLUDE`d for this feature in the options file. This limits the use to particular end users, but enables the license administrator to determine which users.

The licenses in feature definition lines that include this option are only available when there is at least one `INCLUDE` statement in the options file that specifies the feature. Any users specified in `INCLUDEALL` statements in the options file will also have access to any `USER_BASED` features. The include list is created from all the `INCLUDEALL` and `INCLUDE` lines in the options file.

If *=n* is specified, then the number of users which can be `INCLUDE`d is limited to *n*. Otherwise, the limit is the *num_lic* field. If an `INCREMENT` appears where some licenses are `USER_BASED` and some are not, the vendor daemon tracks these in separate license pools.

VENDOR_STRING

`VENDOR_STRING="..."`

Optional field. Publisher-defined license data. If a checkout is conditional on the contents of the vendor string, then `LM_A_CHECKOUTFILTER` is the recommended way to do this. If the `VENDOR_STRING` is set, you will probably also need to set `ls_compare_vendor_*` in `lsvendor.c`.

The `VENDOR_STRING` is optionally configured as a pooling component, in which case, if the string differs in any way for two `INCREMENT` lines for the same feature name, they are pooled and counted separately.

See Also
The following in the *C/C++ Function Reference*:

```
lc_auth_data
LM_A_CHECKOUTFILTER
LM_A_CHECKOUTFILTER_EX
ls_compare_vendor_on_increment
ls_compare_vendor_on_upgrade
```

VM_PLATFORMS

`VM_PLATFORMS=PHYSICAL | VM_ONLY | VM_ALL`

Optional field. Enables you to specify whether feature usage should be restricted to virtual machines or restricted to physical machines.

If the `VM_PLATFORMS` value differs in any way between two `INCREMENT` lines for the same feature name, they are pooled and counted separately.

Important ▪ `VM_PLATFORMS` *is not supported in the Java kit.*

The following keyword values are supported. Specify one and only one value when using the `VM_PLATFORMS` keyword.

Table 9-5 ▪ VM_PLATFORMS Values

Value	Description
PHYSICAL	Restricts the application to run only on a physical machine; application cannot be run in a virtual environment.
VM_ONLY	Restricts the application to run only in a virtual environment; application cannot be run on a physical machine.
VM_ALL	Allows running the FlexEnabled application on both physical and virtual machines (default behavior). This is equivalent to not specifying the `VM_PLATFORMS` keyword.

See Also
Virtualization

asset_info

```
asset_info="..."
```

Optional field. Additional information provided by the license administrator for asset management. Not encrypted into the feature's signature.

dist_info

```
dist_info="..."
```

Optional field. Additional information provided by the software distributor. Not encrypted into the feature's signature.

sort

`sort=nnn`

Optional field. This attribute is used alone or with the `LM_A_SORT_TS_FIRST` attribute to override the default sorting order of feature definition lines. The value *nnn* is an integer that specifies the relative sort order. The default value for the `sort` attribute is `100`. Lines with a sort order value of less than `100` are sorted before all lines without this attribute, and lines with a sort order value greater than `100` appear after all unmarked lines. All lines with the same number are sorted as they appear in the file.

To turn off automatic ordering:

- Set `LM_A_SORT_TS_FIRST` to zero (`0`) using the **lc_set_attr** function.
- Add `sort=nnn`, where *nnn* is the same on all lines.

Automatic ordering does not affect the order of features returned by **lc_feat_list.**

For more information about configuring the license sort capabilities, see Sorting Licenses During Initialization.

See Also
Sorting Licenses During Initialization
`LM_A_SORT_TS_FIRST` in the *C/C++ Function Reference*

user_info

`user_info="..."`

Optional field. Additional information provided by the license administrator. Not encrypted into the feature's signature.

vendor_info

`vendor_info="..."`

Optional field. Additional information provided by the software publisher. Not encrypted into the feature's signature.

UPGRADE Lines

```
UPGRADE feature vendor from_feat_version to_feat_version \
        exp_date num_lic [options ... ] SIGN=sign
```

The `FEATURE`, `INCREMENT`, and `UPGRADE` lines are collectively called feature definition lines. The structure of an `UPGRADE` line is that same as a `FEATURE` or `INCREMENT` line, with the addition of the *from_feat_version* field. An `UPGRADE` line removes up to the number of licenses specified by *num_lic* from any old version (>= *from_feat_version*) and creates a new version with that same number of licenses.

`UPGRADE` operates on preceding `FEATURE` or `INCREMENT` lines, starting with the first one with the lowest version, whose version number is >= *from_feat_version*, and < *to_feat_version*.

For example, the two lines—

```
INCREMENT f1 demo 1.0 1-jan-2020 5 SIGN=9BFAC03164ED
UPGRADE f1 demo 1.0 2.0 1-jan-2020 2 SIGN=1B9A30316207
```

—result in 3 licenses of version 1.0 of f1 and 2 licenses of version 2.0 of f1.

And, the three lines—

```
INCREMENT f1 demo 1.0 1-jan-2020 5 SIGN=9BFAC03164ED
INCREMENT f1 demo 2.0 1-jan-2020 4 SIGN=8BF3fb031643
UPGRADE f1 demo 1.0 3.0 1-jan-2020 2 SIGN=1B9A30316207
```

—result in 3 licenses for version 1.0, the original 4 licenses for version 2.0, and 2 licenses for version 3.0 (taken from the original group of five version 1.0 licenses).

Consider this scenario:

```
INCREMENT f1 demo 1.0 1-jan-2020 5 SIGN=9BFAC03164ED
INCREMENT f1 demo 2.0 1-jan-2020 4 SIGN=8BF3fb031643
UPGRADE f1 demo 1.0 3.0 1-jan-2020 10 SIGN=afb303162056
```

This results in 9 licenses for version 3.0 (5 version 1.0 plus 4 version 2.0 all upgraded to version 3.0) and 1 unused upgrade.

License Pool Considerations

Multiple `UPGRADE` lines applied to a set of `FEATURE` and `INCREMENT` lines can have the result of creating separate license pools where they did not previously exist. Subsequent checkout requests are subject to multiple license pool restrictions. For example:

```
INCREMENT f1 demo 1.0 1-jan-2020 8 SIGN=9BFAC03164ED
UPGRADE f1 demo 1.0 2.0 1-jan-2020 5 SIGN=afb303162056
UPGRADE f1 demo 1.0 3.0 1-jan-2020 3 SIGN=afb303162056
```

This upgrade scenario results in breaking up the one license pool of 8 licenses for version 1.0 into two license pools: one with 5 licenses for version 2.0 and one with 3 licenses for version 3.0. A checkout request of 6 licenses for version 1.0 is denied because neither pool has 6 licenses; whereas, before the upgrade it would be granted because of the single pool of 8 licenses for version 1.0.

PACKAGE Lines

```
PACKAGE package vendor [pkg_version] COMPONENTS=pkg_list \
        [OPTIONS=SUITE|SUITE_RESERVED] \
        [SUPERSEDE[="p1 p2 ..."]ISSUED=date]
        SUPERSEDE_SIGN={"pkg1:pkg_sign" "pkg2:pkg_sign"} SIGN=pkg_sign
```

Table 9-6 ▪ PACKAGE Lines

PACKAGE Line Input	Description
package	Name of the package. The corresponding feature definition line must have the same name.
vendor	Name of the vendor daemon that supports this package (*VENDOR_NAME* in `lm_code.h`).

Table 9-6 ▪ PACKAGE Lines

PACKAGE Line Input	Description
pkg_version	Provide the version of the package. The corresponding feature definition line must have the same version.
pkg_sign	Signature generated by `lmcrypt` or the vendor's customized license generator.
pkg_list	A space-delimited list of components. The format of each component is: *feature*[:*version*[:*num_lic*]] The package must consist of at least one component. *version* and *num_lic* are optional, and if omitted, their values come from the corresponding feature definition line. *num_lic* is only legal if `OPTIONS=SUITE` is not set. Examples: ● `COMPONENTS="comp1 comp2 comp3 comp4"` ● `COMPONENTS="apple:1.5 orange pear:2.0:4"`
OPTIONS=SUITE	With `OPTIONS=SUITE`, the package `FEATURE` is checked out in addition to the component feature being checked out. See PACKAGE Line Example with the OPTIONS=SUITE Keyword for more details.
OPTIONS= SUITE_RESERVED	Reserves a set of package components. Once one package component is checked out, all the other components are reserved for that same user. See PACKAGE Line Example with OPTIONS=SUITE_RESERVED Keyword for more details.
SUPERSEDE [="*p1 p2* **..."]**	Optional field, but if used, use with `ISSUED` date. Replaces all `PACKAGE` lines for the same package name with previous or no `ISSUED` dates. ***Note ▪*** *Do not use the* `SUPERSEDE_SIGN` *and* `SUPERSEDE` *keywords in the same package line.*
SUPERSEDE_SIGN= {"pkg1:pkg_sign" "pkg2:pkg_sign"}	Optional field. Allows you to override the license models of another package line. ***Note ▪*** *Do not use the* `SUPERSEDE_SIGN` *and* `SUPERSEDE` *keywords in the same package line.*
ISSUED= *dd-mmm-yyyy*	Optional field. If `ISSUED` is used with `SUPERSEDE`, the `PACKAGE` line containing `SUPERSEDE` replaces all `PACKAGE` lines for the same package name with previous or no `ISSUED` dates.

The purpose of the `PACKAGE` line is to support different needs:

- To provide a way of distributing one line for a license file that has a large number of features, which largely share the same license model and keywords

- To license a product suite

A PACKAGE line, by itself, does not license anything. A PACKAGE line groups multiple components together and requires a corresponding feature definition line to license the entire package. The components listed are a representation of features available to be checked out.

A PACKAGE line can be shipped with an application, independent of any licenses. Later, you can issue one or more corresponding feature definition lines that will enable the package. PACKAGE lines can be kept in a separate license file. The path to the license file containing the package line should be specified in the application to support this transparently, using LM_A_LICENSE_DEFAULT.

Using the OPTIONS keyword in a PACKAGE line

You can alter the behavior of the PACKAGE line using the OPTIONS keyword. You can assign one of the following values to the OPTIONS keyword:

- OPTIONS keyword is not defined
- OPTIONS = SUITE_RESERVED
- OPTIONS = SUITE

PACKAGE Line Example without the OPTIONS Keyword

The lines in the following example define a package containing seven components. In this example, the OPTIONS keyword is not defined. The license behavior is the same as defining individual feature lines. The benefit is that you minimize the number of lines in the license file. Rather than creating seven separate FEATURE lines, you create only the single PACKAGE line. This example will be used in the following sections to describe the different license behaviors.

```
PACKAGE pkg demo 1.0 COMPONENTS="c1 c2 c3 c4 c5 c6 c7" \
        SIGN=504091605DCF
FEATURE pkg demo 1.0 permanent uncounted HOSTID=778da450\
        SIGN=DB5CC00101A7
```

For the PACKAGE and FEATURE lines in the example, note the following:

- The FEATURE line enables the PACKAGE line.
- Each component inherits information from the enabling FEATURE line. In this example, they all inherit the expiration date, number of licenses, and hostid.
- The enabling FEATURE line must match the name, version, and vendor name of the PACKAGE line.
- The PACKAGE line can be shipped with the product.
- PACKAGE lines can be shipped in a separate file that does not need to be edited by the user as long as the file is included in the license search path.
- These PACKAGE and FEATURE lines, together, are a more efficient way of delivering the following seven FEATURE lines:

  ```
  FEATURE c1 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=D03F02432106
  FEATURE c2 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=99375F40FD85
  FEATURE c3 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=68FAC130DB90
  FEATURE c4 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=D3D617E2075A
  FEATURE c5 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=5A91D6EFB68C
  ```

```
FEATURE c6 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=8F75798EB975
FEATURE c7 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=790545E90575
```

PACKAGE Line Example with OPTIONS=SUITE_RESERVED Keyword

In this example, the OPTIONS keyword is defined as SUITE_RESERVED. This means that when a user checks out one of the components (features) defined in the COMPONENTS property, the license server automatically reserves the other features in the package for that user to access later. The license server records that only one license is being used by that user. All components in the package remain reserved for that user until all features are checked back in.

The SUITE_DUP_GROUP keyword must be specified on the FEATURE line, in conjunction with the OPTIONS=SUITE_RESERVED PACKAGE-line keyword, in order configure the reserved package component behavior.

```
PACKAGE office demo 1.0 COMPONENTS="write paint draw" \
        OPTIONS=SUITE_RESERVED SIGN=00504091605D
FEATURE office demo 1.0 permanent 2 \
        DUP_GROUP=NONE SUITE_DUP_GROUP=U SIGN=DB5CC00101A7
```

This license file defines a package suite named office with three components: write, paint, and draw. Note that the suite duplicate grouping criterion, SUITE_DUP_GROUP=U, is defined in this example to the user level (the feature duplicate grouping criterion, DUP_GROUP=NONE, is the default and is supplied in this example for clarity). At most, two distinct users can be granted licenses at any one time, and when one component is checked out, the other components in the package are reserved for that user. This concept is illustrated in the following table with two users.

Table 9-7 ▪ PACKAGE_SUITE_RESERVED: Example 1

License Checkout Attempt	User	Requested Feature	Result	Explanation
#1	u1	write	granted	User u1 checks out one component, thereby reserving one package license.
#2	u2	paint	granted	User u2 checks out one component, thereby reserving the second package license.
#3	u1	write	denied	Maximum number of package licenses exceeded
#4	u3	paint	denied	Maximum number of package licenses exceeded.
#5	u1	paint	granted	A paint license is available for user u1.
#6	u1	paint	denied	The second paint license is checked out by user u2.
#7	u2	write	granted	User u2 rightfully gets the second write license.

Table 9-7 ▪ PACKAGE_SUITE_RESERVED: Example 1

License Checkout Attempt	User	Requested Feature	Result	Explanation
#8	u1	draw	granted	A `draw` license is available for each user.
#9	u2	draw	granted	
#10	u1 or u2	draw	denied	Maximum number of `draw` licenses exceeded.

Consider this variation where one user obtains both package licenses:

Table 9-8 ▪ PACKAGE_SUITE_RESERVED: Example 2

License Checkout Attempt	User	Requested Feature	Result	Explanation
#1	u1	write	granted	User u1 checks out one component, thereby reserving one package license.
#2	u1	write	granted	User u1 checks out another component, thereby reserving the second package license.
#3	u2	write, paint, or draw	denied	Maximum number of package licenses exceeded
#4	u1	draw	granted	Both `draw` license are reserved for user u1.
#5	u1	draw	granted	

Note that both available sets of package components are reserved for user u1 by virtue of that user initially being granted two identical component licenses. In the previous example, user u1 gets both of the grants for `write`, thereby reserving the rest of the components from both available packages for him. This paradigm shuts out user u2 from any of the licenses in the package.

Using the SUITE_RESERVED Option with Duplicate Grouping

When `OPTIONS=SUITE_RESERVED` is specified in the `PACKAGE` line, set `DUP_GROUP=NONE` in the component `FEATURE` lines. Specifying `DUP_GROUP` with any other setting masks the "reserved" feature and the behavior is as if `OPTIONS=SUITE` had been specified in the `PACKAGE` line. The only difference between `OPTIONS=SUITE` and `OPTIONS=SUITE_RESERVED` is when `DUP_GROUP` is set to `NONE`.

Consider the following license file with `DUP_GROUP` set:

```
PACKAGE office demo 1.0 COMPONENTS="write paint draw" \
        OPTIONS=SUITE_RESERVED SIGN=00504091605D
FEATURE office demo 1.0 permanent 2 DUP_GROUP=U SUITE_DUP_GROUP=U
        SIGN=DB5CC00101A7
```

This variation has the same behavior as in A Fixed Number of Users with Unlimited Component Usage below.

See Also
`LM_A_LICENSE_DEFAULT` in the *C/C++ Function Reference*

PACKAGE Line Example with the OPTIONS=SUITE Keyword

In this configuration, the `OPTIONS` keyword is defined as `SUITE`. This means that the total number of licenses that can be checked out is fixed, regardless of the feature that is checked out.

A package suite provides more flexibility in the way floating licenses are used than just granting a license that applies to one feature. Within a package suite, a floating license is shared among the components in the package rather than being applied to one specific feature.

Various license sharing scenarios are created within this paradigm using the `SUITE_DUP_GROUP` and the `DUP_GROUP` keywords. The examples given below use the U (user) duplicate grouping criterion. Other criteria—H (host), D (display), and V (vendor defined)—can be used to provide a finer granularity for the duplicate grouping specification. More information regarding duplicate grouping is found in *DUP_GROUP* and *SUITE_DUP_GROUP*.

A Fixed Number of Users with Unlimited Component Usage

In this variation, licenses are granted to a fixed number of users at any one time. Each user is granted an unlimited number of licenses for each suite component.

```
PACKAGE office demo 1.0 COMPONENTS="write paint draw" \
        OPTIONS=SUITE SIGN=00504091605D
FEATURE office demo 1.0 permanent 2 \
        DUP_GROUP=U SUITE_DUP_GROUP=U SIGN=DB5CC00101A7
```

This license file defines a package suite named `office` with three components: `write`, `paint`, and `draw`. Additionally, the suite and feature duplicate grouping criteria, `SUITE_DUP_GROUP=U` and `DUP_GROUP=U`, are defined to the user level. At most, two distinct users at any one time can be granted unlimited licenses. This concept is illustrated in the following table.

Table 9-9 ▪ A Fixed Number of Users with Unlimited Component Usage: Example

License Checkout Attempt	User	Requested Feature	Result	Explanation
#1	u1	`write`, `paint`, or `draw`	unlimited grant	Unlimited licenses available for at most two distinct users.
#2	u2	`write`, `paint`, or `draw`	unlimited grant	
#3	u3	`write`, `paint`, or `draw`	denied	Maximum number of distinct users exceeded.

A Fixed Number of Users Sharing Components

In this variation, licenses are granted to a fixed number of users at any one time. Independently, a fixed number of licenses are granted for each suite component at any one time.

```
PACKAGE office demo 1.0 COMPONENTS="write paint draw" \
        OPTIONS=SUITE SIGN=00504091605D
FEATURE office demo 1.0 permanent 2 \
        DUP_GROUP=NONE SUITE_DUP_GROUP=U SIGN=DB5CC00101A7
```

This license file defines a package suite named `office` with three components: `write`, `paint`, and `draw`. Additionally, the suite duplicate grouping criterion, `SUITE_DUP_GROUP=U`, is defined to the user level (the feature duplicate grouping criterion, `DUP_GROUP=NONE`, is the default and is supplied in this example for clarity). At most, two distinct users can be granted licenses at any one time and, at most, two licenses can be granted for each suite component at any one time. When the licenses are consumed, no further licenses are available for checkout. This concept is illustrated in the following table.

Table 9-10 ▪ A Fixed Number of Users Sharing Components: Example

License Checkout Attempt	User	Requested Feature	Result	Explanation
#1	u1 or u2	write	granted	Two `write` licenses available for at most two distinct users.
#2	u1 or u2	write	granted	
#3	u1 or u2	write	denied	Maximum number of `write` licenses exceeded
#4	u3	paint	denied	Maximum number of distinct users exceeded.
#5	u1 or u2	paint	granted	Two `paint` licenses available for at most two distinct users.
#6	u1 or u2	paint	granted	
#7	u1 or u2	paint	denied	Maximum number of `paint` licenses exceeded.
#8	u3	draw	denied	Maximum number of distinct users exceeded.
#9	u1 or u2	draw	granted	Two `draw` licenses available for at most two distinct users.
#10	u1 or u2	draw	granted	
#11	u1 or u2	draw	denied	Maximum number of `draw` licenses exceeded.

Note that components are not reserved for a given user by virtue of that user being granted one component. In the example depicted above, user u1 getting both of the grants for `paint` does not reserve the rest of the components for him. This paradigm allows user u2 to get the remaining four grants: two for `write` and two for `draw`.

Sharing a Floating License

Without feature or suite duplicate grouping constraints specified in the components of package suite, a floating license is shared among the components of the package regardless of the number of users. Consider the following example:

```
PACKAGE office demo 1.0 COMPONENTS="write paint draw" \
        OPTIONS=SUITE SIGN=00504091605D
```

```
FEATURE office demo 1.0 permanent 1 SIGN=DB5CC00101A7
```

This license file defines a package suite named `office`—using `OPTIONS=SUITE` on the `PACKAGE` line—with three components: `write`, `paint`, and `draw`. One license is available and shared among the three components. When the license is consumed for one of the components, no further licenses are available for checkout. This concept is illustrated in the following table.

Table 9-11 ▪ Sharing a floating license: Example

License Checkout Attempt	User	Requested Feature	Result	Explanation
#1	any	`write`, `paint`, or `draw`	granted	One license is available for any one of the components.
#2	any	`write`, `paint`, or `draw`	denied	Maximum number of licenses exceeded

10

License Models

This section explains the following license models:

- Demo Licensing
- Lenient Licensing: Report Log and OVERDRAFT

Demo Licensing

There are many popular methods of handling demo licensing—this section discusses the most popular. However, many companies have unique needs, which may not be covered in this section. Contact your Revenera salesperson for a description of the additional types of license models that FlexNet Publisher supports.

Limited Time, Uncounted Demos

This is the most popular method of providing product evaluations. Advantages include:

- No special coding is required in the application
- No license server is required
- License installation is easy
- License files are easy to distribute, since no end user information is required.

The license file should look similar to the following:

```
FEATURE f1 corp 1.0 1-jan-2020 uncounted HOSTID=DEMO \
    SIGN=AB0CC0C16807
```

This indicates the expiration date and the fact that it is a demo license (node-locked to `HOSTID=DEMO`). The product is fully usable until January 1, 2020. `FEATURE` lines like this can be preprinted with different expiration dates, and given to salespeople and distributors. For example, you may distribute the following file (the examples assume a vendor daemon named corp to avoid confusion):

```
FEATURE f1 corp 1.0 1-jan-2020 uncounted HOSTID=DEMO SIGN=AB1CC0916A06
FEATURE f1 corp 1.0 1-feb-2020 uncounted HOSTID=DEMO SIGN=ABDCC0116A06
FEATURE f1 corp 1.0 1-mar-2020 uncounted HOSTID=DEMO SIGN=BBDCA0D151ED
FEATURE f1 corp 1.0 1-apr-2020 uncounted HOSTID=DEMO SIGN=BBDCB0E155F1
[...]
```

If the current date is February 1, 2020, then the salesperson would give an evaluator the third line, which expires in a month, March 1, 2020. The evaluator could simply save the `FEATURE` line in the license file where the product expects it, and then the product will run for one month.

A `PACKAGE` line can be used to make this even easier for multiple features. If a company ships features A through F, the company can initialize the licenses with:

```
PACKAGE all corp 1.0 COMPONENTS="A B C D E F" SIGN=B0A0F011B491
```

Then appending a single demo `FEATURE` line can enable all these features:

```
FEATURE all corp 1.0 1-mar-2020 uncounted HOSTID=DEMO SIGN=AB1CC0916A06
```

The `FEATURE` line must appear after the `PACKAGE` line to work correctly.

Minimizing Fraudulent Use of Expiring Versions

FlexNet Publisher does some security checks to prevent users from setting system dates back. Though date-setback detection can be circumvented, most honest users (who would pay for licenses that cannot be stolen) find that working with incorrect system dates is annoying and too public a form of theft. For companies that are more concerned with security, there are several things that can be done to make date setback less feasible.

Prominently Display Expiration Date

After a successful checkout, call—

```
conf = lc_auth_data(job, feature)
```

—to get an authenticated copy of the `CONFIG` structure that authorized the checkout. Put the expiration date (`CONFIG`->date) in a prominent place in the interface so that the date-setback detection is more public.

Provide an Insistent Reminder

If it is an expiring evaluation version, periodically do something repetitively—perhaps a pop-up window that appears every few minutes that encourages the user to purchase the product.

Limited Functionality Demos

Another way to configure evaluation software is to provide a version that has limited functionality. The user can test the software, but has an incentive to purchase the full version in order to access the application's complete utility.

An example of a limited-functionality version is a word processing program that alters saved files so that, when printed, the word *EVALUATION* is printed in large letters across every page. This allows evaluators full functionality, without reasonable utility.

The application needs to detect that the `HOSTID` is `DEMO` for this type of evaluation, and **lc_auth_data** is the correct function to use:

```
CONFIG *conf;              /* outline of C source */
LM_HANDLE *job;

lc_new_job(...&job);
rc = lc_checkout(job, feature ... );
if (rc) return rc; /* error handling */
conf = lc_auth_data(job, feature);
if (conf->idptr && conf->idptr->type == HOSTID_DEMO)
{
    /* it's a demo license, disable some functionality... */
}
```

See Also

FEATURE and INCREMENT Lines
PACKAGE Lines
`lc_aut_data` in the *C/C++ Function Reference*
ls_allow_vm

Lenient Licensing: Report Log and OVERDRAFT

More and more companies prefer licensing that does not deny usage, but bills users for their usage.

Report Log File

A report log file (which is enabled using either the `lmswitchr` utility or the options file) provides a relatively secure method of tracking license usage. See the *License Administration Guide* for more information about starting and managing a report log file.

Advantages

The advantages of this system include:

- The user is not denied usage during peak usage periods (within limits).
- The vendor can gain additional revenue over traditional floating usage schemes.

A user can limit costs resulting from overdraft usage by including a `MAX_OVERDRAFT` line in the options file.

Limitations

The report log file, while ASCII (so it can be easily emailed), is not human-readable. In addition, any modifications to the file are detected by FlexNet Manager. However, this does not mean that no tampering is possible. There are three conditions that must be considered:

- The user may simply lose a file (either by accident or on purpose). Files are "ended" when a license server stops and starts or when licenses are re-read using the `lmreread` utility. These sections can be lost without detecting a file modification, although the fact that a time period is missing *can* be detected.
- A policy is needed for missing reporting periods. One example policy is "More than *x* hours per month of missing license usage entries terminates the licensing contract."
- A similar policy will be needed for files that have been altered.

OVERDRAFT Detection

Applications may want to inform users when they are in an overdraft state. This can be done with **lc_auth_data** and **lc_get_attr(**... LM_A_VD_FEATURE_INFO...**)**. **lc_auth_data** gives the CONFIG struct for the license that has been used for the call to **lc_checkout**, and LM_A_VD_FEATURE_INFO returns that actual overdraft state in the server.

```
CONFIG *conf;              /* outline of C source */
LM_HANDLE *job;
LM_VD_FEATURE_INFO fi;

if (rc = lc_new_job(...&job)) return rc; /* error */
if (rc = lc_checkout(job, feature ... )) return rc; /* error */
if (!(fi.conf = = lc_auth_data(job, feature))) /* report error */;
else
{
    if (rc = lc_get_attr(job, LM_A_VD_FEATURE_INFO, (short *)&fi))
        /* report this error */;
    else if (fc.lic_in_use > fi.lic_avail - fi.overdraft)
        printf("%s Number of overdraft uses: %d\n", feature,
            fi.lic_in_use - (fi.lic_avail - fi.overdraft));
}
```

See Also

OVERDRAFT

LM_A_VD_GENERIC_INFOR and LM_A_VD_FEATURE_INFO in the *C/C++ Function Reference*

11

License Signature Configurations

This chapter provides background information on the default license signature mechanism, as well as information on supported license signature configurations. This chapter should be read by publishers who want to migrate from an older signature configuration to a newer configuration.

Signature Types

FlexNet Publisher currently supports the following signature types:

- **Non-TRL signature**—Uses a proprietary, non-public-key digital signature method. Non-TRL signatures are easier to crack and it is more likely that a license file might some day be counterfeited. This license type has been deprecated but is still supported to maintain backward compatibility.
- **TRL signature**—Uses a standard public-key system, which reduces the risk of counterfeiting. There are two types of TRL signatures:
  - TRL1—Used by applications built using version 8.0 or earlier toolkits.
  - TRL2—Improved TRL signature; used by applications built using version 8.1 or later toolkits. This is the default signature type.

  For detailed information about TRL, see Introduction to TRL.

Signature Configurations

The signature configurations supported by FlexNet Publisher have evolved over time. The following table describes the different license signature configurations historically supported by FlexNet Publisher. Use this table to determine the required parameters when upgrading to a newer configuration.

Table 11-1 ▪ Signature configurations

Signature Configuration	Config 1	Config 2	Config 3	Config 4	Config 5 (Best Practice)	Config 6 (2nd Best Practice)
ENCRYPTION_SEEDS	Not set	Set	Set	Not set	Not set	Set
LM_SEEDS	Set	Not set	Set	Set	Set	Set
TRL_KEYS	Not set	Set	Set	Set	Set	Set
LM_STRENGTH	DEFAULT	113/163/239 BIT	113/163/239 BIT	113/163/239 BIT	113/163/239 BIT	113/163/239 BIT
Signature keyword	`SIGN`	`SIGN`	`SIGN` and `SIGN2`	`SIGN`	`SIGN`	`SIGN` and `SIGN2`
Signature type	Bespoke symmetric	TRL1	SIGN: TRL1 SIGN2: TRL2	TRL2	TRL2	SIGN: TRL1 SIGN2: TRL2
Signature length (characters)	12	60/84/120	2 x 60/84/120	60/84/120	60/84/120	2 x 60/84/120
Client version	8.1+	7.1 - 8.1	8.1+	8.1+	10.8.6+	10.8.6+
Library linkage	`lmgr.lib`	`lmgr.lib`	`lmgr.lib`	`lmgr.lib`	`lmgr_trl.lib`	`lmgr_trl.lib`

Configuration 5 is considered best practice; configuration 6 is considered second best practice. Non-TRL licenses have been deprecated, but are supported for compatibility reasons. Configurations 1 and 2 are no longer supported. Configurations 3 and 4 are not supported for new clients, but would be needed when upgrading license servers which need to support clients in production that are older than 10.8.6.

Configuration Elements

The elements that make up a signature configuration depend on the specific configuration. This section briefly describes each configuration element mentioned in Table 11-1, Signature configurations.

ENCRYPTION_SEEDS

ENCRYPTION_SEEDs are located in `machind/lm_code.h`. As a general rule, when upgrading to a newer version of FlexNet Publisher, copy the ENCRYPTION_SEEDs from the previous `lm_code.h` to the new `lm_code.h` (unless they are no longer required, as indicated in Table 11-1).

LM_SEEDS

LM_SEEDs are random 32-bit numbers that are used to generate unique, publisher-specific encryption keys. `LM_SEED` definitions are located in `machind/lm_code.h`.

TRL_KEYS

You received TRL_KEY values from Revenera when you purchased the toolkit. `TRL_KEY` values must be set in `machind/lm_code.h`. Attempting to use TRL signatures without TRL keys results in a compile-time error.

LM_STRENGTH

`LM_STRENGTH` is a setting in `machind/lm_code.h` that defines the default signature length. The default setting is `LM_STRENGTH_113BIT`, which will result in a `SIGN=` keyword with 60 characters (113 bit).

Signature Keyword

The signature keyword in the license file holds the signature that is used to authenticate `FEATURE` or `INCREMENT` lines. Depending on your signature configuration, the keyword must be either `SIGN` or `SIGN` and `SIGN2`.

In particular, as per Table 11-1, configuration 3 clients require licenses with the `SIGN2` keyword. Changing a served license file from configuration 3 to configuration 4 or 5 would break these existing clients. Therefore, in order to move on from `SIGN2`, producers needs to retire or upgrade their `SIGN2`-based production client base.

When producers update their clients, they should prefer configuration 5, which is the most secure and simplest license signature configuration. This will enable (eventual) retiring of `SIGN2`. If `SIGN2` is still required, publishers should at least move to configuration 5.

For more information about the `SIGN` and `SIGN2` keywords, see the Knowledge Base article Q111634.

Signature Type

The signature type is either symmetric (proprietary, non-public-key digital signature) or TRL (standard public-key system).

Signature Length

The signature length is defined by the LM_STRENGTH value.

Client Version

The client version denotes the earliest FlexNet Publisher version that was used to build the FlexEnabled application running on a client.

Library Linkage

The standard `lmgr.lib` allows verification of all signature types, whereas the recommended `lmgr_trl.lib` allows verification only of TRL signatures. Revenera strongly recommends configuration 5 (or 6), where clients can verify only a TRL signature, to minimize the risk of counterfeiting.

Choosing the Appropriate Configuration

If your oldest client is older than version 10.8.6, Revenera recommends that you choose at least configuration 5 (best practice) or configuration 6 (second best practice).

Configuration 6 is for publishers who still have `SIGN2` clients in production.

Introduction to TRL

TRL is used to reduce the possibility of counterfeit licenses. Counterfeiting is the most serious threat to application security. TRL offers a standard public-key system that is recognized by the security community and recommended for US government work (with US government export approval). The system comes from Certicom (see www.certicom.com) and is based upon elliptical curve cryptography (ECC), which is considered secure, fast, and efficient.

There are three different signature lengths offered. The longer the signature, the higher degree of security. The lengths of the `SIGN=` keyword in the license are:

- 60 chars (113 bits ECC)
- 84 chars (163 bits ECC)
- 120 chars (239 bits ECC)

The strength is defined by setting `LM_STRENGTH` in `machind/lm_code.h` to the appropriate value. As shipped, the FlexNet Publisher Licensing Toolkit has TRL enabled at the 113-bit strength, denoting a public-key length of 113 bits.

In this documentation, the term “TRL” generally refers to TRL2 licenses, unless otherwise noted.

Public-Key Technology

Public-key is based on mathematics, not “hiding” (obfuscation), which is what is used without TRL.

There are two tasks to be accomplished for the `SIGN=` keyword:

- **Generate digital signature**—`lmcrypt`
- **Authenticate digital signature**—application and vendor daemon

Without public-key, there is one key, which is the same key in `lmcrypt` that is hidden in the application and vendor daemon.

With public-key, there are two different keys: private and public. The private key, used only in the license generators (`lmcrypt`), generates the `SIGN=` keyword. The public key, used by applications and vendor daemon, authenticates the `SIGN=` keyword. It is difficult (but theoretically not impossible) to derive the private from the public key; the longer the signature the harder it is to derive. In practice, cracking the signatures used by FlexNet Publisher would require a large number of resources, considerable mathematical skill, and time.

To put this in perspective, the supplier of the elliptical curve cryptography employed by Tamper Resistant Licenses have an ongoing challenge with a maximum $100,000 reward to crack their cryptography. In 1999, a researcher required 195 volunteers, 40 days of calculation, 16000 MIPS years of computation, and 740 computers located in 22 countries to solve a 97-bit key, which is far weaker than any of the TRL options. See http://www.certicom.com for current information about the ongoing challenge.

Choosing a Signature Strength

This decision determines what strength of protection you want against counterfeiting: the longer the `SIGN=` keyword, the greater the protection. The strength is defined by setting `LM_STRENGTH` in `machind/lm_code.h` to the appropriate value.

Here are samples of `FEATURE` lines with each length:

```
LM_STRENGTH_113BIT (60 chars):
    FEATURE f1 xyzd 1.0 permanent uncounted 1234567890AB \
    HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
    90AB CDEF 1234 5678 90AB CDEF 4078 D48A BC93"
LM_STRENGTH_163BIT (84 chars):
    FEATURE f1 xyzd 1.0 permanent uncounted 1234567890AB \
    HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
    90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 \
    5678 90AB CDEF 1234"
LM_STRENGTH_239BIT (120 chars):
    FEATURE f1 xyzd 1.0 permanent uncounted 1234567890AB \
    HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
    90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 \
    5678 90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 5678"
```

Note ▪ *Consider TRL strength carefully. If you implement one TRL strength in an application now it cannot be changed later.*

Migrating to Tamper-Resistant Licensing

The information in this section is for publishers who want to implement TRL, for the first time, into their existing FlexEnabled application. This is the equivalent of moving to configuration 5 (see Table 11-1, Signature configurations).

The following components compose TRL:

- TRL keys (to be set in `machind/lm_code.h`)
- License file signed using TRL
- TRL-enabled vendor daemon
- TRL-enabled application
- TRL-enabled license generators

If a TRL-enabled application attempts either to communicate with a non-TRL vendor daemon or to authenticate a non-TRL license, an error message is displayed and the application does not run. The error message informs you that either the license file and/or the vendor daemon is not TRL compliant.

It's important to remember that the TRL application itself is the *trigger* that requires both TRL licenses and a TRL-enabled vendor daemon in order to run.

If your previous FlexNet Publisher toolkit (that includes client application, vendor daemon and license generation tool) was built with `LM_STRENGTH` set to `LM_STRENGTH_DEFAULT` (in `machind/lm_code.h`), then it is not possible to upgrade to TRL. The upgraded server and the licenses will not work with older clients that were built with `LM_STRENGTH_DEFAULT`.

If your previous toolkit was built with `LM_STRENGTH` set to `LM_STRENGTH_LICENSE_KEY`, you can perform a partial migration which maintains backward compatibility with older clients. For more information, see Maintaining Backward Compatibility.

Using the Correct Client Library

The TRL-only libraries provide increased security when using TRL licenses. They prevent circumvention of TRL.

If you use are using Tamper Resistant Licensing for your product and you do not need backward compatibility with non-TRL-licenses, use the following client libraries:

- On Windows: `lmgr_trl.lib`
- On UNIX: `liblmgr_trl.a` or `liblmgr_pic_trl.a`

FlexEnabled applications built with these libraries will work with TRL licenses only. If you try to use the FlexEnabled application with a non-TRL license, the checkout will fail with an "Invalid (inconsistent) license" error message.

See the *Development Environment Guide* for information about building the vendor daemon.

For detailed information about configurations and library linkage, see Table 11-1, Signature configurations.

If you are applying TRL to an existing product for the first time, note that you cannot use the following attribute settings (used to provide backward compatibility with non-TRL licenses) within FlexEnabled applications built with these libraries:

- `LM_A_SIGN_LEVEL` set to `zero`
- `LM_A_STRENGTH_OVERRIDE` set to `LM_STRENGTH_DEFAULT`

Enabling TRL in Your FlexEnabled Application

The following procedure lists the steps necessary to create a TRL-enabled toolkit and application (configuration 5 as per Table 11-1, Signature configurations). Note that this procedure is a complete migration to TRL. After the migration, the new TRL-enabled vendor daemon can no longer serve the old licenses that were built using `LM_STRENGTH_DEFAULT` to the old client applications. For information about a partial migration which maintains backward compatibility with older clients, see Maintaining Backward Compatibility.

Task ***To create a production toolkit and application that uses tamper-resistant licensing:***

1. Open `machind/lm_code.h` in a text editor.
2. Locate the `LM_STRENGTH` line:

   ```
   #define LM_STRENGTH LM_STRENGTH_DEFAULT
   ```

3. Set `LM_STRENGTH` to the desired value (`LM_STRENGTH_xxxBIT`). See Choosing a Signature Strength for a discussion on `LM_STRENGTH` values.
4. Locate the two dummy `TRL_KEY` lines:

   ```
   #define TRL_KEY1 0x0
   #define TRL_KEY2 0x0
   ```

5. Replace the two `TRL_KEY`s with those that you received from Revenera.
6. Save and close `machind/lm_code.h`.
7. Build the production toolkit using `make` (UNIX) or `nmake` (Windows). Complete details for building the toolkit are found in "Building the Production Licensing Toolkit" in the *Development Environment Guide*.
8. Enable TRL in your application by relinking it to a TRL-only library (see Using the Correct Client Library). Follow instructions in "Building Your FlexEnabled Application for License File–Based Licensing" in the *Development Environment Guide*.
9. Using the files from the toolkit built in Step 7, generate new TRL-enabled licenses.
10. Ship the following components to the user:
    - License file with the TRL signature
    - TRL-enabled vendor daemon
    - TRL-enabled application

Maintaining Backward Compatibility

This section describes a partial migration to TRL (configuration 1 to 4 in Table 11-1, Signature configurations) which maintains backward compatibility with older clients. Note that Revenera strongly recommends a full migration to TRL-only licenses, to reduce the risk of counterfeiting.

Refer to Table 11-1 for an overview of possible migration scenarios and elements that are required for each signature configuration.

Client Library

If you previously used (non-TRL) licenses with the keyword `LM_STRENGTH_LICENSE_KEY` and are now upgrading to TRL but want to maintain backward compatibility with older license files, you need to link a non-TRL client library to the vendor daemon. This will enable the vendor daemon to serve the older licenses to the older client applications. These libraries are:

- On Windows, `lmgr.lib`

- On UNIX: `liblmgr.a` or `liblmgr_pic.a`

If you previously used (non-TRL) licenses with the keyword `LM_STRENGTH_DEFAULT`, a partial migration to TRL is not possible. Instead, you would need to perform a full migration and re-issue new licenses for all clients. For more information, see Migrating to Tamper-Resistant Licensing.

License File

If you want the vendor daemon to serve old non-TRL licenses (with the keyword `LM_STRENGTH_LICENSE_KEY`) and TRL licenses, you need to include the old license key and the new TRL license signature (using the `SIGN2` keyword) in the same feature line. The `FEATURE` line would look similar to this:

```
FEATURE f1 demo 1.0 permanent 4 A20045D0B095 SIGN2="003D B062 54D1 47E3 1FF6 145F \
    59CE 0200 8002 FE69 0FBC F4FF AA10 FFB3 6EB3"
```

The highlighted hexadecimal character sequence indicates the old 12-digit license key.

For other migration scenarios, consult Table 11-1 about the elements required in the license file.

Setting Application Attributes

Producers who are using TRL but want to be backward compatible with non-TRL licenses should set these attributes as follows:

- `LM_A_SIGN_LEVEL` set to `zero`
- `LM_A_STRENGTH_OVERRIDE` set to `LM_STRENGTH_DEFAULT`

However, producers who wish to harden their licensing applications against exploits should not set the above attributes.

12

Cross-Version License Files

For software publishers that want to begin using a newer version of FlexNet Publisher while still supporting FlexEnabled applications that used client libraries from earlier releases, cross-version license files simplify potential deployment issues.

Normally, an application built with a particular version of FlexNet Publisher cannot validate a license that includes keywords introduced in a newer version of FlexNet Publisher. Cross-version licenses enable you to issue license files containing new licensing keywords to customers who will continue to use older versions of your FlexEnabled application alongside newer versions of your application.

Important ▪ *Cross-version license files are only applicable to served licenses.*

When a FlexEnabled application checks out a feature from a license server, it authenticates and validates the information in the feature definition line against the signature. Typically, it must recognize all keywords present in the feature definition line in order to validate this information.

To create a cross-version license file, label each feature definition line that contains the new keyword with the older client library versions that you want to support. When an older FlexEnabled application authenticates the feature definition line, the newer keywords are ignored and it uses the version specific signature.

The following sections in this chapter provide more detail:

- Identifying If You Need to Use Cross-Version License Files
- Implementing Cross-Version License Files
- Regenerating the License File

Identifying If You Need to Use Cross-Version License Files

Cross-version licensing is useful to software publishers who

- Are deploying or have deployed a FlexEnabled application that is licensed using one or more of a specified set of keywords (see Table 12-1).
- Have deployed a previous version of the same application that incorporates a version of the FlexNet Publisher client libraries that don't recognize one of these keywords (see Table 12-1).
- Have both later (newer) and earlier versions of this FlexEnabled application actively in use at the customer site.

Cross-Version license files are not required if your clients are V11.12.0 or later.

- In clients < v11.12, the INCREMENT line received by the server is both authenticated (signature checked) and validated (keywords and their values are checked against the client environment). This means that a keyword sent to the client which is not recognized by the client causes a LM_FUTURE_FILE error. A cross version signature (with any client-unsupported keyword removed from the signature calculation) is therefore needed to support legacy clients.
- In clients >= v11.12, all keywords are authenticated against the INCREMENT line signature as before. However, unrecognized keywords are not validated, meaning cross version signatures will not be needed for 11.12 and later clients.

Cross-Version License Scenario

Consider the following scenario. You are building a next-generation FlexEnabled application that takes advantage of a new feature keyword introduced in the current release of FlexNet Publisher. This keyword was not available in previous releases. Both the older FlexEnabled application and the new application will coexist at your customer sites and checkout the same feature name. Only the newer FlexEnabled application uses the new keyword.

You plan to issue a new license file containing this new keyword, but you want older (<v11.12) FlexEnabled applications to successfully authenticate the information in the feature definition line. The problem is that the client libraries in the older FlexEnabled applications will not recognize the new keyword.

To solve this problem, you can issue a cross-version license file that serves licenses to all versions of your FlexEnabled applications. Without a cross-version license file, you must either redeploy each version of the application with the new client library or issue separate license files for each version of the application, each containing the keywords that version understands.

Use the table below to help determine whether cross-version license files are necessary for your FlexEnabled application.

Version Signatures for Cross-Version License Files

Consult Table 12-1 to determine if you can apply cross-version license signatures to your existing applications. This table cross-references the keyword with the version of the client libraries your FlexEnabled application uses. Where a cross-version signature can be used, the table shows the correct version signature. (Empty cells mean no cross-version signature is necessary.) For example, to use the `VM_PLATFORMS` keyword with a FlexEnabled application that uses client library version 11.0, use the V11.4 version signature.

Table 12-1 ▪ Version Signatures for Cross-Version License Files

	Version of Client Libraries in FlexEnabled Application						
Keyword	**7.1-7.2**	**8.0**	**8.1-8.3**	**8.4**	**9.0-10.8.n**	**11.0-11.4.1**	**11.5-11.6.1**
BORROW	V7.1						
COMPOSITE	V7.1	V8.0	V8.1	V8.4			
FLOAT_OK	V7.1						
SUITE_RESERVED	V7.1	V8.0	V8.1				
SUPERSEDE_SIGN	V7.1	V8.0	V8.1	V8.4	V10.8	V11.4	
TZ	V7.1	V8.0	V8.1	V8.4	V10.8	V11.4	V11.6
VM_PLATFORMS	V7.1	V8.0	V8.1	V8.4	V10.8	V11.4	V11.6

Implementing Cross-Version License Files

Implementing a cross-version license file consists of the following steps:

1. From Table 12-1, determine the keyword you need to include in an existing feature definition line.
2. Rebuild your vendor daemon and license generator (`lmcrypt`). See the *Development Environment Guide* for information about rebuilding the vendor daemon.
3. Regenerate your license file using the revised feature definition lines. See Regenerating the License File.
4. Link the new version of your FlexEnabled application with the FlexNet Publisher client libraries from the toolkit built in step 3. See the *Development Environment Guide* for details.
5. Deploy the rebuilt vendor daemon and your rebuilt FlexEnabled application.
6. Distribute the new cross-version license file according to your standard practice for updating your user.

See Also
`LM_A_LKEY_CASE_SENSITVE` in the *C/C++ Function Reference*:
ls_single_xver_signature

Regenerating the License File

A feature definition line in a cross-version license file contains multiple signatures. Each version-specific signature is generated using only those license keywords that are valid for that version.

Task ***To regenerate your license files:***

1. Add the new keyword, as determined from Table 12-1, to the feature definition line. This becomes your cross-version license file.
2. Regenerate the license, using the current `lmcrypt` utility or a compatible license generator.
3. Examine the new license file.

Each keyword that you append in step 1 above has been converted to one or more version-specific signatures. Each corresponding to a signature type in the original template.

Examples

1. Let us assume you have 11.4, 11.6 and 11.11 clients already deployed in the field and you wish to introduce the TZ keyword.

   The following are the two solutions for you to choose accordingly:

   - In this scenario use the V11.4 cross-version signature from Table 12-1, for the 11.4 clients, the V11.6 cross-version signature for 11.6 clients and the 11.11 clients understand the TZ keyword, and are fine with the non-versioned signature.

     ```
     INCREMENT f1 demo 1.0 permanent 5 TZ=SERVERTZ SIGN= V11.4_SIGN= V11.6_SIGN=
     ```

   - Use only one cross-version signature (corresponding to the oldest client you have) and set vendor variable ls_single_xver_signature = 1. When set, this vendor variable ensures that any client with version < 11.12 is sent the first cross-version signature discovered on the increment line. ls_single_xver_signature is intended to allow publishers to only have to deliver two signatures in license files: one corresponding to their oldest client, and the non-versioned signature

     ```
     INCREMENT f1 demo 1.0 permanent 5 TZ=SERVERTZ SIGN= V11.4_SIGN=
     ```

     In this solution, all clients with version <11.12 gets the v11.4 cross-version signature. The clients with version 11.11 gets the TZ data, since this is additionally sent in the configuration data for clients >=11.7

2. Following on from Example 1, suppose a future version of FlexNet Publisher (11.99) offers the authenticated 'KWDZ' keyword. You wish to support all the old clients, including the 11.12 clients deployed from Example 1, and you wish to introduce this keyword for your new clients.

   Since you have a client with 11.7 <= version <= 11.11 deployed in the field, and there is no cross-version-signature for these client versions, you need to set ls_single_xver_signature = 1, in order to force the 11.11 client to accept an older cross-version signature. If ls_single_xver_signature is not set, the V11.11 client will fail validation on the KWDZ keyword.

   ```
   INCREMENT f1 demo 1.0 permanent 5 KWDZ=123 TZ=SERVERTZ SIGN= V11.4_SIGN=
   ```

   All deployed clients (11.4, 11.6, 11.11, 11.12, and 11.99) work with this configuration. The 11.12 client will receive and authenticate the KWDZ keyword, but will not attempt to validate it.

After giving the above increment line in the server license file, the following client checkout behavior can be expected:

Table 12-2 ▪ Results of cross version signature for License versions.

client version	ls_single_xver_signature	checkout result
11.4	0 or1	successful
11.6	0	LM_FUTURE_FILE
11.6	1	successful
11.11	0	LM_FUTURE_FILE
11.11	1	successful
11.12	0 or 1	successful

Please refer to FlexNet Publisher EOL website (https://docs.revenera.com/eol/flexnet-publisher.htm) for support statement on old versions.

13

IPv6 Support

Internet Protocol version 6 (IPv6) is the next generation IP protocol. This section contains information for software publishers who want to support IPv6 addresses in their license model. The information in this section assumes the reader has a familiarity with the IPv6 networking protocols.

Capabilities that Support IPv6

When working with a software publisher to obtain a software package that supports IPv6, you should collect and provide the IP addresses of systems (FlexEnabled clients and license servers) that will be used in the license file.

Note ▪

- *In the license and options files, FlexNet Publisher supports only the site-local form of the IPv6 address (those addresses prefixed with FEC0).*
- *While an IPv6 address can be used in license or options files, the best-practice recommendation is to use hostname or IPv4 address.*
- *A mix of IPv4 and IPv6 addresses in the license and/or options file is not supported.*

License File

In a license file, the `SERVER` line can define an IPv6 address as the hostid and host value.

In `FEATURE`, `INCREMENT`, and `UPGRADE` lines, an IPv6 address can be used as the `HOSTID` value when using the `INTERNET` type.

The following keywords (used with duplicate grouping) support IPv6 addresses:

- The `DUP_GROUP` keyword, using `HOST` as the type.
- The `SUITE_DUP_GROUP` keyword using `HOST` as the type.

Options File

An options file can contain an IPv6 address to specify host restrictions when using the:

- `INTERNET` type in these keywords: `EXCLUDE`, `EXCLUDEALL`, `EXCLUDE_BORROW`, `INCLUDE`, `INCLUDEALL`, `INCLUDE_BORROW`, `MAX`, and `RESERVE`.
- `HOST` type in these keywords: `EXCLUDE`, `EXCLUDE_ENTITLEMENT`, `EXCLUDEALL`, `EXCLUDE_BORROW`, `INCLUDE`, `INCLUDE_ENTITLEMENT`, `INCLUDEALL`, `INCLUDE_BORROW`, `MAX`, and `RESERVE`
- `HOST_GROUP` keyword (it takes IP addresses).

License Search Path

Entries in the license search path that use the 'port@host' convention to identify the license server, can specify an IPv6 address as the 'host' value.

FlexEnabled Application

Entries in the license search path that use the *`port@host`* convention to identify the license server, can specify an IPv6 address as the *`host`* value.

- **flxActCommonHandleSetRemoteServer(...)**
- **flxActBorrowReturn(...)**
- **flxActBorrowActivate(...)**

The following attributes, set using the **lc_set_attr** function, support IPv6 addresses:

- `LM_A_HOST_OVERRIDE`
- `LM_A_PROMPT_FOR_FILE` (Windows Only) - the dialog that appears takes an IPv6 address.
- `LM_A_DISPLAY_OVERRIDE`
- `LM_A_INTERNET_OVERRIDE`

You can use an IPv6 address when setting the `LM_DUP_HOST` flag in the **lc_checkout** function.

License Server

The `lmhostid` utility returns an IPv6 address.

Environment Variables

Listed below are the environment variables that handle performance and hostname resolutions.

FNP_IP_ENV

This client-side environment variable determines how the client's IP address is presented to the server.

- If the variable is set to a value of 0, 4, or 6, the IP address sent to the server is resolved on the client, from the client's hostname, as follows:
  - If the variable is set to 0, then the client resolves its hostname to both an IPv4 and an IPv6 address.
  - If the variable is set to 4, then the client resolves its hostname only to an IPv4 address.
  - If the variable is set to 6, then the client resolves its hostname only to an IPv6 address.

- If the variable is set to 1 (default value), the client-side hostname resolution is bypassed. Instead, the client's IP address is determined from the socket connection at the server, which means that a NAT-translated IP address for the client can be obtained.

FNP_IP_PRIORITY

The environment variable `FNP_IP_PRIORITY` decides the IP priority for hostname resolution, and can apply to the server, the client, or utilities like `lmhostid`. Values are:

- **4** (default): hostname resolution is attempted to IPv4 address first; if that fails, resolution to IPv6 address is attempted.
- **6**: hostname resolution is attempted to IPv6 address first; if that fails, resolution to IPv4 address is attempted. Examples of when to set to this value:
  - on the machine where running `lmhostid`, if IPv6 address is required from `lmhostid -internet`
  - on the client if node-locked IPv6 INTERNET HostID is used
  - on the server if IPv6 address is used in the options file or as a server HostID.

    Note: IPv4 and IPv6 addresses cannot be mixed on the server.

Client IP Address Received by the Server

Clients send both IPv4 and IPv6 addresses in the checkout message to the server.

In some licensing models, it may be desirable for the server to operate on a NAT-translated IP address instead of the client's actual IP address—for example, if using the `EXCLUDE` keyword in an options file to exclude all clients originating from behind a specific firewall. To enable such use cases, set `FNP_IP_ENV=1` on the client. Setting this environment variable prevents the client resolving its own hostname to an IP address, which means the server instead obtains an IP address for the client from the socket connection.

Using Wildcards in an IPv6 Address

The wildcard character, “*,” may be used in place of an entire field or on a byte-by-byte basis to specify a range of addresses without having to list them all. For example, in this example feature definition line is locked to four specific addresses:

```
FEATURE f1 myvendor 1.0 31-dec-2020 uncounted \
     HOSTID="INTERNET=127.17.0.1,\
        INTERNET=fec0::1:e947:a213:1378:253c,\
        INTERNET=127.17.0.4,\
        INTERNET=fec0::1:e947:a213:1378:253c" \
     SIGN=0
```

In the following example feature definition line specifies an entire range of addresses, including the four specific ones from the line above:

```
FEATURE f1 myvendor 1.0 31-dec-2020 uncounted \
     HOSTID="INTERNET=127.17.0.*,\
        INTERNET=fec0:0db8:0000:0000:*:*:*:000*"\
     SIGN=0
```

Testing Considerations

When testing IPv6-compatible FlexEnabled applications and license server, remember to accommodate for the behavior of 4to6 routers, 6to4 routers, and IPv6 firewalls configured on your system.

14

Hostids

A hostid is a means used to uniquely identify a specific system. There are two different contexts in which a hostid is used:

- **A license**—The license is bound to one or more hostids, which are used to define which FlexEnabled client can run the application. The hostid is placed in the feature definition line.
- **A license server**—A hostid used to define which system is authorized to run a license server that serves licenses to the FlexEnabled application. The hostid is placed in the `SERVER` line.

Each platform supports at least one method of determining its hostid. You need to determine which hostid types you accept for each platform you support.

The `lmhostid` utility prints the default hostid(s) that FlexNet Publisher expects to use on any given platform. FlexNet Publisher supports platform-specific as well as publisher-defined hostids. In addition, there are a number of special hostid types that apply to all platforms. These special hostid types can be used on either a `SERVER` line or a feature definition line, wherever a hostid is required.

The decision to use a particular hostid is made in the license file rather than the FlexEnabled application. However, there are occasions when the FlexEnabled application may want to determine a hostid type specified in a particular feature definition line, for example, to detect the usage of special hostid types such as `DEMO` and `ANY`. The Flexible API function, **lc_auth_data**, called by the application, provides this information.

There are two types of hostids—those valid for a specific platform and those valid on all platforms. The following section describes hostids that are available only on some platforms. Special Hostids describes hostids that are available on all platforms.

Platform-Specific Hostids

The following table lists all the platform-specific hostids by platform.

Obtaining any TPM, virtualization, or cloud-licensing hostid requires that the FlexNet Licensing Service be installed.

Table 14-1 ▪ Hostids Available for Specific Platforms

Operating System	Description
AIX	32-bit hostid
HP-UX	String ID, maximum length is `MAX HOSTID_LEN`
Mac OS	Ethernet address
	FlexNet ID dongles: `FLEXID=9`, `FLEXID=10`
Microsoft Windows	Windows disk serial number
	Ethernet address
	UUID support on virtual platforms
	Internet v4 or v6 address, Amazon Instance ID, and Amazon template ID in Amazon EC2 environments
	Enforced physical machine
	TPM (Trusted Platform Module) version 2.0 (version 1.2 is not supported) ***Note ▪*** *Obtaining this hostid requires that the FlexNet Licensing Service be installed.*
	FlexNet ID `dongles: FLEXID=9, FLEXID=10, PHY_FLEXID=9, PHY_FLEXID=10`
	Generation ID of the virtual machine ***Note ▪*** *This identifier is available only in Windows VMs.* GenerationID cannot be used together with the HOSTID keyword. Refer to the FlexNet Publisher Virtualization White Paper for examples of how to use GenerationID.
	CONTAINER ID hostid identifies a docker container.

Table 14-1 ▪ Hostids Available for Specific Platforms

Operating System	Description
Linux	Ethernet address
	UUID support on virtual platforms
	Internet v4 or v6 address, Amazon Instance ID, and Amazon template ID in Amazon EC2 environments
	Enforced physical machine
	FlexNet ID dongles: `FLEXID=9`, `FLEXID=10`, `PHY_FLEXID=9`, `PHY_FLEXID=10`
	CONTAINER ID hostid identifies a docker container.
Solaris	32-bit hostid
	Ethernet address

Using Ethernet Address as a Hostid

The ethernet address (also referred to as a MAC address) of ethernet devices or NICs (network interface cards) can be used as a hostid. Several devices with an ethernet address can be attached to a machine; some of these may be virtual devices that may generate a different ethernet address each time they are activated. An example of a virtual device that may generate an ethernet address is VPN (virtual private network) software.

Some devices that have an ethernet address can be detachable from the machine. For example, a laptop plugged into a docking station uses the ethernet address of the docking station; however, when it is disconnected from the docking station, the ethernet address is no longer available. A wireless adapter also has an ethernet address and this address is not available when either the wireless adapter is removed from the machine or when the wireless adapter is disabled, but still physically attached to the machine.

When an ethernet address is used as a hostid, or is included in a composite hostid, you should ensure that you choose a permanent ethernet address. If you use an ethernet hostid that can be removed or changed, then your end user will lose licenses when the ethernet device is removed or changed. Information about virtual ethernet adapters can be obtained using system-specific commands. Ask your end users to obtain detailed information about all of the devices on their system that have ethernet addresses and choose an appropriate ethernet address as a hostid.

It is important that a *permanent* ethernet address is chosen when it is used as a hostid on a `SERVER` line. The license server shuts down when the hostid to which it is locked is removed or changed.

On Windows platforms, `lmhostid` filters out the hostid of any device that it identifies as a virtual ethernet adapter. The Flexible API attribute, `LM_A_PHYSICAL_ETHERNETID`, can be set so that `lc_hostid` operates to provide the same filtering of ethernet hostids and returns the same list of ethernet hostids as that provided by `lmhostid`. Additionally, the vendor daemon can be customized so that only hostids from physical ethernet adapters are used when authenticating a license (see ls_allow_physical_ethernetid_only). The default behavior of `lc_hostid` and the vendor daemon is to include virtual ethernet hostids; therefore, you must make changes to your code or the vendor daemon if you want to remove virtual ethernet hostids.

Note ▪ *FlexNet Publisher considers the address of a team-bonding virtual adaptor (for teamed Ethernet interfaces) as a stable identifier for use as a hostid on Windows and Linux machines. If this address is available on a Windows machine, FlexNet Publisher returns this hostid despite the value of* `LM_A_PHYSICAL_ETHERNETID` *or* `ls_allow_physical_ethernetid_only`.

The Flexible API function `lc_hostid` and the utility `lmhostid` may report multiple ethernet hostids on some platforms.

See Also
`lc_hostid` in the *C/C++ Function Reference*
`LM_A_PHYSICAL_ETHERNETID` in the *C/C++ Function Reference*
Multiple Composite Hostids for HOSTID_ETHER
`lmhostid` in the *License Administration Guide*

TPM Hostid

FlexNet Publisher supports the TPM (Trusted Platform Module) hostid for to uniquely identify a computer, specified as TPM_ID1 in license files. The TPM hostid is currently supported on Windows platforms only.

As a prerequisite to obtaining and using the TPM hostid, a TPM version 2.0 device must be available and enabled. In addition, the FlexNet Licensing Service must be installed.

The TPM hostid is only supported on the SERVER line. It is supported as a served node-locked hostid from FlexNet Publisher version 11.15.0 onwards. The TPM hostid is not supported for trusted storage–based licensing.

You can identify the TPM status using the utilities `lmtpminfo` and `lmhostid`, or the API **lc_tpmstatusget**, or natively in Windows by using the Microsoft Management Console with `tpm.msc` specified, or by using WMI and the `Win32_Tpm` class.

Troubleshooting

If you encounter problems when trying to obtain the TPM hostid, check that the following requirements have been met:

- The FlexNet Licensing Service must be installed.
- The TPM must be turned on and enabled in the machine's BIOS. To enable it, do the following:
  1. Restart the machine. During the restart, follow the instructions on the screen that explain how to interrupt normal startup and enter the BIOS setup utility. Locate the Security Settings and enable the TPM. Reboot the system. Run `lmtpminfo` or `lmhostid` to identify the TPM status. If the status cannot be obtained, continue with step 2.
  2. Open the TPM management console (`tpm.msc`). If the status is `TPM not fully enabled`, it may be necessary to enable UEFI mode in the machine's BIOS.

Using FlexNet ID Dongles

A FlexNet ID dongle is a hardware device that locks FlexNet license rights to the machine to which the dongle is attached. Each FlexNet ID dongle contains a unique identity. This identity is used to provide a hostid. This type of hostid is referred to as a *FLEXID*.

The FLEXID can be used to lock license rights either to a server or to an end-user machine. FlexNet dongles are normally used with license rights that are held in license files and this document assumes that this is the case in all examples.

For the FlexNet ID dongle to communicate with the computer it is attached to, the appropriate drivers must be installed on the computer. Drivers are platform- and OS-version- specific. For a complete list of FlexNet ID dongles, the platforms on which they are supported, and driver installation information, see the *Driver Installation Guide for FlexNet ID Dongles*.

Container ID

FlexNet Publisher supports the Container ID hostid to uniquely identify a docker container. The Container ID hostid is currently supported on Windows and Linux platforms.

The Container ID hostid is supported on the SERVER and FEATURE line. It is supported as a served node-locked hostid. The Container ID hostid is supported for certificate based licensing.

You can identify the Container ID status using the lmhostid utility, or the API **lc_dockerstatusget**.

Special Hostids

The licensing toolkit contains a number of special hostid types that apply to all platforms. These hostid types are valid to use in both `SERVER` lines and `FEATURE` lines, wherever a hostid is required.

Table 14-2 ▪ Special Hostid Types

Hostid	Description
ANY	Locks the software to any system (meaning that it does not lock anything).
DEMO	Similar to `ANY`, but only for use with uncounted `FEATURE` lines.
COMPOSITE= `<composite_hostid>`	Locks the software to a composite hostid. The composite hostid can be made up of any number of the simple hostids supported by FNP. These are aggregated together and the resultant buffer hashed down to a 12-byte hexidecimal value. The process is one-way hashing, hence makes it impossible to reverse the constituent input simple hostids. Composite hostids are not returned by `lmhostid` or `LMTOOLS`. When composite hostids are used, the software publisher needs to provide a utility that determines the publisher's composite hostid. On some systems, multiple composite hostids may be provided, any of which may be used to identify the system to which the software is locked. ***Note ▪*** *We do not support Dongles in composite_hostid.*

Table 14-2 ▪ Special Hostid Types

Hostid	Description
DISPLAY= *<display>*	Locks the software to display. ● On UNIX, `display` is `/dev/ttyxx` (which is always `/dev/tty` when an application is run in the background) or the X-Display name. ● On Windows, it is the system name or, in the case of a terminal server environment, the terminal server client name (version 8 or later FlexEnabled applications only)
HOSTNAME= `<host>`	Locks the software to computer host name `<host>`
ID=`<n>`	Functionally equivalent to the `ANY` hostid—it runs on any system. The difference is that the license is unique and is used to identify the end user. This hostid is used to lock the license server (on the `SERVER` line) or the FlexEnabled application (on the feature definition line). The number can include dashes for readability—the dashes are ignored. Examples: ● **`ID=12345678`** is the same as ● **`ID=1234-5678`** is the same as ● **`ID=1-2-3-4-5-6-7-8`**
INTERNET= `<IP address(es)>`	Locks the software to an internet IP address, or group of IP addresses. Wildcards are allowed. For example, **`198.156.*.*`** means any host with a matching internet IP address. The main use is to limit usage access by subnet, implying geographic area. For this purpose, it is used on the feature definition line as a hostid lock. For more information about obtaining this hostid, see the description for the `lmhostid` utility (`-internet` option) in FlexNet Publisher's *License Administration Guide*. ***Note ▪*** *(Windows only) When a Windows machine is connected to a VPN,* `lmhostid -internet` *returns the virtual adapter IP address. To ensure that this command returns the physical adaptor IP address during checkouts, the user needs to perform this workaround: Run* `regedit`*, navigate to* `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Linkage`*, locate the* `Bind` *parameter, and edit the adapter order so that the physical adapter is listed first. Run* `ipconfig/registerdns` *to put the edit into effect.*

Table 14-2 ▪ Special Hostid Types

Hostid	Description
VM_GENID	Generation ID of the virtual machine ***Note ▪*** *This identifier is available only in Windows guests.* *Requires the FlexNet Licensing Service to be installed.* *GenerationID cannot be used together with the HOSTID keyword. Refer to the FlexNet Publisher Virtualization White Paper for examples of how to use GenerationID.*

Virtualization Hostids

The hostids are:

- VM_UUID
- VM_GENID
- ETHER
- FLEXID (when USB passthrough is supported by the hypervisor)

Note ▪ *The FlexNet Licensing Service must be installed on machines that use virtualization features.*

Virtualization hostids and the ls_allow_vm Variable

In conjunction with the `ls_allow_vm` variable, a publisher can use this type of keyword to allow different virtualization policies on a per-customer basis. For example, a publisher could build a restrictive vendor daemon that will not work on a virtual machine (`ls_allow_vm=PHYSICAL`), but then override this setting on a per-customer basis by issuing a new license file that contains a `VM_UUID` value in the `SERVER` line.

Important ▪ *Three-server redundancy configurations require all three servers use the same platform type. You can use any of the following platform types in the configuration, but each server must use the same platform type: VM_UUID or PHY_*.*

Cloud-Licensing Hostids

FlexNet Publisher uses the following hostids to bind licenses to Amazon machine instances in an Amazon EC2 environment.

Obtaining a cloud-licensing hostid requires that the FlexNet Licensing Service be installed.

Table 14-3 ▪ Hostids in an Amazon EC2 Environment

Hostid Type	lmhostid Syntax	License File Syntax	License File Example
Elastic IP (EIP) address **License server only**	**Syntax:** *./lmhostid -ptype AMZN -eip* **Output:** The FlexNet host ID of this machine is "AMZN_EIP=52.221.107.30"	**Syntax:** *AMZN_EIP=IPv4 address* **Example:** SERVER this_host AMZN_EIP=52.221.107.30	**Example:** SERVER this_host AMZN_EIP=52.221.107.30 27007 VENDOR qavend1 PORT=27006 USE_SERVER # counted license FEATURE f1 qavend1 1.0 permanent 4 SIGN=xxx
AMI template ID	**Syntax:** *./lmhostid -ptype AMZN -ami* **Output:** The FlexNet host ID of this machine is "AMZN_AMI=ami-2c95344f"	**Syntax:** *AMZN_AMI=AMI template ID* **Example:** INCREMENT F1 demo... HOSTID=AMZN_AMI=ami-2c95344f...SIGN=xxx	**Example:** # unserved node-locked license FEATURE f2 qavend1 1.0 permanent uncounted HOSTID=AMZN_AMI=ami-2c95344f \ SIGN=xxx
VM_UUID (Amazon Instance ID in Amazon EC2 environment) **License server only**	**Syntax:** *./lmhostid -ptype VM -uuid* **Output:** The FlexNet host ID of this machine is "VM_UUID=i-09ddb75647ab1a499"	**Syntax:** *VM_UUID=Amazon Instance ID* **Example:** SERVER this_host VM_UUID=i-09ddb75647ab1a499	**Example:** SERVER this_host VM_UUID=i-09ddb75647ab1a499 27007 VENDOR qavend1 PORT=27006 USE_SERVER # counted license FEATURE f1 qavend1 1.0 permanent 4 SIGN=xxx

Table 14-3 ▪ Hostids in an Amazon EC2 Environment

Hostid Type	lmhostid Syntax	License File Syntax	License File Example
Elastic Network Interface (ENI)	**Syntax:** *./lmhostid -ether* **Output:** `The FlexNet host ID of this machine is ""06007e2c60c7 06835b200fe7""`	**Syntax:** *lmhostid -ether* **Example:** `Host ID =""06835b200fe7 06835b200fe7""` ***Note*** ▪ *ifconfig returns eth0 and eth1.* *eth0(primary network interface)=06835b200fe7 eth1(Elastic Network Interface)=06007e2c60c7*	**Example:** `# unserved node-locked license FEATURE f2 qavend1 1.0 permanent uncounted HOSTID=06007e2c60c7 SIGN=xxx`

Using Composite Hostids

A composite hostid combines hostids together to provide a more secure hostid. A composite hostid is a hashed 12-character hexadecimal value formed by combining the values of one or more simple hostid types. It can be used anywhere a simple hostid is used: on a `SERVER` line or feature definition line of a license file.

Incorporating a composite hostid into your license policy involves the following actions:

- Defining a Composite Hostid—Decide which hostid types are included in your composite hostid.
- Initializing a Composite Hostid in Your Application—Include code in your FlexEnabled application to initialize the composite hostid.
- Initializing a Composite Hostid in Your Vendor Daemon—For served licenses, modify your vendor daemon to initialize the composite hostid.
- Creating a Composite Hostid Utility—Provide a way for your user to derive the composite hostid from the system where your product will be installed.
- Providing a Composite Hostid License—Issue licenses that specify the `COMPOSITE=` hostid type.

See Also
Hostids for Supported Systems
SERVER Lines
HOSTID
Multiple Composite Hostids for HOSTID_ETHER

Defining a Composite Hostid

A composite hostid is defined by a list of simple hostid types. You can incorporate as few or as many types in the list as is appropriate for your license policy. Do not include a hostid type multiple times in the list. When FlexNet Publisher validates the system's composite hostid, it retrieves the values for each hostid type in the list, combines them and applies a hashing algorithm to produce a 12-character hexadecimal value.

At license authentication time, FlexNet Publisher compares the calculated value to that in the `FEATURE` or `SERVER` line. An exact match indicates a legitimate system for processing to proceed. That means the value for each constituent in the composite hostid list must match exactly to that which is used in the original composite hostid calculation. The more hostid types included in the composite hostid, the tighter the tolerance is to accept the system as a legitimate host.

Consult **lc_hostid** in the *C/C++ Function Reference* for a list of simple hostid types that can be included in a composite hostid. The following sections in this chapter use a composite hostid comprised of `HOSTID_ETHER` and `HOSTID_USER`.

Initializing a Composite Hostid in Your Application

Use **lc_init_simple_composite** to initialize the composite hostid in your application. Place code like the following in your application:

```
#include lmclient.h
LM_HANDLE *job
/* Set up the list of hostid types which compose the
   composite hostid, hostids must appear in the same order
   as those in hostid lists specified in other components.
*/
int hostid_list[] = {HOSTID_ETHER, HOSTID_USER};
/* Do not repeat a hostid type: int hostid_list[] = {HOSTID_ETHER, HOSTID_USER,
   HOSTID_ETHER} for example is not allowed. */
int num_ids = 2;

void
main()
{
   /*...*/
   lc_new_job(..., &job);
   /* Register the composite hostid */
   ret = lc_init_simple_composite(job, hostid_list, num_ids);
   if (ret != 0)
   {
      /* error processing */
   }
   /* ... */
   /* Now, any reference to HOSTID_COMPOSITE,
      during this job, calculates it based on both
      HOSTID_ETHER and HOSTID_USER*/
   /*...*/
}
```

Invoke **lc_init_simple_composite** immediately after the call to **lc_new_job.** Checkout requests using licenses that specify the `COMPOSITE=` hostid type succeed only if the hostid in the license matches exactly with that dynamically calculated on the system where the application is installed. That means both composite hostid constituents, `HOSTID_ETHER` and `HOSTID_USER`, have to be defined and match those which make up the hostid specified by the `COMPOSITE=` keyword.

A complete sample application incorporating a composite hostid is located in `examples\composite\lmcompex.c`.

See Also
The following functions in the *C/C++ Function Reference*

`lc_hostid`
`lc_init_simple_composite`

Initializing a Composite Hostid in Your Vendor Daemon

For served licenses, your vendor daemon has to validate the hostid specified on the `SERVER` line with that retrieved from the system where the license server is running. If a composite hostid is specified on the `SERVER` line, then the vendor daemon must validate it.

Task ***To modify your vendor daemon to validate your composite hostid:***

1. Open `machind\lsvendor.c` in a text editor.
2. At the beginning of the vendor initialization routine section, add a line defining **init_composite_hostid** and modify the initial value of `ls_user_init1` from 0 to `init_composite_hostid`.

```
/* Vendor initialization routines */

void init_composite_hostid();
void (*ls_user_init1)() = init_composite_hostid;
```

3. Save and close `machind\lsvendor.c`.
4. Open a new file, `init_composite.c`, in a text editor.
5. In `init_composite.c`, provide code similar to the following for **init_composite_hostid:**

```
#include lmclient.h
/* Set up the list of hostid types which compose the
   composite hostid, hostids must appear in the same order
   as those in hostid lists specified in other components.
*/
int hostid_list[] = {HOSTID_ETHER, HOSTID_USER};
int num_ids = 2;
extern LMHANDLE *lm_job; /*this must be "lm_job" */

void
init_composite_hostid()
{
   /*...*/
   /* Register the composite hostid */
   ret = lc_init_simple_composite(job, hostid_list, num_ids);
   if (ret != 0)
   {
```

```
      /* error processing */
   }
   /* ... */
   /* Now, any reference to HOSTID_COMPOSITE,
      during this job, it is calculated based on both
      HOSTID_ETHER and HOSTID_USER*/
   /*...*/
}
```

For your convenience, this routine is provided in the toolkit as `examples\composite\init_composite.c`.

6. Open `<platform_dir>\makefile` in a text editor. This example uses a Windows makefile.
   a. Add the vendor-defined hostid code to the list of `EXECS`. For this example, add `init_composite.obj`.
   b. After the `$(DAEMON)` section, add a section to build `init_composite.obj`. For example:

```
init_composite.obj :   $(SRCDIR)\init_composite.c
        $(CC) $(CFLAGS) -I..\h $(SRCDIR)\init_composite.c
```

   c. Add `init_composite.obj` to the link line for `$(DAEMON)`.
7. Rebuild your vendor daemon and distribute it to your customer.

See Also

The following functions in the *C/C++ Function Reference*

`lc_hostid`
`lc_init_simple_composite`

Creating a Composite Hostid Utility

Because the `LMTOOLS`, `lmutil`, and `lmhostid` utilities cannot determine the composite hostid value, you need to provide a means for your user to determine the composite hostid value for the system on which your product runs, and for served licenses, the license server. You use this value in `FEATURE` and `SERVER` lines that specify the `COMPOSITE=` hostid type.

Note ▪ *Host names generated dynamically will change the composite hostid value.*

Provide a utility with code similar to the following:

```
#include lmclient.h
LM_HANDLE *job
char buf[MAX_CONFIG_LINE];
/* Set up the list of hostid types which compose the
   composite hostid, hostids must appear in the same order
   as those in hostid lists specified in other components.*/
int hostid_list[]={HOSTID_ETHER, HOSTID_USER};
int num_ids = 2;

void
main()
{
   /*...*/
   lc_new_job(..., &job);
   /* Register the composite hostid */
```

```
   ret = lc_init_simple_composite(job, hostid_list, num_ids);
   if (ret != 0)
   {
      /* error processing */
   }
   /* ... */
   /* Now, call lc_hostid specifying HOSTID_COMPOSITE
      as the hostid type */
   /*...*/
   ret = lc_hostid(job, HOSTID_COMPOSITE, buf);
   if (ret == 0)
   {
      printf(
        "The Composite HostID of this system is %s\n",
         buf);
   }
   else
   {
      lc_get_errno(job);
   }
   lc_free_job(job);
   exit(0);
}
```

For your convenience, this code is provided in its full form in `examples\composite\lmcomposite.c`.

Distribute this utility to your user.

See Also
The following functions in the *C/C++ Function Reference*

`lc_hostid`
`lc_init_simple_composite`

Providing a Composite Hostid License

To take advantage of your composite hostid, you need to issue licenses to your user that include the `COMPOSITE=` hostid type. This hostid type can be used anywhere a hostid is specified.

- For `FEATURE` lines in your license file:

  `HOSTID=COMPOSITE=`*`composite_hostid`*

- For `SERVER` lines in your license file:

  `COMPOSITE=`*`composite_hostid`*

—where *`composite_hostid`* is a value your user provides to you via your composite hostid utility, for example `lmcomposite`, as described in Creating a Composite Hostid Utility.

See Also
SERVER Lines
HOSTID

Multiple Composite Hostids for HOSTID_ETHER

On platforms where multiple ethernet hostids are reported (see Using Ethernet Address as a Hostid and the current FlexNet Publisher *Release Notes* for details), and the following conditions are met—

- There are more than one ethernet cards or devices on a system
- The composite hostid includes `HOSTID_ETHER`

—each ethernet address will be used to generate a composite hostid. A string of composite hostids is generated: one for each ethernet address on the system.

For example, when there are two ethernet cards and a composite hostid is specified that includes `HOSTID_ETHER`, **lc_hostid** returns a string that contains two composite hostids: one generated with each of the ethernet card addresses. For information about `lc_hostid`, see the *C/C++ Function Reference*.

When the licensing components use a hostid in a feature definition or `SERVER` line to authenticate the machine before using license rights, all the available composite hostids are checked until a match is found.

Using Multiple Composite Hostids with Feature Definition Lines

When there are multiple composite hostids generated for a machine because there are multiple ethernet cards, you can:

- **Specify all the hostids on a feature definition line**—The license rights defined in the feature definition line are available when any of the ethernet cards are on the machine.
- **Specify a single hostid on a feature definition line**—The license rights defined in the feature definition line are only available when that ethernet card is on the machine. You should choose this ethernet card with care to avoid unnecessary license breaks if an ethernet card is changed.

Using Multiple Composite Hostids with SERVER Lines

Only a single hostid can be specified on a `SERVER` line. You should choose one of the available composite hostids to put in the `SERVER` line. Choose the ethernet card that generates this composite hostid with care to avoid unnecessary license breaks if an ethernet card is changed.

Using Vendor-Defined Hostids

You can use the Flexible API to define your own hostid type.

Note ▪ *To discuss whether vendor-defined hostids are feasible for your application, contact Revenera Technical Support. FlexNet Publisher does not support using reserved keywords (such as `INTERNET`, `VSN`, `DISK_SERIAL_NUM`) or any name which resembles reserved keywords as a vendor-defined hostid label.*

The FlexNet Publisher toolkit contains a sample C source file, `examples\vendor_hostid\vendor_hostid.c`, in which a fixed vendor-defined hostid is set up. In this section, you can use this file to run through a procedure for setting up a vendor-defined hostid. In a real situation, you would not use a fixed vendor-defined hostid, but would define and call a function that returns the hostid that you want to use. A vendor-defined hostid can be used on a `SERVER` or `FEATURE` line of a license file.

Defining and Using a Vendor Hostid

You must define your hostid type (for this example, we are using `vendor_hostid.c`), then make sure that the vendor daemon, license generators, and your FlexEnabled application can recognize and use your hostid type. Only `lmcrypt` can generate licenses with vendor-defined hostids.

Task ***To define a vendor hostid and use it in licensing applications:***

1. Make a copy of your production toolkit. Follow these instructions using the files in the duplicate toolkit.
2. Copy `examples\vendor_hostid\vendor_hostid.c` to the `machind` directory.
3. View the file and find the `#define` statements. (See the file `machind\lmclient.h` for `HOSTID` and `LM_VENDOR_HOSTID` definitions.)

   ```
   #define VENDEF_ID_TYPE HOSTID_VENDOR+1
   #define VENDEF_ID_LABEL "VDH"
   #define VENDEF_ID "12345678"
   ```

   The `VENDEF_ID` assignment would not be needed in a real situation in which you had a function that returned your vendor-defined hostid. Close `vendor_hostid.c`.
4. Open `machind\lsvendor.c` in a text editor. At the beginning of the vendor initialization routine section, add a line defining **x_flexlm_newid** and modify the initial value of `ls_user_init1` from `0` to `x_flexlm_newid`.

   ```
   /* Vendor initialization routines */
   void x_flexlm_newid();
   void (*ls_user_init1)() = x_flexlm_newid;
   ```
5. Open `machind\lmcrypt.c` in a text editor. After the call to **lc_init**, add the following line:

   `x_flexlm_newid();`

   That section of the code should resemble:

   ```
   if (lc_init((LM_HANDLE *)0, VENDOR_NAME, &site_code, &lm_job))
   {
         lc_perror(lm_job, "lc_init failed");
         exit(-1);
   }
   x_flexlm_newid();
   ```
6. Open your FlexEnabled application source file in a text editor. In this example, `machind\lmflex.c` is used.

   a. Make the `lm_job` variable global by moving it before **main**.

   ```
   VENDORCODE code;
   LM_HANDLE *lm_job;

   void
   main()
   ```

   b. After the call to **lc_new_job**, add the following line:

   `x_flexlm_newid();`

   That section should resemble:

   ```
   if (lc_new_job(0, lc_new_job_arg2, &code, &lm_job))
   ```

```
{
    lc_perror(lm_job, "lc_new_job failed");
    exit(lc_get_errno(lm_job));
}
x_flexlm_newid();
```

7. Open `<platform_dir>\makefile` in a text editor. This example uses a Windows `makefile`.

   a. Add the vendor-defined hostid code to the list beginning of the list of `EXECS`. For this example, add `vendor_hostid.obj`.

   b. After the `$(DAEMON)` section, add a section to build `vendor_hostid.obj`. For example:

   ```
   vendor_hostid.obj :   $(SRCDIR)/vendor_hostid.c
           $(CC) $(CFLAGS) -I../h $(SRCDIR)\vendor_hostid.c
   ```

   c. Add `vendor_hostid.obj` to the link line for `$(DAEMON)`, `lmcrypt`, and `lmflex`. The following examples show where `vendor_hostid.obj` should be placed:

   ```
   $(DAEMON):   $(XTRAOBJS) $(DAEMONLIBS) lsvendor.obj $(SRCDIR)\lsserver.h $(LMNEW_OBJ)
         $(RC) -r -fo $(VENDORNAME).res $(VENDORNAME).rc
         $(LD) /subsystem:console /out:$(DAEMON) lsvendor.obj vendor_hostid.obj $(LMNEW_OBJ) \
               $(XTRAOBJS) $(DAEMONLIBS) $(CRT_LIB) $(XTRALIB1) $(DONGLELIB)
         $(ACTSTUB) $(VENDORNAME).res
         $(EMBED_MANIFEST)


   lmcrypt.exe:   $(SRCDIR)\lmcrypt.c \
         $(SRCDIR)\lmclient.h $(SRCDIR)\lm_code.h $(STATIC_CLIENTLIB)
         $(CC) $(CFLAGS) $(SRCDIR)\lmcrypt.c
         $(LD) /out:lmcrypt.exe lmcrypt.obj vendor_hostid.obj $(STATIC_CLIENTLIB) $(XTRALIB)
         $(ACTSTUB)
         $(EMBED_MANIFEST)


   lmflex.exe:   $(SRCDIR)/lmflex.c  $(LMNEW_OBJ) $(CLIENTLIB) lmstrip.exe
         $(CC) $(CFLAGS) $(SRCDIR)/lmflex.c
         $(LD) /out:lmflex.exe lmflex.obj vendor_hostid.obj \
               $(LMNEW_OBJ) $(CLIENTLIB) $(XTRALIB)
         if exist lmflex.obj del lmflex.obj
   ```

8. Rebuild the duplicate toolkit.

Test the Vendor-Defined Hostid

You will use the vendor daemon, license generator, and FlexEnabled application you just built to test a vendor-defined hostid.

Task ***To test a vendor-defined hostid:***

1. Create a license file that contains a `VENDOR` line with the vendor daemon you just built. Change the hostid on the `SERVER` line to:

   `VDH=12345678`

2. Run this license file through the newly built `lmcrypt`.

3. Create a license file containing this license rights and start your license server pointing to this file.
4. Run `lmflex`. You should be able to check out **f1**.
5. Exit `lmflex` and stop the license server.

Additional Steps for Production Use of a Vendor-Defined Hostid Type

To implement a real vendor-defined hostid type, you must write a function that can find the hostid that you want to use, then use that function's return value instead of the fixed value `VENDEF_ID` in `strncpy` in `vendor_hostid.c`:

```
if (idtype == VENDEF_ID_TYPE)
{
        h->type = VENDEF_ID_TYPE;
        strncpy(h->id.vendor, VENDEF_ID, MAX_HOSTID_LEN);
        h->id.vendor[MAX_HOSTID_LEN] = 0;
        return(h);
}
```

15

FlexNet Licensing Service

The FlexNet Licensing Service is an administrative proxy that runs on behalf of the user to perform system tasks where elevated privilege is required.

The FlexNet Licensing Service should be installed to enable the FlexEnabled application and the vendor daemon to access virtualization data (on Windows and Linux platforms only). The FlexNet Licensing Service regularly polls the system to update virtualization data. Therefore, the FlexNet Licensing Service runs continuously. The virtualization poll interval is not configurable. The FlexNet Licensing Service is optional for certificate-only applications that do not use virtualization features.

The FlexNet Licensing Service is also required to obtain the TPM hostid.

Producers can perform a pre-emptive check for the FlexNet Licensing Service in their own applications, via the new **lc_fnpservice_present** API, supported on Linux and Windows, which returns a positive integer if the FlexNet Licensing Service is installed. For more information on **lc_fnpservice_present**, see the *C/C++ Function Reference*, chapter *Flexible API*.

Installing the FlexNet Licensing Service on Windows

On Windows platforms, the FlexNet Licensing Service API is used to install and uninstall the service and register that an application requires the service. It is strongly recommended that the API is used within your product installer. The Windows operating system requires that a user have administrator privileges to install and uninstall services. Additionally, the installer must be run with elevated privileges.

Two types of functions are provided:

- **MSI-compatible Installer**—Use **fnpActSvcInstallForMSI** and **fnpActSvcUninstallForMSI**.
- **C/C++ Applications**—Use **fnpActSvcInstallWin** and **fnpActSvcUninstallWin** within an application. For remote installation, use **fnpActSvcRemoteInstallWin** within the application.

There are example files included in the licensing toolkit that demonstrate the use of these functions.

The recommended method of installing the FlexNet Licensing Service on Windows is from the MSI-compatible installer used to install the FlexEnabled application.

Task ***To install/uninstall the FlexNet Licensing Service using an MSI-based installer***

1. Use **fnpActSvcInstallForMSI** in the MSI to install the FlexNet Licensing Service and **fnpActSvcUninstallForMSI** to uninstall the service.
2. Set the required execution level to Administrator in the MSI.
3. Package the MSI as an executable.

   This is required because Windows will not allow an MSI to call out to a DLL (in this case `FNP_Act_Installer.dll`) that executes code requiring Administrative privileges.
4. Sign the MSI executable package.

   Windows requires executables that call out to DLLs that execute administrative-privilege code to be signed. Note that InstallShield 12 or later has settings in the Release Wizard that enable you to sign the generated installer executable.

Task ***To install/uninstall the FlexNet Licensing Service using an application or C-based installer***

1. Use **fnpActSvcInstallWin**, **fnpActSvcRemoteInstallWin** (for remote installation), **fnpActSvcUninstallWin**, and **fnpActSvcGetLastErrorWin** in your application.
2. Run the application/installer with elevated privileges.

   This requires the user to log on with administrator privilege and then use the 'run-as' option set to administrator. Alternatively, signing the executable with an Authenticode license file will ensure that elevated privileges are used.

Installing the FlexNet Licensing Service on Linux

On Linux, the FlexNet Licensing Service comprises the FlexNet Licensing Service executable and the FlexNet Licensing Service daemon. The daemon is launched by the FlexNet Licensing Service executable.

The following command-line options are available when running `install_fnp.sh`.

Table 15-1 ▪ `install_fnp.sh` options

Option	Description
`--cert`	Restricts the installation to only those components that are required for license file–based licensing.
`--nolsb`	Creates a symlink to the native loader to enable the FlexNet Publisher binaries to run without full lsb support on the target platform.

Table 15-1 ▪ `install_fnp.sh` options

Option	Description
`--nodaemon`	Prevents the installation script from automatically starting the FlexNet Licensing Service daemon. This can be useful when using systemd.
`--user`	Restrict Trusted Storage access to single-user or composite of user name and group name as `user[:group]`.

Task ***To install the FlexNet Licensing Service***

1. Run `install_fnp.sh --cert`.

   The `install_fnp.sh` shell script must be run with root privilege. The script configures the FlexNet Licensing Service executable to run as a root-privilege setuid process.

2. The FlexNet Licensing Service daemon needs to run continuously. Therefore, a further installation step is required to ensure it is started a boot time. Because the FlexNet Licensing Service daemon does not need to run with root privilege, it can be started by adding the following line to a nominated user's crontab:

   ```
   @reboot /usr/local/share/FNP/service64/<version>/FNPLicensingService -r 2>&1 >/tmp/fnpd.log
   ```

Notes

- For the i86_linux FlexNet Licensing Service daemon, the path is `/usr/local/share/FNP/service/<version>/FNPLicensingService`.
- The machine's administrator is free to redirect output as required.
- The machine's administrator may want to add an additional line to the crontab to run the above command at regular intervals to ensure that if the FlexNet Licensing Service daemon gets stopped for some unexpected reason, it will be restarted by the *crond*. The FlexNet Licensing Service daemon itself ensures there is never more than one instance running.
- Typical messages output on running the startup command are `Licensing Service daemon activated` and `Licensing Service daemon already active`.
- Rather than using cron, administrators may want to add a boot-script for the FlexNet Licensing Service daemon or optionally use systemd. The following commands are required:

  To start the FlexNet Licensing Service daemon

  ```
  /usr/local/share/FNP/service64/<version>/FNPLicensingService -r
  ```

  Note ▪ *The process ID of the daemon will be written to `/var/run/FNPLicensingService64.pid` (`/var/run/FNPLicensingService.pid` on 32-bit).*

  To stop the FlexNet Licensing Service daemon

  ```
  /usr/local/share/FNP/service64/<version>/FNPLicensingService -k
  ```

- When the Linux FlexNet Licensing Service daemon is not installed and an operation occurs that requires it, a descriptive error is displayed. For example:

  ```
  ./lmhostid -ptype VM -uuid
  ```

  results in

  ```
  lmhostid: The VM Host ID is not available. (-215,14704) - The FlexNet Licensing Service does not appear to be running.
  ```

- FlexNet Publisher no longer supports the `--nolsb` option from FlexNet Publisher release R3, 2023, onwards.

Installing the FlexNet Licensing Service on OS X

On OS X, a shell script is used to configure the FlexNet Licensing Service executable to run as a root-privilege setuid process.

Task ***To install the FlexNet Licensing Service***

Run `install_fnp.sh`.

Sample Applications

The Windows toolkit contains two sample programs that install and uninstall the FlexNet Licensing Service using **fnpActSvcInstallWin**, and **fnpActSvcUninstallWin** functions of the FlexNet Licensing Service API:

- `installanchorservice.exe`
- `uninstallanchorservice.exe`

These files are located in the `<platform_dir>` directory of the toolkit. The source code is available in `\examples\anchor_service`. These executables must be run as an Administrative user.

Note ▪ *The* `installanchorservice` *and* `uninstallanchorservice` *executables are intended for testing only.*

installanchorservice.exe

The sample executable `installanchorservice.exe` installs the FlexNet Licensing Service and registers the FlexEnabled application as requiring the service. To run this executable, ensure that the FlexNet Licensing Service installer component, `FNP_Act_Installer.dll`, and the executable are in the same folder.

Usage

`installanchorservice` *`publisher_name product_name`*

Where:

Table 15-2 ▪ installanchorservice and uninstallanchorservice Inputs

Input	Description
`publisher_name`	The name you assign to yourself as a publisher. This string can be anything meaningful to you and does not need to match your vendor name as registered with Revenera and cannot contain the comma character. This value and the *product_name* value must uniquely identify the FlexEnabled application.
`product_name`	The name of the application. This name is used to identify that the application is using activation functionality. This string cannot contain the comma character. This value and the *publisher_name* value must uniquely identify the FlexEnabled application.

uninstallanchorservice.exe

The sample executable `uninstallanchorservice.exe` removes the specified FlexEnabled application from the registry of all applications requiring the FlexNet Licensing Service. If no other application is registered to the FlexNet Licensing Service, the executable removes the FlexNet Licensing Service from the system. To run this executable, ensure that the FlexNet Licensing Service installer component, `FNP_Act_Installer.dll`, and the executable are in the same folder.

Usage

`uninstallanchorservice` *`publisher_name product_name`*

The command line arguments are described in Table 15-2.

16

Software Publisher Utility Programs

The `lmcrypt` utility can be used to generate licenses for your users. The source to `lmcrypt` is in the `machind` directory. You can use the source as the basis for your own license generation utility.

The `lmcrypt` utility generates the same signatures on all FlexNet Publisher–supported platforms for all FlexNet Publisher versions, allowing you to create licenses for any supported platform on any other supported platform.

lmcrypt

When you know the format of the `FEATURE` or `INCREMENT` lines that you need for your license model, create a license file and specify the signature value as `SIGN=0`. The `lmcrypt` utility replaces the zero (0) with the generated the signature. The signed license file is ready to ship to the user. To use `lmcrypt`, you need to either understand the license file syntax or use an example license file. Examples are available in the `examples/licenses` directory.

Syntax

```
lmcrypt [files][-i infile] [-o outfile] [-maxlen n] [-e errfile]
       [-verfmt { 2 | 3 | 4 | 5 | 5_1 | 6 | 6_1 | 7 | 7_1 | 8 }]
        [-decimal] [-help]
```

If no input file is specified, or if specified as “-” or stdin, standard input is used. If no output file is specified, or if specified as “-” or stdout, standard output is used. `files` are read and written back in place. If no file arguments are specified, `lmcrypt` reads from stdin and writes to stdout. All signatures are recomputed. `lmcrypt` works only on lines that have a vendor name matching the vendor’s daemon name.

If this is not possible, an error is produced, and the affected license line is left unaltered.

The maximum line length is controlled with the `-maxlen` argument. A value of 50 is commonly used to make shorter lines less subject to mailers inserting new lines. Licenses can be generated to be compatible with older versions by using the `-verfmt` argument.

Task ***To use lmcrypt:***

1. Copy an existing good file to another name such as `newlicense`.
2. Edit `newlicense` and make any desired changes, such as changing the feature name, the number of licenses, or adding new features, and save the file.
3. Change the existing signature on each `FEATURE` line to `SIGN=0`.
4. Type the following:

   `lmcrypt` ***`newlicense`***

   `newlicense` is then filled with correct signatures and is usable by a user. Comments are passed through unaltered.

See `lmcrypt.c` in the `machind` directory.

Error Returns

Errors are printed to stderr, or as specified with `-e`. Most errors prevent generation of signatures and the text is output unchanged from the input. An example of error reporting: If `-e` is not used, the error messages also appear at the top of the output file.

Input

```
FEATURE f1 demo 1.a50 01-jan-2005 uncounted HOSTID=08002b32b161 \
        SIGN=0
```

Error Reported

```
stdin:line 1:Bad version number - must be floating point number, with no letter.
```

17

License Administration

The following sections are described in this chapter:

- Options File
- License Administration Tools
- Efficient Reservation

Options File

License Administrators can customize software usage using an administrative options file. An options file enables the license administrator to reserve licenses for specified users or groups of users, to allow or disallow software usage to certain people, to set software timeouts, and to start a report log that can be read by FlexNet Manager.

The options file can be specified on the `VENDOR` line in the license file as follows:

`VENDOR` *`vendor`* `[`*`vendor_daemon_path`*`] [`*`options_file_path`*`]`

If the path to the vendor daemon is not specified on the `VENDOR` line, the path to the options file must be preceded by `options=`, as in:

`VENDOR` *`vendor`* `[options=`*`options_file_path`*`]`

The options file does not need to be specified if the options file is both:

- Named *`vendor`*`.opt`
- Located in the same directory as the license file

The following options keywords are available:

- `AUTOMATIC_REREAD`
- `BORROW_LOWWATER`
- `DAEMON_SELECT_TIMEOUT`
- `DEBUGLOG`

- EXCLUDE
- EXCLUDE_BORROW
- EXCLUDEALL
- FQDN_MATCHING
- GROUP
- GROUPCASEINSENSITIVE
- HOST_GROUP
- INCLUDE
- INCLUDE_BORROW
- INCLUDEALL
- LINGER
- MAX
- MAX_BORROW_HOURS
- MAX_CONNECTIONS
- MAX_OVERDRAFT
- NOLOG
- REPORTLOG
- RESERVE
- TIMEOUT
- TIMEOUTALL

The `TIMEOUT` option allows an idle license to return to the free pool for use by another user. `TIMEOUT` requires that the application not send heartbeats when it is idle. Therefore, for the `TIMEOUT` option to work, the application must support it. If timers are disabled, then the application must regularly call a heartbeat function when active and not call a heartbeat function when inactive. If a heartbeat function is not called regularly when active, the `TIMEOUT` option can cause the FlexEnabled application to lose its license. If the application uses timers (`LM_MANUAL_HEARTBEAT` not set), a `TIMEOUT` specification is ineffective.

For complete syntax and descriptions of options, see the *License Administration Guide*.

License Administration Tools

Syntax and detailed descriptions of `lmutil` commands can be found in the *License Administration Guide*.

On UNIX, all license administration tools are contained in the single executable `lmutil`. `lmutil` behavior is determined by its first argument, or its `argv[0]` name. `lmutil` renamed to `lmstat` behaves the same as `lmutil lmstat`. When you installed FlexNet Publisher, the installation created hard links from `lmutil` to each `lmutil` program names listed. You should also create these hard links when your software is installed on the system.

On Windows, `lmutil.exe`, a command-line program similar to `lmutil` on UNIX, is provided. A utility is invoked with `lmutil.exe function`, where `function` is `lmstat`, `lmdiag`, and so on. A tool interface called `LMTOOLS` is also provided on Windows with the same functionality as `lmutil.exe`.

`lmutil` and `lmutil.exe` contain the following utilities:

- `lmborrow`—Supports license borrowing via `BORROW`.
- `lmdiag`—Diagnoses license checkout problems.
- `lmdown`—In a three-server set-up, on running `lmdown -c license_file_list`, users are allowed to select a server to be shut down, while allowing other two servers in the triad to be running. This helps the user to bring down one server in the triad for maintenance temporarily. Note that using `lmdown` with the `-c` option specifying the license file configured for three-server redundancy is mandatory to see the desired behavior.
- `lmhostid`—Reports the hostid of a system.
- `lminstall`—Converts license files between different formats.
- lmlicvalidator—Allows a user to pre-validate a certificate license file (from a remote client) against a license server.
- `lmnewlog`—Moves existing report log information to a new file name and starts a new report log file with existing file name.
- `lmobfslog`—Processes the debug log file to generate the obfuscated usernames.
- `lmpath`—Allows users direct control over license file settings.
- `lmremove`—Releases a hung license to the pool of free licenses.
- `lmreporter`—Allows users to view feature usage, peaks, and denials.
- `lmreread`—Causes the license server to reread the license file and options file and start any new vendor daemons.
- `lmstat`—Displays the status of a license server.
- `lmswitch`—Switches debug log for this vendor daemon to new file name.
- `lmswitchr`—Switches the report log to a new file name.
- `lmver`—Reports the version of a library or binary file.

Note ▪ *Note the following information:*

- *If an application is active when a license is removed with `Lmremove`, it simply checks out the license again (assuming the applications FlexNet Publisher Licensing Toolkit timers are enabled or a heartbeat function is called). Therefore, `Lmremove` cannot normally be used to "steal" licenses.*
- *Because an end-user options file can be reread by a vendor daemon, a delay has been added to the activation of any `INCLUDE` lines in the end-user options file that affect `USER_BASED` or `HOST_BASED` features. The default delay is 12 hours, but can be reset in `machind/Lsvendor.c`.*

All utilities take arguments described in Table 17-1.

Table 17-1 ▪ License Server Utility Arguments

Argument	Description
-c *license_file_path*	Operate on this license file.
-v	Print version and exit.
-verbose	Prints longer description of all errors found. The output from the utilities may be harder to read with this option, but is useful for diagnostics.

Efficient Reservation

The feature licenses reserved with RESERVE line in the Options file will stay reserved almost for lifetime of the vendor daemon process. In case of these reserved licenses are no longer required to be part of reserve pool, the License Administrator can change the Options file to modify the reserved licenses with following process:

1. Access the License server machine.
2. Edit the Options file situated at local license server machine as required.
3. Wait for next re-read or re-start of the license server to re-process the latest license file and Options file.

Hence, re-read and re-start of the license server would be performance intensive process. Following the above process for editing the reservation policy is appropriate if Options file is not changed frequently. If license in reserve pool did not use effectively, then it may introduce inefficiency. License Administrator should be vigilant to understand the significance of each reserved pool and update it accordingly. Consider the scenario where certain job depends on multiple feature for its completion and at different time durations. For executing that job with reserving licenses with Options file may be inefficient process.

The licenses are no longer required once job has been completed. It is an additional overhead on license server and License Administrator to keep changing the reservation policy once the job has completed. There might be multiple jobs running at different intervals.

In majority of the instances, licenses are not required to be reserved for longer duration and not required to be in reserve pool anymore, once the licenses are checked out. Efficient reservation manages effectively the licenses that are to be reserved for shorter duration. License server can apply or revoke the reservation at runtime without much performance impact. FlexNet publisher provided API **lc_reservation_create** and **lc_reservation_delete** or **lmreserve.exe** are designed to accomplish these tasks. The authenticated user can remotely connect with the license server and efficiently apply the reservation on certain features for the job. Reservation can be revoked once job has been completed. The reserved licenses will be moved back to the floating pool and open for all authenticated users. Efficient reservation is based on RESERVE line in the Options file and exhibits similar behavior. Some of the important highlights:

- Efficient reservation can be applied and activated altogether without running expensive server re-read functionality.
- Single reservation request can have multiple RESERVE lines for different features. If any of the RESERVE line failed to acquire the reservation, then license server rejects whole reservation request. It is expected that job

may require multiple feature for completion. If any of the feature is not available, then job may not execute efficiently.

For example, the license file contains two feature lines as below:

```
INCREMENT f1 qacompb 1.0 permanent 3 SIGN
```

```
INCREMENT f2 qacompb 2.0 permanent 2 SIGN=
```

And the reservation lines are:

```
RESERVE 3 f1 PROJECT abc
```

```
RESERVE 3 f2 PROJECT abc
```

The whole efficient reservation is rejected as f2 does not contain sufficient number of licenses in its floating pool to reserve.

- Efficient reservation is only applied on licenses in floating pool. The request will be rejected if sufficient licenses are not in the floating pool.
- Efficient reservation does not have any dependency on the Options file and can be applied without it. The only exception is if reservation is applied for the GROUP. In that case, it is expected that the Options file enables that GROUP in the vendor daemon.

  For example, Efficient reservation with below mentioned RESERVE line expects that the GROUP is already created with Options file:

  ```
  REERVE 5 f1 GROUP eng
  ```

- RESERVE line reserves same number of licenses from all the available license pool of the same feature if just feature name is mentioned. The request considered failed if any license pool does not have sufficient number of license. Use key:value pair to reserve licenses from specific feature license pool.

  For example,

  License file contains two pools of feature f1:

  ```
  INCREMENT f1 qacompb 1.0 permanent 3 SIGN
  ```

  ```
  INCREMENT f1 qacompb 2.0 permanent 4 SIGN=
  ```

  Following RESERVE line in efficient reservation will reserve 2 licenses each from both the pools:

  ```
  RESERVE 2 f1 PROJECT abc
  ```

  Reservation request will be unsuccessful as 4 floating licenses are expected from both the pools:

  ```
  RESERVE 4 f1 PROJECT abc
  ```

  Reservation request will be successful with key value pair. It will reserve licenses only from pool with version 2.0:

  ```
  RESERVE 4 f1:2.0 PROJECT abc
  ```

- License server generates unique reservation ID for each successful efficient reservation for future references. Same reservation ID is passed back to server to remove the reservation and move licenses back to floating pool.

- In Efficient reservation, licenses can be moved out of reserved pool to floating pool once job is completed. Efficient reservation can follow the policy to move licenses out of reserve pool promptly once purpose solved. The lifetime of the reservation can be one of following:
  - Lifetime of the vendor daemon process on which efficient reservation is applied. Efficient reservation will no longer be available, if license server exits.
  - Intentionally removed with the Producer provided tool based on `lc_reservation_delete`.
  - The reservation will be removed automatically once duration has expired.
- Checked out license will stay in checked out state even if license is moved from reserve pool to floating pool.

Reserve License Using lmreserve.exe Utility

Alternatively, you can reserve/delete the license(s) online by using `lmreserve.exe` utility. To reserve/delete the license(s) online, create the lmreserve.exe utility after building the toolkit.

Creating lmreserve.exe utility

Ensure to create the `lmreserve.exe` utility after building the toolkit to do online reservation.

Task ***To create the lmreserve.exe utility,***

1. Copy `lmreserve.c` file from `examples\online_reservation` to the machind directory before building the toolkit.
2. Open makefile in the text editor. For example, In Windows, the location of the makefile is `<platform_dir>\makefile.`
3. In the makefile,
   1. add the online reservation code to the list of EXECS.
   2. add `lmreserve.exe` after the lmflex.exe to build the `lmreserve.exe` as below:
      ```
      lmreserve.exe: $(SRCDIR)\lmreserve.c $(LMNEW_OBJ_CLIENT) \
              $(CLIENTLIB)
          $(CC) $(CFLAGS) $(SRCDIR)\lmreserve.c
          $(LD) /out:lmreserve.exe lmreserve.obj $(LMNEW_OBJ_CLIENT) $(CLIENTLIB) $(XTRALIB)
      $(ACTSTUB)
          $(EMBED_MANIFEST)
          if exist lmreserve.obj del lmreserve.obj
      ```
4. The `lmreserve.exe` is created in the platform directory after rebuilding the toolkit.

Reserving the License Online

The license(s) can be reserved in the server using the `lmreserve.exe` utility. Ensure the The `lmreserve.exe` is situated in the platform directory.

Usage

`lmreserve -reserve -duration <Time> <Reservation lines>`

Where:

Table 17-2 ▪ lmreserve Input

Input	Description
<Time>	Duration in seconds, will set the duration for which reservation will be active on the server.
<Reservation lines>	Feature reservation request. It can have maximum of 50 multiple reserve lines for different features. The reserve lines are separated by comma.

Deleting the Reserved License

The license(s) that is reserved already can be deleted from the server using the `lmreserve.exe` utility.

Usage

`lmreserve -delete <reservation id>`

Where:

Table 17-3 ▪ lmreserve Delete Input

Input	Description
<reservation id>	A reservation id, received while reserving the license.

18

Internationalization Support

Internationalization is an important part of the development process for products that require multilingual support. This chapter describes the available internationalization support for the areas of FlexNet Publisher involved with end-user interaction.

- License Files
- Options Files
- Client Library
- Debug Log Output
- Report Log Output
- License Server Utilities

License Files

A license file contains several types of text elements. This section describes each text element, along with its considerations for internationalization support.

- **Comments**—Comment lines can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC.
- **Contents of** `NOTICE`, `ISSUER`, `VENDOR_STRING`, `SN`, `asset_info`, `dist_info`, `user_info`, **and** `vendor_info` **fields**—The contents of these fields can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC. Note that FlexNet Publisher does no codepage conversion on these fields. Therefore, if the FlexEnabled application intends to access one or more of these fields at runtime, it must either accept the contents as being represented in the codepage you use in the license file or it must convert the contents to the codepage it requires. Also, if you intend the license file to be read by the user, the text editor must accept the license file as being represented in the codepage used in the license file.

- **Feature and package names**—Feature and package names can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC.

  Note ▪ *Keep the following restrictions in mind:*

  - *FlexNet Publisher does not perform codepage conversion on the names. Therefore, in the case of feature names, the FlexEnabled application must check out the feature name using the same codepage used in the license file in order to authenticate the license.*
  - *Feature names can contain only letters, numbers, and underscore characters. Letters in the feature name are case insensitive by default. If case sensitivity is desired, see the "LM_A_LICENSE_CASE_SENSITIVE" section in the C/C++ Function Reference.*

- **Host, display, and user names**—Host, display, and user names can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC. Because there is inconsistent support for non-ASCII host, display, and user names in operating systems and applications, non-English system administrators rarely use non-ASCII characters in these names.

- **File names (Windows only)**—File names can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC. Because there is inconsistent support for non-ASCII file names in operating systems and applications, non-English system administrators rarely use non-ASCII characters in these names.

- **Vendor names**—Revenera registers vendor names that contain ASCII characters only. These names are not visible to the user of the FlexEnabled application and they usually represent an abbreviated identifier for a full company or division name.

- **Keywords**— keywords in a license file must be represented with ASCII characters only. This does not allow them to be translated.

- **Dates**—Expiration, start, and issued dates are represented in a format common in the United States. These dates are not visible to the user of the FlexEnabled application.

UTF-8 Encoded Files

When saving a file using UTF-8 encoding, some text editors (for example Windows Notepad) insert special bytes at the beginning of the file to help those programs that read the file understand that the file contains UTF-8 encoded characters. The license server, the client library, and `lmcrypt` treat the first line of a license file written in this way as a comment line; that is, the line is ignored, even if it contains a syntactically valid license line. To support such files as license files, put a comment line as the first line of the license file. Alternatively, leave the first line blank.

Implementation Considerations

UTF-8 Encoding on Windows Platforms

Windows-based license servers and FlexEnabled applications assume the license files they read are UTF-8 encoded. For Windows-based unserved applications or applications served by a Windows-based license server, you must supply a UTF-8 encoded license file.

Case Folding

By default, FlexNet Publisher performs case folding on license file text. The case-folding algorithms support the ASCII and ISO 8859-1 codepages, only. If your license file contains characters from codepages other than these, then you must turn off case folding by enabling case sensitivity in your licensing implementation.

This is accomplished as follows:

- In the FlexEnabled application, before the call to **lc_checkout**, set the `LM_A_LICENSE_CASE_SENSITIVE` attribute as follows:

  ```
  lc_set_attr(lc_job, LM_A_LICENSE_CASE_SENSITIVE, (LM_A_VAL_TYPE) 1);
  ```

- In `lsvendor.c`, set the `ls_a_license_case_sensitive` attribute as follows:

  ```
  int ls_a_license_case_sensitive = 1;
  ```

- In the license generator, before the call to **lc_cryptstr**, set the `LM_A_LICENSE_CASE_SENSITIVE` attribute as follows:

  ```
  lc_set_attr(lc_job, LM_A_LICENSE_CASE_SENSITIVE, (LM_A_VAL_TYPE) 1);
  ```

- For publisher-defined hostids, set the `case_sensitive` field of the `LM_VENDOR_HOSTID` structure to 1.

See Also
`LM_A_LICENSE_CASE_SENSITIVE` in the *C/C++ Function Reference*
ls_a_license_case_sensitive

Options Files

The options file contains several types of text elements:

- **Group names**—Group names can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC.
- **Comments, keywords, feature names, host names, display names, user names, ad file names**—See the discussion regarding these elements in the License Files.

UTF-8 Encoded Files

When saving a file using UTF-8 encoding, some text editors (for example Windows Notepad) insert special bytes at the beginning of the file to help those programs that read the file understand that the file contains UTF-8 encoded characters. The license server, the client library, and `lmcrypt` treat the first line of a license file written in this way as a comment line; that is, the line is ignored, even if it contains a syntactically valid license line. To support such files as license files, put a comment line as the first line of the license file. Alternatively, leave the first line blank.

Client Library

Error Messages

The Flexible API provides the function **lc_err_info**, which returns a numeric status code to the application upon an error in one of the other Flexible API functions. Each error code has three levels of error text associated with it in varying degrees of length.The English version of the error text is provided by the toolkit in the following files:

- **short**—located in `machind/lmerrors.h`
- **long**—located in `machind/lm_lerr.h`
- **context**—located in `machind/lcontext.h`

You translate the text in these files into any language, using any codepage, including Unicode. You can then display the translated text in your application in any way you want. The *C/C++ Function Reference* provides a short-error message reference, listing each status code, along with its corresponding English text.

For convenience when developing applications for English-speaking users, FlexNet Publisher provides the function **lc_errstring**, which, instead of returning the numeric status codes, returns the English text associated with the status code.

Environment Variables

When setting environment variables that are read by FlexNet Publisher, such as `LM_LICENSE_FILE_DEFAULT`, make sure they are set with UTF-8 encoded strings.

Diagnostic Messages

By default, routines in the client library do not generate diagnostic messages. However, if the `FLEXLM_DIAGNOSTICS` environment variable is set to a value of 1, 2 or 3, diagnostic messages are written to a log file named `flexpid.log` in the current directory (Windows platforms) or written to the standard error stream of the application (UNIX platforms).

These messages are intended used to diagnose technical problems that occur. During the debugging process, the diagnostic text is usually communicated among several organizations. Therefore, for consistency and accuracy of support, the text in these messages is in English, as is customary in other software applications. See the format of these messages in the *License Administration Guide*.

By, default, when the user sets the `FLEXLM_DIAGNOSTICS` environment variable, diagnostic messages are generated by the client library. However, you can prevent the `FLEXLM_DIAGNOSTICS` environment variable from having any effect via the `LM_A_DIAGS_ENABLED` attribute in your FlexEnabled application. To disable the messages, set the attribute as follows:

```
lc_set_attr(lc_job, LM_A_DIAGS_ENABLED, (LM_A_VAL_TYPE) 0);
```

Debug Log Output

Debug log output is available to diagnose technical problems. During the debugging process, the text in the debug log is usually communicated among several organizations. Therefore, for consistency and accuracy of support, any diagnostic text is in English, as is customary in other software applications. See the format of this file in the appendix entitled “The Debug Log File” in the *License Administration Guide*.

Windows Event Logging

The software publishers can enable the license server to record a subset of the debug log messages to the Microsoft Windows event-logging service. Messages are organized in standard Microsoft Windows message file format and can be localized. See “Localizing the Event-Logging Message Files” in the *Development Environment Guide* for further details.

Report Log Output

Report log output is intended for FlexNet Manager users. The text in this file cannot be read by humans. It can be read by FlexNet Manager only, is totally derived from the text in a license file, and, thus, supports the same codepages as described in License Files.

License Server Utilities

These utilities (`lmutil`, and others) are used by the license administrator to help manage license servers. They have hard-coded English text in their output and error messages.

Due to limitations with the DOS command-line processor, all utilities interpret their arguments as being composed with wide characters. For example, UTF-8 encoded characters cannot appear in any command line argument.

lmreporter

The lmreporter is an optional user interface that allows to view the feature usage, peaks, and denials. The lmreporter utility uses the debug logs, where the server events are captured, created by the vendor daemon. This functionality must be explicitly enabled by the vendor.

To enable the lmreporter, see the Debug Log’s Verbose Information for lmreporter.

For more information on usage of the lmreporter, see the *Using License Administration Tools* chapter in the *License Administration Guide*.

Note ▪ *The lmreporter tool is supported only on the Windows and Linux platforms.*

LMTOOLS

The `lmtools.exe` application is a Windows-only graphical version of the common functions found in the utilities. As such, this application is used by the license administrator to help manage license server.

The user interface of `lmtools.exe` is in English. However, because it is a standard Windows application, the user interface can be translated into any language: Edit the resources in `lmtools.exe` using standard Windows resource editing tools.

The `lmtools.exe` application sometimes directly displays the output in a window. Therefore, if the utility displays some English text, the corresponding window in `lmtools.exe` will display that same English text.

License Finder Runtime Graphical User Interface

For Windows applications, when a license is denied, FlexNet Publisher displays a dialog box by default, that gives the user the ability to direct the application to either a license file or a license server in order to locate the license. It does this under the assumption that the user has a license available, but did not correctly configure their application to refer to the license file or license server that contains the license.

The text in this dialog box appears in English. Even though it is a standard Windows interface, software publishers cannot translate the user interface of this dialog box because the resources for the dialog box are not bound to the client library for access by a resource editor.

To prevent display of the License Finder dialog box, set the `LM_A_PROMPT_FOR_FILE` attribute in your application as follows:

```
lc_set_attr(lc_job, LM_A_PROMPT_FOR_FILE, (LM_A_VAL_TYPE) 0);
```

19

Time Zone Licensing

Time zone licensing enables you to bind license usage to specified geographic locations, which are identified in the license certificate using time zones. This makes it possible for you to vary license model pricing based on the geographic location in which your software is used and to minimize the impact of corporate networks, which may enable clients to access a license server from anywhere in the world.

- Understanding Time Zone Licensing Basics
- Implementing Time Zone–Based Licensing
- Examples
- Testing Time Zone Licensing
- Limitations and Interactions with Existing Licensing Functions

Understanding Time Zone Licensing Basics

The TZ keyword specifies the time zones in which the FlexEnabled feature is licensed to run. The time zones are specified either as values relative to Greenwich Mean Time (GMT) or as the time zone of the license server (SERVERTZ).

At run time, when a license checkout is requested, the time and time zone of the client machine is read and compared to the time zone specified in the license file. If the time zone read from the client machine matches the time zone in the license file, the checkout proceeds as usual. If, however, there is a mismatch between the time zone read from the client machine and the time zone specified in the license file, the checkout request is denied.

Note the following when using the `TZ` keyword:

- 15 minutes of clock drift is allowed when validating the time and time zone of the client with those of the license server.
- 60 minutes is allowed for Daylight Saving Time (DST) when validating the time and time zone of the client with those of the license server.
- If a license is borrowed with `SERVERTZ`, license rights are not bound to any time zone when the borrowed license is used.

- If SERVERTZ is specified for an unserved, node-locked license, license rights are not bound to any time zone.

Important ▪ *Time zone licensing is intended for use with customers with whom you have a trusted relationship. There is no secure programmatic restriction with time zone licensing functionality: If an end user manually changes the time zone to match the time zone specified in the license certificate, a license can be consumed.*

Note ▪ *The Flexible API attribute,* LM_A_TZ_OVERRIDE*, and the vendor variable, ls_allow_tz_override, are provided for use in testing time zone licensing. See FlexNet Publisher's C/C++ Function Reference for more information about these and other time zone–related attributes.*

Implementing Time Zone–Based Licensing

Follow these steps to apply time zone licensing to your product.

Task ***To implement time zone–based licensing:***

1. Re-link the FlexEnabled application with the new library components.
2. Re-link the vendor daemon with the new library components.
3. Specify the geographic location of the computer system on which the FlexEnabled application should be running by using the new keyword TZ in the license file on the FEATURE or INCREMENT line.
4. Encrypt the updated license files.
5. Distribute the updated license files and vendor daemon to your customers.

Tip ▪ *FlexNet Publisher provides an API attribute and a vendor variable to help you test time zone–based licensing:*

- *Use the new Flexible API attribute, LM_A_TZ_OVERRIDE, in* `lc_set_attr()` *to override the client time zone.*
- *Use the new vendor variable, ls_allow_tz_override to override the server time zone or time using the environment variables.*

See Testing Time Zone Licensing for more information.

Examples

This section provides syntax examples for various time zone licensing use cases.

- Specifying Single and Multiple Time Zones
- Specifying Time Zone Ranges
- Specifying Time Zone Ranges and Multiple Time Zones
- Specifying Half and Quarter Time Zones

- Accounting for Daylight Saving Time
- Using the Time Zone of the License Server
- Using TZ with a Fully Qualified Domain Name (FQDN)
- Including Cross-Version Signatures

Specifying Single and Multiple Time Zones

To specify multiple time zones, use a space as a delimiter and enclose the values in quotes.

```
FEATURE f1_single demo 1.0 permanent 4 TZ=-08 SIGN=0
FEATURE f2_multiple demo 1.0 permanent 4 TZ="+05 +01" SIGN=0
```

The feature f1_single can be used only on clients that reside in the time zone GMT–08:00, while the feature f2_multiple can be used on clients that reside in time zone GMT+05:00 or on clients that reside in time zone GMT+01:00.

Specifying Time Zone Ranges

The keyword allows time zones ranges, including ranges that span across GMT.

```
FEATURE f1_range demo 1.0 permanent 4 TZ="-05:-08" SIGN=0
FEATURE f2_range demo 1.0 permanent 4 TZ="–05:–00 +00:+01" SIGN=0
```

The feature f1_range can be used on clients that reside in time zones that fall in the range GMT–05:00 through GMT–08:00, inclusive. The feature f2_range can be used on clients that reside in time zones that fall in the range from GMT–05:00 through GMT+01:00, inclusive.

Specifying Time Zone Ranges and Multiple Time Zones

You can also specify ranges, along with single or multiple time zones.

```
FEATURE f1_range_multi demo 1.0 permanent 4 TZ="-08 +00:+02 +05.30:+08.30" SIGN=0
```

The feature f1_range_multi can be used on clients that reside in time zones that fall in the ranges GMT+00:00 through GMT+02:00 and GMT+5:30 and GMT+08:30, inclusive, and on clients that reside in the time zone GMT–08:00.

Specifying Half and Quarter Time Zones

To specify half and quarter time zones when applicable, use the notation .30 for half and .45 for quarter.

```
FEATURE f1_half demo 1.0 permanent 4 TZ=+05.30 SIGN=0
FEATURE f2_quarter demo 1.0 permanent 4 TZ=+05.45 SIGN=0
```

The feature f1_half can be used on clients that reside in time zone GMT+05:30, which is India, while the feature f2_quarter can be used on clients that reside in time zone GMT+05:45, which is Nepal.

Accounting for Daylight Saving Time

To account for time zones in which Daylight Saving Time (DST) is observed, provide an offset of an additional time zone while specifying a TZ value in the license file. For example, to allow license usage for clients that reside in GMT–08:00, include an additional offset hour to handle DST:

```
FEATURE f1_dst demo 1.0 permanent 4 TZ="-08 -07" SIGN=0
```

Using the Time Zone of the License Server

As an alternative to coding specific time zones, use TZ=SERVERTZ to require that the time zone on the client making the checkout request matches the time zone on the license server.

```
FEATURE f1_server demo 1.0 permanent 4 TZ=SERVERTZ SIGN=0
```

Using TZ with a Fully Qualified Domain Name (FQDN)

To provide more granularity, use the license file keyword TZ in combination with the fully qualified domain name (FQDN). With FQDN, the host name of the client making the checkout request must be in the domain specified using the special hostid HOSTNAME. For example:

```
FEATURE f1_fqdn demo 1.0 permanent 4 TZ="-08:-05" HOSTID=HOSTNAME= *.revenera.com SIGN=0
```

The feature f1_fqdn can be used on machines that meet *both* of the following criteria:

- Reside in the range of time zones from GMT–08:00 to GMT–05:00, inclusive
- Are in the domain *.revenera.com

For more information about using FQDN, see the section describing the Flexible API attribute LM_A_USE_FQDN in the *C/C++ Function Reference*.

Including Cross-Version Signatures

Keywords such as TZ are authenticated by the license signature. To enable deployed applications (created using an earlier version of FlexNet Publisher) to recognize a signature that uses the TZ keyword, include a cross-version signature for the feature line. This approach eliminates the need to distribute separate license files for use with FlexEnabled applications built using earlier versions of FlexNet Publisher.

The version identifier V11.6_SIGN creates version signatures that can be authenticated by FlexEnabled applications built using FlexNet Publisher 11.5 to 11.6.1.

For example, to ensure that a license can be served to FlexEnabled applications built using a new version of the license server, together with a legacy 11.6.1 client, use a feature definition line similar to the following:

```
INCREMENT f1 demo 1.0 permanent 5 TZ=SERVERTZ SIGN=0 V11.6_SIGN=0
```

At encryption time, two signatures are created for the feature definition line:

- The SIGN signature is served to FlexEnabled applications containing the current client libraries.
- The V11.6_SIGN signature is served to FlexEnabled applications containing the 11.6.1 client libraries.

Note ▪ *For more information about using time zone licensing in FlexEnabled applications built with prior versions of FlexNet Publisher, see Cross-Version License Files.*

Testing Time Zone Licensing

FlexNet Publisher provides the following attributes, variables, and environment variables to facilitate testing of time zone licensing:

- Flexible API attribute: `LM_A_TZ_OVERRIDE` works with the `lc_set_attr` function to override the client time zone or time zone and time settings. (See the *C/C++ Function Reference* for details about this attribute and related error codes.)
- Vendor daemon variable: `ls_allow_tz_override` operates in your test environment to enable the vendor daemon to read values you set in the `LM_SET_SERVER_TIMEZONE` and `LM_SET_SERVER_TIME` environment variables.
- Environment variables: `LM_SET_SERVER_TIMEZONE` and `LM_SET_SERVER_TIME` carry the override values you set for the license server time zone and time, respectively. These variables are not used unless `ls_allow_tz_override` is set to 1.

Environment Variable Details

Set these environment variables on the machine that hosts the license server.

Table 19-1 ▪

Environment Variable	Description
LM_SET_SERVER_TIMEZONE	For testing purposes, sets the time zone of the license server to the value specified. The time zone is specified relative to Greenwich Mean Time (GMT). For example, to override the license server to be in the Eastern time zone (GMT–05.00), set the value of `LM_SET_SERVER_TIMEZONE to -05`. This environment variable can be used alone or in combination with `LM_SET_SERVER_TIME`.
LM_SET_SERVER_TIME	For testing purposes, sets the time of the license server to the value specified. The syntax is *`hh:mm`*, based on a 24-hour clock. For example, to override the license server time to be 5:45 P.M., set the value of `LM_SET_SERVER_TIME to 17:45`. ***Important*** ▪ *This environment variable can be used only in combination with `LM_SET_SERVER_TIMEZONE`.*

Environment Variable Examples

The following examples show how to use the LM_SET_SERVER_TIMEZONE and LM_SET_SERVER_TIME environment variables for time zone licensing tests.

Table 19-2 ▪

Example	Syntax for Setting Environment Variables	Behavior
Override server time zone	Set LM_SET_SERVER_TIMEZONE to +05.30 (LM_SET_SERVER_TIME is not set.)	Server time zone would be overridden with +05.30. Server time is not overridden; server machine time would be taken into account.
Override server time zone and time	Set LM_SET_SERVER_TIMEZONE to +05.30 Set LM_SET_SERVER_TIME to 20:33	Server time zone would be overridden with +05.30. Server time would be overridden with 8:33 P.M.

Note ▪ *General information about working with environment variables on the license server host machine is available in the License Administration Guide.*

See Also
ls_allow_tz_override
LM_A_TZ_OVERRIDE and “Flexible API Error Return Values” in the *C/C++ Function Reference*

Limitations and Interactions with Existing Licensing Functions

All existing licensing functionality can be combined with time zone licensing except for the limitations and interactions listed in this section.

Three-server Redundancy

There is a limitation when using time zone licensing in the case of three-server redundancy and a redundant configuration using a license search path (setting multiple license servers using the environment variable LM_LICENSE_FILE).

In a three-server configuration, the primary and secondary server should reside in the same time zone when TZ=SERVERTZ is specified in the feature definition lines.

For example, suppose that the primary and the secondary servers of a triad are configured to run in different time zones. If the primary server goes down and the secondary server becomes the master server, the client's time zone would no longer match the time zone of the master server. As a result, a checkout request to the secondary (acting as master) would fail. Therefore, to maintain the validity of licenses, it is recommended to configure primary and secondary servers to reside in the same time zone.

In a redundant configuration where multiple license servers are specified in the license search path either of the following is required to avoid license denials when an alternative license server is used:

- All license servers must reside in the same time zone. In this case no alteration of the TZ entry in the feature definition lines for different license servers is required and TZ=SERVERTZ can be used.
- When license servers reside in different time zones, the TZ entry in the feature definition lines for each license server must reflect its time zone and TZ=SERVERTZ cannot be used.

20

COAVAIL Checkout

The COAVAIL checkout functionality can help avoid checkout denials that can occur despite a sufficient number of licenses being available. A COAVAIL checkout is achieved by setting the checkout option flag `LM_CO_AVAIL_NOWAIT` when calling **lc_checkout**.

Read this chapter in conjunction with the section "lc_checkout" in the *C/C++ Function Reference*, which describes the function **lc_checkout** in detail.

Introduction

The **lc_checkout** function is used to request a license for a feature. The `flag` parameter of **lc_checkout** determines the response to a checkout denial. The following checkout option flags are available for **lc_checkout**: `LM_CO_NOWAIT`, `LM_CO_WAIT`, `LM_CO_QUEUE`, `LM_CO_LOCALTEST`, and `LM_CO_AVAIL_NOWAIT`. For a description of these flags and the syntax of **lc_checkout**, see the *C/C++ Function Reference*, section "lc_checkout".

The flag `LM_CO_AVAIL_NOWAIT` is best described by contrasting it with its sibling, `LM_CO_NOWAIT`.

The `LM_CO_NOWAIT` flag is used for producing a non-blocking response in case of a license denial. As we will see in the following example, a checkout request for a feature can be denied despite licenses for the feature being available on the license server. The flag `LM_CO_AVAIL_NOWAIT` can be used to avoid a checkout denial in specific scenarios.

For example, there are three versions (1.0, 2.0, and 3.0) of feature f1 available on the license server. Each license pool contains two licenses; hence, there are six licenses available in total.

A client application makes a checkout request for version 1.0 with the `LM_CO_NOWAIT` flag for three licenses of feature f1. On the license server, each license pool only contains two licenses, meaning that the request cannot be satisfied by any single license pool. Therefore, the request is denied and the checkout fails with the error code `LM_MAXUSERS`.

Now, a client application makes a checkout request for version 1.0 with the `LM_CO_AVAIL_NOWAIT` flag for three licenses of feature f1. The checkout request is granted by consuming the available license count of two licenses from the first satisfying license pool. If another call to **lc_checkout** is made to request the remaining license count (one license in this example), this request is also granted by consuming a license from a subsequent license pool.

A client example for the `LM_CO_AVAIL_NOWAIT` flag is available in the FlexNet Publisher Licensing Toolkit (`<platform_dir>\examples\advanced\client\lmflex_coavail.c`).

The COAVAIL checkout can be used for license file–based licensing and trusted storage–based licensing.

Usage Examples

This section provides the following examples which contrast the checkout option flags `LM_CO_NOWAIT` and `LM_CO_AVAIL_NOWAIT`:

- Example 1: License Checkout Using LM_CO_NOWAIT
- Example 2: License Checkout Using LM_CO_AVAIL_NOWAIT

Components

The examples use the following components:

- License server manager `lmgrd` (note that checkouts using the `LM_CO_AVAIL_NOWAIT` flag also work with `lmadmin`)
- Vendor daemon
- License file `counted.lic`
- Sample file `lmflex.exe` (Windows) or `lmflex` (UNIX)
- Sample file `lmflex_coavail.exe` (Windows) or `lmflex_coavail` (UNIX)

Preparation

Before you can run through the examples, you need to extract the FlexNet Publisher Licensing Toolkit to the installation folder and prepare the components as described in this section.

Preparation steps include:

- Preparing the Vendor Daemon
- Preparing lmflex_coavail
- Preparing the License File

Preparing the Vendor Daemon

COAVAIL checkouts do not require any specific vendor variables. Therefore, the standard vendor configuration file (`<platform_dir>\machind\lsvendor.c`) can be used for COAVAIL checkouts.

If you haven't already done so, build your vendor daemon.

Task ***To build the vendor daemon***

1. Open `<install_dir>\machind\lm_code.h` and add your vendor keys and publisher name.
2. At a command prompt from the `<platform_dir>` directory, depending on your platform, enter one of the following commands:
   - UNIX: **`make -f makefile`**
   - Windows: **`nmake -f makefile`**

For detailed instructions about the vendor daemon, see Chapter 3, License Servers, section Vendor Daemon.

Preparing lmflex_coavail

The FlexNet Publisher Licensing Toolkit comes with a sample file to demonstrate the COAVAIL checkout (that is, using the `LM_CO_AVAIL_NOWAIT` flag). This sample file `lmflex_coavail.c` is located in `<platform_dir>\examples\advanced\client\`. You need to build it to be able to use it in the example exercises.

Task ***To build lmflex_coavail***

1. Add `lmflex_coavail.exe` (Windows) or `lmflex_coavail` (UNIX) manually as a target to the `makefile`. (The build instructions for `lmflex_coavail` are similar to the toolkit example file `lmflex`.)
2. At a command prompt from the `<platform_dir>` directory, depending on your platform, enter one of the following commands:
   - UNIX: **`make -f makefile`**
   - Windows: **`nmake -f makefile`**

Preparing the License File

To demonstrate the COAVAIL checkout functionality in our exercises, you need a license file that contains licenses for feature f1 in different license pools.

Task ***To prepare a license file***

1. Create a license file called `counted.lic` as follows:

```
SERVER this_host <hostid> 27001
VENDOR demo
USE_SERVER
#a counted license
INCREMENT f1 demo 1.0 permanent 4 SIGN=0
INCREMENT f1 demo 2.0 permanent 5 SIGN=0
INCREMENT f1 demo 3.0 permanent 6 SIGN=0
```

   where `<hostid>` is the hostid of the machine on which `lmflex_coavail` will run.

2. Place the license file in the `<platform_dir>` directory.
3. Sign the license file using the `lmcrypt` utility. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmcrypt counted.lic
   ```

Example 1: License Checkout Using LM_CO_NOWAIT

This example demonstrates the checkout behavior of `LM_CO_NOWAIT`, using the `lmflex` sample file. Compare this behavior to the behavior in example 2, where the `LM_CO_AVAIL_NOWAIT` flag is used.

Ensure that you prepared the components for this example as described in section Preparation.

Task ***To demonstrate the checkout behavior when using the LM_CO_NOWAIT flag***

1. Start the license server manager with the license file `counted.lic`. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmgrd -c counted.lic -l debug.log
   ```

2. Use the `lmstat` utility to check the license usage status on the license server. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmstat -a -c 27001@<serverhost>
   ```

   Output similar to the following is displayed, indicating that all 15 licenses for feature f1 are available (four licenses from pool version 1.0, five licenses from pool version 2.0, and six licenses from pool version 3.0):

   ```
   ...
   Users of f1:  (Total of 15 licenses issued;  Total of 0 licenses in use)
   ...
   ```

3. Simulating User1, check out three licenses of feature f1 using the `lmflex` client application. At a new command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmflex 3
   ```

   Press **Enter** to check out floating licenses.

   An output message confirms the checkout of feature f1: `f1 checked out...press return to exit...`

4. Simulating User2, check out another four licenses of feature f1 by entering the following:

   ```
   lmflex 4
   ```

   Press **Enter** to check out floating licenses.

   An output message confirms the checkout of feature f1.

5. Simulating User3, check out another three licenses of feature f1 by entering the following:

   ```
   lmflex 3
   ```

   Press **Enter** to check out floating licenses.

   An output message confirms the checkout of feature f1.

6. Use the `lmstat` utility as in Step 2 to check the license usage status again. The license status is now as follows:

   - License pool v1.0 contains four licenses in total, of which one license is available.
   - License pool v2.0 contains five licenses in total, of which one license is available.
   - License pool v3.0 contains six licenses in total, of which three licenses are available.

   The `lmstat` output should look similar to the following:

```
...
Users of f1:  (Total of 15 licenses issued; Total of 10 licenses in use)

    "f1" v1.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User1 Host1 Disp1 (v1.0) (serverhost/27001 101), start Thu 2/18 22:23, 3 licenses

    "f1" v2.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User2 Host2 Disp2 (v1.0) (serverhost/27001 101), start Thu 2/18 22:23, 4 licenses

    "f1" v3.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User3 Host3 Disp3 (v1.0) (serverhost/27001 101), start Thu 2/18 22:23, 3 licenses
```

7. Simulating User4, try to check out another five licenses of feature f1 by entering the following:

   `lmflex 5`

   Press **Enter** to check out floating licenses.

   A message similar to the following is displayed:

```
Checkout failed: Licensed number of users already reached (-4, 132)
```

Note ▪ *Keep the licenses checked out, because you will use them in example 2.*

Conclusion

Because the checkout request could not be satisfied by any single license pool—none of the three license pools has five licenses available—the request is denied by the license server.

Example 2: License Checkout Using LM_CO_AVAIL_NOWAIT

This example demonstrates the checkout behavior of `LM_CO_AVAIL_NOWAIT`, using the `lmflex_coavail` sample file. It should be contrasted with the behavior of `LM_CO_NOWAIT` that is described in Example 1: License Checkout Using LM_CO_NOWAIT.

Note that `lmflex_coavail` makes multiple calls to **lc_checkout** until the requested number of licenses have been checked out (provided that a sufficient number of licenses is available in the license pools).

Ensure that you prepared the components for this example as described in section Preparation.

Task ***To demonstrate the checkout behavior when using the LM_CO_AVAIL_NOWAIT flag***

1. If you performed the procedure that is described in Example 1: License Checkout Using LM_CO_NOWAIT, and 10 licenses from three different license pools are checked out, continue with Step 3. Otherwise, go to Step 2.
2. Perform Step 1 through Step 6 from Example 1: License Checkout Using LM_CO_NOWAIT to check out a total of ten licenses from three different license pools.
3. Use the `lmstat` utility to check the license usage status on the license server. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmstat -a -c 27001@<serverhost>
   ```

   The license status should be as follows:

   - License pool v1.0 contains four licenses in total, of which one license is available.
   - License pool v2.0 contains five licenses in total, of which one license is available.
   - License pool v3.0 contains six licenses in total, of which three licenses are available.
4. Simulating User4, check out five licenses of feature f1 using the `lmflex_coavail` client application. Enter the following:

   ```
   lmflex_coavail 5
   ```

   Output similar to the following is displayed:

   ```
   ===== COAVAIL Checkout Request-1 =====
           Feature = f1
           Version = 1.0
           License count requested = 5
           License count needed = 5
           Feature f1 checkout success...
                   Incremental license count: (requested = 5) ==> (obtained = 1)
           *** Feature f1 (1.0), total license count obtained = 1 ***

                   ====== CONFIG Info ============
                   Feature = f1
                   Version = 1.0
                   Code = XXX1
                   COAVAIL license consumption count = 1
                   ====== END CONFIG Info ============

   ===== COAVAIL Checkout Request-2 =====
           Feature = f1
           Version = 1.0
           License count requested = 5
           License count needed = 4
           Feature f1 checkout success...
                   Incremental license count: (requested = 4) ==> (obtained = 1)
           *** Feature f1 (1.0), total license count obtained = 2 ***

                   ====== CONFIG Info ============
                   Feature = f1
                   Version = 2.0
                   Code = XXX2
   ```

```
                COAVAIL license consumption count = 1
                ====== END CONFIG Info ============

===== COAVAIL Checkout Request-3 =====
        Feature = f1
        Version = 1.0
        License count requested = 5
        License count needed = 3
        Feature f1 checkout success...
                Incremental license count: (requested = 3) ==> (obtained = 3)
        *** Feature f1 (1.0), total license count obtained = 5 ***

                ====== CONFIG Info ============
                Feature = f1
                Version = 3.0
                Code = XXX3
                COAVAIL license consumption count = 3
                ====== END CONFIG Info ============

        No more license needed...
-------------------------------------------------
        Press ENTER to checkin license(s)...
```

The output above shows the multiple checkout requests are made to request the desired number of licenses from the license server. This can be broken down as follows:

a. The first checkout request is made for five licenses. The client receives one license from the v1.0 license pool. The client still needs four more licenses.

b. The second checkout is made for four licenses. The client receives one license from the v2.0 license pool. The client still needs three more licenses.

c. The third checkout is made for three licenses. The client receives three licenses from the v3.0 license pool. Now, all requested licenses (five in total) have been checked out.

5. Use the `lmstat` utility to check the license usage status on the license server.

```
Users of f1:  (Total of 15 licenses issued; Total of 15 licenses in use)
    "f1" v1.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User1 Host1 Disp1 (v1.0) (localhost/27001 101), start Thu 2/18 22:23, 3 licenses
        User4 Host4 Disp4 (v1.0) (localhost/27001 401), start Thu 2/18 22:28

    "f1" v2.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User2 Host2 Disp2 (v1.0) (localhost/27001 201), start Thu 2/18 22:23, 4 licenses
        User4 Host4 Disp4 (v1.0) (localhost/27001 501), start Thu 2/18 22:28

    "f1" v3.0, vendor: <name>, expiry: permanent(no expiration date)
    floating license
        User3 Host3 Disp3 (v1.0) (localhost/27001 301), start Thu 2/18 22:23, 3 licenses
        User4 Host4 Disp4 (v1.0) (localhost/27001 601), start Thu 2/18 22:28, 3 licenses
```

Conclusion

In `lmflex_coavail`, the `LM_CO_AVAIL_NOWAIT` flag and the multiple calls to **lc_checkout** result in the license server granting the license request even though the requested number of licenses cannot be satisfied using a single license pool. Instead, the available number of licenses is checked out from each license pool until the request has been satisfied.

Error Codes

These codes are defined in the standard client header file (`<platform_dir>\machind\lmclient.h`) in your FlexNet Publisher Licensing Toolkit.

Table 20-1 ▪ Error codes of COAVAIL checkout

Error Code	Description
LM_COAVAIL_RECONN_NOT_ALL_CNT_AVAIL	Failed to get all requested license(s) in reconnection.
LM_COAVAIL_FLAG_NOT_SUPPORTED	The supplied checkout option flag is not supported. This error code is used when attempting to use the checkout flag `LM_CO_AVAIL_NOWAIT_RC` (defined in `lmclient.h`) which is meant for use by Revenera.
LM_COAVAIL_PACKAGE_NOT_SUPPORTED	The checkout-avail flag is not supported with PACKAGE licenses.
LM_COAVAIL_UNSUPPORTED	The checkout-avail (COAVAIL) checkout is not supported. This error code is used when attempting to consume served uncounted licenses or when `DUP_GROUP` is used.

Limitations

The following limitations apply for the `LM_CO_AVAIL_NOWAIT` flag.

- The `LM_CO_AVAIL_NOWAIT` flag is supported in C clients only; it is not supported in Java clients.
- The `LM_CO_AVAIL_NOWAIT` checkout cannot be combined with checkouts made with checkout option flags other than `LM_CO_NOWAIT` in the same license job. Note that checkouts of `LM_CO_AVAIL_NOWAIT` and `LM_CO_NOWAIT` could be made interchangeably in the same license job.
- A checkout denial with error `LM_MAXLIMIT` or `LM_MAXLIMIT_EXCEED` occurs for COAVAIL checkouts if the available license count is greater than MAX.
- The `LM_CO_AVAIL_NOWAIT` flag is not supported with `DUP_GROUP`. When used, error code `LM_COAVAIL_UNSUPPORTED` is reported.
- The `LM_CO_AVAIL_NOWAIT` flag is not supported with the PACKAGE keyword. When used, error code `LM_COAVAIL_PACKAGE_NOT_SUPPORTED` is reported.

- `LM_CO_AVAIL_NOWAIT` is not supported with the borrow checkout. The limitation lies when the first COAVAIL checkout consumes a subset of the requested licenses, and the subsequent COAVAIL checkout is issued to consume the rest of the required licenses from the license server. In this case, the subsequent COAVAIL checkout yields an incorrect CONFIG type and an incorrect available license consumption count.

21

Batch Checkout

The Batch checkout functionality allows to checkout a collection of features and its quantity from the licenses server in a single transaction with Vendor Daemon. A Batch checkout is achieved by using the flexible APIs, which are listed in the APIs section.

Read this chapter in conjunction with the section “Flexible API Function Descriptions” in the *C/C++ Function Reference*, which describes the function APIs in detail.

Introduction

As a producer who has many features might want to specify a collection of features and its count to checkout from license server. As a result of this producer can avoid making multiple feature requests that result in a better server performance.

The Flexible APIs of Batch Checkout are used to license for a collections of feature. These APIs help in creation or freeing up of the transaction, one or more entries in the transaction, sending a transaction to license server and checking the results of each entry that were submitted in the transaction.

The following APIs are used for the batch checkout:

- lc_txn_create()
- lc_txn_destroy()
- lc_txn_create_checkout
- lc_txn_destroy_entry()
- lc_txn_entry_count()
- lc_txn_entry()
- lc_txn_set_attr()
- lc_txn_get_attr()
- lc_txn_entry_status()

- lc_txn_entry_checkout_info()
- lc_txn_send()

For description of these APIs and their syntax, see the *C/C++ Function Reference*, section Flexible API Description.

When a client application makes Batch Checkout request, the server considers it as a standard single checkout request and there may be some message exchanges between the server and client for additional information (e.g. additional hostid details). Once the additional information are retrieved for a checkout, which is been processed, the data will be available for all subsequent checkouts of the batch. While one of these information exchange taking place, an index value is passed in the message header from the server corresponding to the index into the batch of the checkout request currently being processed.

There may be a preliminary exchange prior to the initial Batch Checkout message where Checkout Filters are run for those entries that have the appropriate attribute set. As a result of these exchanges the `VendorCode` associated with each such entry may be updated from the default `PORT_AT_HOST` value.

Once all the checkouts in the initial Batch Checkout message have been processed by the server, a subsequent aggregated message exchange to validate vendor information is performed.

If there are no job-level errors, the client API repeats the above message cycle until all the checkout requests specified by the server have been completed. Then, it returns to the caller. Thus, as far as the caller is concerned, any number of checkout requests can be added to a Batch Checkout.

The attributes supported at transaction entry level are:

Table 21-1 ▪ The Supported Attributes in Batch Checkout

LM_A_BORROW_EXPIRE	LM_A_BORROW_RETURN_BY_VERSION	LM_A_CHECKOUT_DATA
LM_A_CHECKOUTFILTER LM_A_CHECKOUTFILTER_EX	LM_A_CHECKOUTFILTERLAST_EX	LM_A_CHECK_LOCAL_TS_SIGNATURE
LM_A_IGNORE_FILE_CHECK_IN_HEARTBEAT (Linux Only)	LM_A_LINGER	LM_A_MULTIPLE_CHECKOUT_DATA
LM_A_RETRY_COUNT LM_A_RETRY_INTERVAL	LM_A_SERVEDTRANSFER_EXPIRE	LM_A_SORT_TS_FIRST
LM_A_TS_ONLY	LM_A_VENDOR_CALLBACK_DATA	

The attributes not supported in batch checkout are:

Table 21-2 ▪ The Non-Supported Attributes in Batch Checkout

LM_A_APP_DISABLE_CACHE_READ	LM_A_BEHAVIOR_VER	LM_A_BORROW_STAT
LM_A_CHECK_BADDATE	LM_A_CHECK_INTERVAL	LM_A_CKOUT_INSTALL_LIC

Table 21-2 ▪ The Non-Supported Attributes in Batch Checkout

LM_A_CONN_TIMEOUT	LM_A_CROSS_TZ_OK	LM_A_DIAGS_ENABLED
LM_A_DISABLE_ENV	LM_A_DISPLAY_OVERRIDE	LM_A_HEARTBEAT_LOCK (Windows only)
LM_A_HOST_OVERRIDE	LM_A_IGNORE_CACHED_SERVER_DATA	LM_A_INTERNET_OVERRIDE
LM_A_LF_LIST	LM_A_LICENSE_DEFAULT	LM_A_LICENSE_CASE_SENSITIVE
LM_A_LICENSE_FMT_VER	LM_A_LKEY_CASE_SENSITIVE	LM_A_LONG_ERRMSG
LM_A_MT_HEARTBEAT (UNIX Only)	LM_A_NO_MT_HEARTBEAT_TIMER (UNIX Only)	LM_A_PERIODIC_CALL
LM_A_PERIODIC_COUNT	LM_A_PERROR_MSGBOX (Windows Only)	LM_A_PHYSICAL_ETHERNETID (Windows Only)
LM_A_PLATFORM_OVERRIDE	LM_A_PORT_HOST_PLUS ***Note ▪*** *Combining the LM_A_PORT_HOST_PLUS attribute with batch checkout APIs will not give any additional benefits.*	LM_A_PROMPT_FOR_FILE (Windows Only)
LM_A_RETRY_CHECKOUT	LM_A_TCP_TIMEOUT	LM_A_TS_CHECK_BADDATE
LM_A_TZ_OVERRIDE	LM_A_USE_FQDN	LM_A_USER_EXITCALL LM_A_USER_EXITCALL_EX
LM_A_USER_OVERRIDE	LM_A_USER_RECONNECT LM_A_USER_RECONNECT_EX	LM_A_USER_RECONNECT_DONE LM_A_USER_RECONNECT_DONE_EX
LM_A_USER_RECONNECT_V1_DATA	LM_A_USER_RECONNECT_V1	LM_A_USER_RECONNECT_V1_DONE
LM_A_VD_GENERIC_INFO LM_A_VD_FEATURE_INFO	LM_A_VENDOR_ID_DECLARE	LM_A_VM_FAMILY
LM_A_VM_NAME	LM_A_WINDOWS_MODULE_HANDLE (Windows only)	

Usage Examples

This section provides the following examples for various checkout type:

- Example 1: Simple Batch Checkout
- Example 2: Batch Checkout with Attributes
- Example 3: Batch Checkout with 2 Transactions

Example 1: Simple Batch Checkout

This example demonstrates the simple Batch Checkout:

```
int simple_batch_checkout(LM_HANDLE* job, int count, const char** feats, int* results)

{

    int res = LM_NOERROR; // Optimist
    const char* vers = "1.0"; // Use version "1.0" of the licences
    const int nlic = 1; // Checkout 1 instance of each of the licenses
    LM_TXN txn = LM_TXN_NULL; // Transaction handle
    do
    {
        int i;
        // Create the transaction
        //
        if( res=lc_txn_create(job, &txn) != LM_NOERROR )
            break;
        // Create the batch checkout entries
        //
        for( i=0; i<count; i++ )
        {
            LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
            res=lc_txn_create_checkout_entry(job, &txn, &txn_entry, feats[i], vers, nlic, LM_CO_NOWAIT,
            LM_DUP_NONE);
            if( res != LM_NOERROR )
            break;
        }
        if( i != count )

            break;

        // Perform sent the transaction to be processed

        //

        if( (res=lc_txn_send(job, &txn)) != LM_NOERROR)
            break;
        // Iterate over the individual checkout requests, reporting the status
        //
        for( i=0; i<count; i++ )
        {
            LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
            if( (res=lc_txn_entry(LM_HANDLE job, &txn, &txn_entry, i)) != LM_NOERROR )
               break;
            if( (res=lc_txn_entry_status(job, &txn_entry, &results[i])) != LM_NOERROR )
```

```
                break;
        }
    } while(0);
    (void)lc_txn_destroy(job,&txn);
    return res;

}
```

Example 2: Batch Checkout with Attributes

This example demonstrates the Batch Checkout with attributes:

```
int batch_checkout_with_attr(LM_HANDLE* job, int count, const char** feats, int* results)
    {
    int res = LM_NOERROR; // Optimist
    const char* vers = "1.0"; // Use version "1.0" of the licences
    const int nlic = 1; // Checkout 1 instance of each of the licenses
    LM_TXN txn = LM_TXN_NULL; // Transaction handle
    do
    {
        int i;
        // Create the transaction
        //
        if( res=lc_txn_create(job, &txn) != LM_NOERROR )
            break;
        // Create the batch checkout entries
        //
        for( i=0; i<count; i++ )
        {
            LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
            const char* vcd; // Vendor Checkout data
            res=lc_txn_create_checkout_entry(job, &txn, &txn_entry, feats[i], vers, nlic, LM_CO_NOWAIT,
LM_DUP_VENDOR);
            if( res != LM_NOERROR )
               break;
            // We use Vendor Checkout Data "low" for the first half of the features and "high" for the
            rest
            //
            vcd = ( i<count/2 ) ? "low" : "high";
            if( (res=lc_txn_entry_set_attr(job, &txn_entry, LM_A_CHECKOUT_INFO, vcd)) != LM_NOERROR )
            break;
        }
        if( i != count )
        break;
        // Set a 30s heart-beat for job. (This would fail if passed to lc_txn_entry_set_attr())
        //
        if( (res=lc_set_attr(job, LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE) 30)) != LM_NOERROR )
            break;
        // Perform sent the transaction to be processed
        //
        if( (res=lc_txn_send(job, &txn)) != LM_NOERROR)
            break;
        // Iterate over the individual checkout requests, reporting the status
        //
        for( i=0; i<count; i++ )
```

```
        {
            LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
            if( (res=lc_txn_entry(LM_HANDLE job, &txn, &txn_entry, i)) != LM_NOERROR )
            b  reak;
            if( (res=lc_txn_entry_status(job, &txn_entry, &results[i])) != LM_NOERROR )
               break;
        }
    } while(0);
    (void)lc_txn_destroy(job,&txn);
    return res;

}
```

Example 3: Batch Checkout with 2 Transactions

This example demonstrates the Batch Checkout with 2 transactions:

```
int batch_checkout_with_two_txn(LM_HANDLE* job, int count, const char** feats, int* results)
{
    int res = LM_NOERROR; // Optimist
    const char* vers; // Desired version of the licences
    const int nlic = 1; // Checkout 1 instance of each of the licenses
    LM_TXN txn1 = LM_TXN_NULL; // Transaction handle for Phase 1
    LM_TXN txn2 = LM_TXN_NULL; // Transaction handle for Phase 2
    // Phase 1: Attempt to checkout version 2.0 of the licenses
    //
    do
    {
        int i;
        vers = "2.0";
        // Create the transactions
        //
        if( (res=lc_txn_create(job, &txn1)) != LM_NOERROR || (res=lc_txn_create(job, &txn2)) !=
        LM_NOERROR )
            break;
        // Create the phase 1 batch checkout entries
        //
        for( i=0; i<count; i++ )
        {
            LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
            res=lc_txn_create_checkout_entry(job, &txn1, &txn_entry, feats[i], vers, nlic,
            LM_CO_NOWAIT,
            LM_DUP_NONE);
            if( res != LM_NOERROR )
               break;
        }
        if( i != count )
            break;
        // Perform sent the phase 1 transaction to be processed
        //
        if( (res=lc_txn_send(job, &txn1)) != LM_NOERROR)
            break;
        // Iterate over the phase 1 checkout requests
        //
        for( i=0; i<count; i++ )
```

```
{
    LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
    if( (res=lc_txn_entry(LM_HANDLE job, &txn1, &txn_entry, i)) != LM_NOERROR )
       break;
    // Process the Phase 1 results
    //
    if( (res=lc_txn_entry_status(job, &txn_entry, &results[i])) != LM_NOERROR )
       break;
    if( results[i] != LM_NOERROR )
    {
       // Failure on Phase 1, so add to the Phase 2 transaction.
       // We change to use version "1.0" of the license, and queue if it's not immediately
    available
       //
       int j;
       vers = "1.0";
       // First extract the checkout data
       //
       const struct LM_TXN_ENTRY_CHECKOUT_INFO* co_data;
       if( (res=lc_txn_entry_checkout_info(job, &txn_entry, &co_data)) != LM_NOERROR )
       break;
       // Create the new entry in txn2
       //
       res=lc_txn_create_checkout_entry(job, &txn2, &txn_entry, co_data->feature, vers, nlic,
LM_CO_QUEUE, LM_DUP_NONE);
       if( res != LM_NOERROR )
          break;
       // Copy over any entry attributes
       //
       for( j=0; j<co_data->attr_count; j++ )
       {
          struct LM_TXN_ENTRY_CHECKOUT_ATTR* attr = &co_data->attr[j];
          if( (res=lc_txn_entry_set_attr(job, &txn_entry, attr->key, (LM_A_VAL_TYPE)attr-
    >value)) !=LM_NOERROR )
             break;
       }
       if( res != LM_NOERROR )
          break;
       }
    }
    if( res != LM_NOERROR || (res=lc_txn_entry_count(job, txn2)) <= 0 )
    break;
    // Perform send the phase 2 transaction to be processed
    //
    if( (res=lc_txn_send(job, &txn2)) != LM_NOERROR)
       break;
    // Incorporate the Phase 2 results
    //
    for( i=0, j=0; i<count; i++ )
    {
       if( results[i] != LM_NOERROR )
       {
          LM_TXN_ENTRY txn_entry = LM_TXN_ENTRY_NULL;
          if( (res=lc_txn_entry(LM_HANDLE job, &txn2, &txn_entry, j++)) != LM_NOERROR )
             break;
```

```
                    // Update with the Phase 2 result
                    //
                    if( (res=lc_txn_entry_status(job, &txn_entry, &results[i])) != LM_NOERROR )
                       break;
                 }
              }
              if( i != count )
                 break;
           } while(0);
           (void)lc_txn_destroy(job,&txn1);
           (void)lc_txn_destroy(job,&txn2);
           return res;

}
```

Error Codes

These codes are defined in the standard client header file (`<platform_dir>\machind\lmclient.h`) in your FlexNet Publisher Licensing Toolkit.

Table 21-3 ▪ Error codes of Batch Checkout

Error Code	Description
LM_TXN_NOEXISTS	The transaction handle is invalid or not associated with this job.
LM_TXN_ATTR_NOTSET	The specified attribute is not set for this transaction entry.
LM_TXN_TOO_MANY_ENTRY	The transaction cannot accept any further entries

Supported Checkout Scenarios

The possible supported scenarios for the Batch Checkout are:

- Checkout scenarios with the following possible values for Checkout Option Flag:
  - LM_CO_NOWAIT (Already-tested scenario - in the default "lmflex.c")
  - LM_CO_AVAIL_NOWAIT
  - LM_CO_WAIT
  - LM_CO_QUEUE (Queued checkout scenario)
  - LM_CO_LOCALTEST
- Custom DUP_GROUP checkout (using LM_A_CHECKOUT_DATA attribute)
- Bulk Custom DUP_GROUP checkout (using LM_A_MUTIPLE_CHECKOUT_DATA attribute)
- License borrowing with BORROW keyword
- Overdraft checkout of features specified with OVERDRAFT keyword

- Checkout of features specified using the TZ keyword (Timezone scenarios)
- Node-locked license - Unserved model
  - Node-locked to a laptop/desktop computer
  - Node-locked to a FlexNet ID Dongle
  - Node-locked to a FlexNet ID Dongle with FLOAT_OK keyword
  - Node-locked to a User Name
- Node-locked license served from a license server
- License Fulfilled from a Prepaid License Pool
- Checkout of concurrent and hybrid licenses served from Server-Side Trusted Storage - ?
- Checkout of features of type PACKAGE (without the OPTIONS keyword on PACKAGE line)
- Checkout of features of type PACKAGE (having OPTIONS=SUITE on PACKAGE line). Here, a Batch checkout and a set of individuallc_checkout()-s are expected to yield similar results/behavior:
  - with SUITE_DUP_GROUP=U specified for the PACKAGE-enabling FEATURE line
  - with SUITE_DUP_GROUP=U not specified for the PACKAGE-enabling FEATURE line
- Checkout of features of type PACKAGE (having OPTIONS=SUITE_RESERVED on PACKAGE line). Here, a Batch checkout and a set of individual lc_checkout()-s are expected to yield similar results/behavior:
  - with SUITE_DUP_GROUP=U specified for the PACKAGE-enabling FEATURE line
  - with SUITE_DUP_GROUP=U not specified for the PACKAGE-enabling FEATURE line
- Checkout filter attributes LM_A_CHECKOUTFILTER and LM_A_CHECKOUTFILTER_EX

Note ▪ *The above listed scenarios are possible supported scenarios for Batch Checkout and the implementation may be phased in future releases based on the requirement. Where a particular scenario is not implemented for Batch Checkout it must fail in a deterministic fashion returning an LM_NOTSUPPORTED error.*

Limitation

The following limitation apply for the Batch Checkout:

- A single Batch Checkout exchange between client and server is limited by message size restriction to a maximum number of checkouts defined by macro <name>.

22

Virtualization

You can use FlexNet Publisher to implement license models that determine whether the license server or the client is running in a virtual environment. Use FlexNet Publisher virtualization features to create a license policy that limits usage of FlexEnabled applications to physical machines only or to supported virtual machines only. You can also enforce if a license server is allowed to run on a virtual machine or not by specifying virtualization hostids on the SERVER line in the license file.

By default, FlexNet Publisher clients and license servers make no distinction between physical and virtual machines. Both the FlexEnabled application and the license server can be run on a physical machine or in a virtual environment.

- Understanding Virtualization Features
- Limitations and Interactions with Existing Licensing Functions

Understanding Virtualization Features

Virtualization features enable you to restrict licensing to virtual environments or to physical machines. (Virtualization-based restrictions are ignored on unsupported virtual environments.)

Key Virtualization Features

Virtualization functionality is controlled by these key features:

- **Vendor daemon variable: ls_allow_vm** enables you to specify whether a vendor daemon can run on physical machines only or virtual environments only.
- **License Keyword: VM_PLATFORMS** enables you to specify whether a feature is restricted to physical machines only or virtual environments only.
- **Hostid Keyword on the SERVER line:** A hostid keyword on the SERVER line enables you to specify a Virtualization policy regarding license servers. The hostid keyword can take the form VM_UUID or PHY_*, indicating restrictions for a virtual machine or physical host. If either of these keyword types is used, the value of the variable `ls_allow_vm` is ignored.

Virtualization Requirements

Generally, virtualization functionality:

- Requires vendor keys (provided by Revenera) that support Virtualization.
- Requires a vendor daemon and FlexEnabled application built with FlexNet Publisher 11.7 or newer.
- Requires that the FlexNet Licensing Service be installed. For more information, see Virtualization and the FlexNet Licensing Service.

In addition, for FlexEnabled applications, `VM_PLATFORMS` successfully restricts licenses to virtual environments only when the FlexEnabled application is running in a supported virtual environment platform or physical machine.

For vendor daemons, the successful operation of the `ls_allow_vm` setting requires that a vendor daemon is running in a supported virtual environment platform or on a physical machine.

Note ▪ *See the current FlexNet Publisher Release Notes for a complete list of supported virtual environment platforms.*

Prohibiting or Allowing Virtualization Support

The implementation process varies depending on whether you are implementing Virtualization support for the vendor daemon or your FlexEnabled applications.

Note ▪ *Defaults are that licensing is allowed on both physical and virtual platforms.*

Task ***To implement Virtualization support on a license server:***

1. Modify the file `lsvendor.c` to specify the value for the ls_allow_vm vendor variable.
2. Rebuild the vendor daemon.
3. Distribute the updated vendor daemon to customers.

Task ***To implement Virtualization support on a FlexEnabled application:***

1. Update license files to include the new license keyword VM_PLATFORMS on feature definition lines that should be limited to, or restricted from, usage in virtual environments. Include cross-version signatures if implementing for deployed applications built using pre-11.7 versions of FlexNet Publisher.
2. Encrypt the updated license files.
3. Distribute the updated license files to customers.

See Also
VM_PLATFORMS
ls_allow_vm
“Flexible API Error Return Values” in the *C/C++ Function Reference*

Virtualization and the FlexNet Licensing Service

For producers who want to use virtualization features, best practice is to have the FlexNet Licensing Service started automatically on OS boot, so that virtualization attribute extraction is complete ahead of a first-checkout.

Without the FlexNet Licensing Service, the following virtualization behavior occurs:

- SERVER line hostid keyword VM_UUID is not extracted: the vendor daemon will therefore shut down if this hostid is used when the FlexNet Licensing Service is not installed on the server.
- `lc_virtualstatusget` returns the pre-existing error code `-1` instead of `0` (physical) or `1` (virtual).
- `lc_get_attr` (LM_A_VM_FAMILY or LM_A_VM_NAME) returns a null attribute string.
- The server log (SLOG) virtualization entry reports the virtualization status as `Not determined`.

For information about installing and uninstalled the FlexNet Licensing Service, see Chapter 15, FlexNet Licensing Service.

Limitations and Interactions with Existing Licensing Functions

All existing licensing functionality can be combined with the virtualization vendor daemon variable and license keyword except for the limitations and interactions listed in this section.

- Duplicate Grouping using DUP_GROUP
- Terminal Services Clients: TS_OK and ONE_TS_OK
- PLATFORMS

Duplicate Grouping using DUP_GROUP

Duplicate grouping operates normally when the `VM_PLATFORMS` keyword is also included in the feature definition line. However, the `VM_PLATFORMS` setting also applies to determine whether a checkout is treated as a duplicate.

Consider the following example feature definition line:

```
INCREMENT edit demo 2.0 31-dec-2020 10 DUP_GROUP=U VM_PLATFORMS=PHYSICAL SIGN=.....
```

A user attempts to checkout the feature **edit** from the following machines:

- On physical machine A—the checkout succeeds.
- On physical machine B—the checkout succeeds and is treated as a duplicate of the checkout from A, and so does not consume an additional license.
- On a virtual machine running on physical machine C—the checkout fails because the request is from a virtual machine
- On a virtual machine running on physical machine A—the checkout fails because the request is from a virtual machine.

Terminal Services Clients: TS_OK and ONE_TS_OK

Licensing controls on the use of terminal services clients—by the feature definition lines TS_OK and ONE_TS_OK—are implemented as normal when the feature definition line contains the VM_PLATFORMS keyword. VM_PLATFORMS is validated first as is illustrated in the following example:

```
INCREMENT heap demo 2.0 31-dec-2020 10 TS_OK VM_PLATFORMS=PHYSICAL SIGN=.....
```

A checkout request for the feature **heap** is issued from a remote desktop connection to a virtual machine. The checkout fails due to the request being made from a virtual machine (error -190 LM_VM_PHYSICAL_ONLY reported).

PLATFORMS

When the keyword PLATFORMS and VM_PLATFORMS are used in the same feature definition line, they both are applied. So, for the following example feature definition line:

```
INCREMENT part demo 1.0 31-dec-2020 8 PLATFORMS="i86_n i86_linux", VM_PLATFORMS=PHYSICAL SIGN=.....
```

- PLATFORMS="i86_n i86_linux" limits the license to Windows 32-bit or Linux 32-bit machines.
- VM_PLATFORMS=PHYSICAL limits the license to physical machines only.

Thus this license is valid only on a Windows or a Linux physical machine.

The PLATFORMS keyword is validated first so the following errors are reported on failed checkout of feature **part**:

- Checkout attempted from a virtual machine on a Windows 32-bit machine reports error -190 LM_VM_PHYSICAL_ONLY.
- Checkout attempted from a virtual machine on a Windows 64-bit machine reports error -89 LM_PLATNOTLIC.

23

Mobile Licensing

Users often want to use applications on systems that do not have a continuous connection to a license server. These situations include:

- Working on a laptop
- Using a computer both at work and at home
- Working from several different computers not connected to a license server

FlexNet Publisher supports licenses that allow one of several kinds of mobile licensing:

- Node-Locked to a Laptop Computer
- Node-Locked to a FlexNet ID Dongle
- Node-Locked to a FlexNet ID Dongle with FLOAT_OK
- Node-locked to a User Name
- Fulfilled from a Prepaid License Pool
- License borrowing using the activation borrow API. See *Programming Reference for Trusted Storage–Based Licensing* for more information about this capability.
- License Borrowing using the BORROW Keyword
- License Transferring Using the Transferrable Counted Model—Windows Only

License rehosting is the consequence of a user wanting to move a license without using one of these methods. This means a new, node-locked license file is generated by the publisher for each new system. The user is trusted to destroy the previous licenses so that only one is in force at any given time. Rehosting is cumbersome and not secure because:

- Administrative overhead is incurred because the software publisher is involved with each move.
- The user can get multiple uses from a single license if older license files are not destroyed.

Node-Locked to a Laptop Computer

If a license is to be used exclusively on one laptop computer, that license can simply be node-locked to a hostid of that computer. The license file would reside on the laptop computer.

Node-Locked to a FlexNet ID Dongle

If a license is to be moved between different systems, it can be node-locked to a FlexNet ID dongle that connects to a parallel or USB port. This license can be moved between systems by installing a copy of the license file on each system and moving the dongle from one system to another. See Using FlexNet ID Dongles for a list of platforms for which FlexNet ID dongles are available.

Node-Locked to a FlexNet ID Dongle with FLOAT_OK

This method of license mobility has an advantage over simply issuing a license node-locked to a FlexNet ID dongle, because the dongle can be attached to a system where the license server runs and its license can float on the network.

A vendor issues the user a license file with a `FEATURE` line node-locked to a FLEXID hostid and containing the `FLOAT_OK` keyword and a FLEXID for that `FEATURE` line. One `FEATURE` line containing the `FLOAT_OK` keyword and one FLEXID is needed for each instance of a license that can be mobile. When the dongle is attached to the system where the license server runs, the license floats on the network. When the dongle is removed from the system, the license is available only on the standalone computer with the dongle attached.

This method supports both parallel and USB FlexNet ID dongles.

Using FlexNet ID Dongle with FLOAT_OK Keyword

This section describes the procedures that both software publishers and license administrators must perform to use this capability in a license model.

Procedure for Software Publishers

To allow a user to use a FlexNet ID dongle with `FLOAT_OK`:

1. Issue a license file to your user with a `FEATURE` line containing the `FLOAT_OK` keyword. That `FEATURE` line is uncounted and node-locked to a FLEXID. One `FEATURE` line is needed for each instance of a mobile license. For example:

   ```
   FEATURE f1 xyzd 1.0 permanent uncounted FLOAT_OK \
       HOSTID=FLEXID=9-b28520b9 SIGN=123456789012
   ```

2. Send your user the dongle to which the `FEATURE` line is node-locked. One dongle is needed for each instance of a license that can be mobile.

See Using FlexNet ID Dongles for a list of supported FlexNet ID dongles.

Procedure for License Administrators

While the FlexNet ID dongles are attached to the system where the license server runs, the node-locked licenses associated with them float on the network. Each of the FLOAT_OK uncounted node-locked FEATURE lines has a count of *one* while it is available on the network.

To transfer a license from the pool of floating licenses to a disconnected computer, a user must put a copy of the node-locked FLOAT_OK license and the matching dongle on the disconnected computer. When the dongle is removed from the system, this license is unavailable on the network and is available on the computer with the dongle.

To return a license to the license server, so it can float on the network again, the user removes the dongle from the disconnected computer and replaces it on the system where the license server runs, then rereads the license file (lmreread) so the license server once again serves the floating license.

See the *License Administration Guide* for detailed instructions a user would use to initiate FlexNet ID dongle with FLOAT_OK mobility.

Security Issues for FlexNet ID Dongle with FLOAT_OK

By default, it is possible to get one extra use from FLOAT_OK licenses by running a node-locked license on the license server, because the license can also float while on the license server. This extra use can be disabled, though it is less convenient, using a hostid modifier on the FLOAT_OK keyword:

FLOAT_OK=*server_hostid*

For example:

```
SERVER myhost 009027319bbb
VENDOR xyzd
FEATURE f1 xyzd 1.0 permanent uncounted FLOAT_OK=009027319bbb           HOSTID=FLEXID=9-b28520b9
SIGN=123456789012
```

When FLOAT_OK=*server_hostid* is specified on a FEATURE line:

- The server_hostid, is the same hostid that appears on the SERVER line of the license file, or could be a different hostid that refers to the same system. If the SERVER line hostid is ANY, the FLOAT_OK server_hostid should be specific to the system where the license server runs.
- The feature f1 is available only when checked out from a license server running on a system that has the same hostid as that specified by the FLOAT_OK parameter.
- A user can run on the system where the license server runs, but he can use only the license being served by the license server, not the node-locked license. Otherwise, an extra use for each FLOAT_OK license could occur. Note that without the server_hostid specified on the FEATURE line, the FLOAT_OK license can be served from any computer, as long as the FlexNet ID dongle is attached to it.
- The hostid on the FLOAT_OK FEATURE line must be only a single hostid. For multiple FlexNet ID dongles, use individual FEATURE lines for each dongle.

Adding a server_hostid places some extra burden on the user and the license issuer because the system's hostid, as well as the FLEXID hostid, must be specified in the license.

Node-locked to a User Name

If a license is to be used exclusively by one user on different systems, that license can be node-locked to the user's user name. The license file is copied to the different systems on which the user might work; the user's user name must be identical on each system.

For this method to be useful, individual user names in an organization need to be unique. When used in a license file in this way, a user name cannot contain spaces.

Important ▪ *This method is one of the least secure of those methods described in this chapter, but may be appropriate if the user is trusted.*

Fulfilled from a Prepaid License Pool

In this method, the user buys a prepaid number of license-days from the vendor. The user can then fulfill a license using a partial amount of the total license-days for the given borrow period, node-locked to a particular system. For example, in preparation for a business trip (or even during a business trip), the user fulfills a license that expires in five days that is node-locked to their laptop. Each fulfillment can be node-locked to a different system (or even multiple times to the same system), thus allowing mobility of license usage within the prepaid number of license-days.

This is like a pay-per-use pricing model because each fulfillment is made from a decreasing number license-days. It is different than certain pay-per-use models because once node-locked to a system, that system is allowed unlimited use of the application until the license expires. This short-term license cannot be returned early; once fulfilled, those license-days cannot be refunded. Other pay-per-use models charge based on the number of times the application is used.

Note ▪ *This license model requires your license activation/fulfillment system to track the available number of prepaid units.*

License Borrowing using the BORROW Keyword

If a license is to be used on a computer that is intermittently connected to a license server, that license can be issued as a floating license with the `BORROW` keyword. This type of license can be borrowed from a license server via a special checkout and used later to run an application on a computer that is no longer connected to the license server. License borrowing must be enabled by the vendor before a user can borrow licenses.

For license borrowing, a user initiates borrowing and specifies the expiration date when a borrowed license is to be returned to the floating license pool. While still connected to the network, the application is run from the system. This writes licensing information locally into a borrow cache on the system. (On Windows platforms, the borrowed license is cached in the system registry. On Linux platforms, the borrowed license is cached in a local secured data file.) The resulting local licensing information behaves as an uncounted license even if the borrowed license on the server is counted. The system can now be disconnected from the network.

The license server keeps the borrowed license checked out via the "linger" mechanism. The FlexEnabled application automatically uses the borrow cache to do checkouts during the borrow period.

Borrowed licenses can be returned early, that is, before the borrow period expires, if enabled in the vendor daemon via the `ls_borrow_return_early` setting in `machind/lsvendor.c`. Upon the earlier of either the expiration of the borrow period or the early return of a borrowed license, the borrow cache no longer authorizes checkouts and the license server returns the borrowed license to the pool of available licenses. No clock synchronization is required between the license server and the FlexEnabled application.

Note ▪ *While checking out a borrowed license from the borrow cache, the requested license should be in the same case as that of the license borrowed from the license server. Checkout from the borrow cache is case-sensitive.*

Enabling License Borrowing

Procedure for Software Publishers

To enable borrowing in a FlexEnabled application, it must be enabled in the toolkit (it is enabled by default) and in an application's license file:

Task ***To enable borrowing in the production toolkit:***

1. In `machind/lm_code2.h` of the toolkit, ensure that `LM_BORROW_OK` is set to 1:

   ```
   #define LM_BORROW_OK 1
   ```

2. Rebuild the toolkit, if necessary (see the *Development Environment Guide*).
3. To enable borrowing in an application's license file, add the keyword `BORROW[=n]` to the feature definition line for a feature that can be borrowed. `n` is the maximum number of hours that the license can be borrowed. The default maximum borrow period is 168 hours, or one week. For most platforms, a value under 60,000 hours works for a maximum value.

If you want to write an optional interface for your users to implement borrowing from within your application, you must use the Flexible API. See `LM_A_BORROW_STAT` and `LM_A_BORROW_EXPIRE` in the *C/C++ Function Reference* for further details.

Procedure for License Administrators

A license administrator can initiate license borrowing in one of three ways:

- Running the `lmborrow` utility to set `LM_BORROW` in the Windows registry or `$HOME/.flexlmborrow`.
- Setting the `LM_BORROW` environment variable directly
- Using a borrowing interface in an application, if implemented by the developer

See the *License Administration Guide* for detailed instructions a user would use to borrow a license.

Support for License Borrowing

See the *License Administration Guide* for more information about the following utilities and keywords for the options file that support borrowing:

- `lmdown -force`
- `lmstat`
- `lmborrow`
- `BORROW_LOWWATER`
- `EXCLUDE_BORROW`
- `INCLUDE_BORROW`

Security Issues for License Borrowing

To decrease the likelihood that the local secured borrow data is counterfeited, each time a new version of an application is built, link the application with a newly generated version of `lm_new.o` (UNIX) or `lm_new.obj` (Windows) object files. Refer to the *Development Environment Guide* for suggested methods of generating the object file.

The borrowed license state persists in a borrow-data cache managed by the license server. In the event, a license server is stopped and restarted while licenses are borrowed, the restarted license server is refreshed with the contents of the borrow-data cache. This mechanism prevents reuse of currently borrowed licenses. The borrowed license state persists only on the license server from which borrowing is initiated. The persistent state does not transfer to other license servers configured for three-server redundancy.

When a license is borrowed from a license server, the borrow information is recorded in a file/registry on the client side, to be used for subsequent checkouts. This borrow information is encrypted using keys which are automatically generated each time `lmnewgen` is run when you build the FlexNet Publisher Licensing Toolkit. Each run of `lmnewgen` generates keys that are different from the previous run for security purposes. However, this means that a borrow record created by one FlexNet Publisher version of the FlexEnabled client will not be readable by another FlexNet version of the same FlexEnabled client.

For example, if version 1 of your FlexEnabled client is built using FlexNet Publisher v11.5, and version 2 of your product is built using FlexNet Publisher v11.8, then version 2 of your product may not be able to use the borrowed license checked out by version 1 of your product.

This is the default behavior of FlexNet Publisher, which can be overridden by specifying the `-bfixed` option to `lmnewgen` during the toolkit build. Note that this may make the borrow cache encryption more vulnerable, since a single version of the borrow cache can enable multiple versions of your product.

Important ▪ *For `-bfixed` to work, the `LM_A_BORROW_RETURN_BY_VERSION` value must be the same across old and new clients. For clients in the version range 11.14.0 to 11.15.0, the effective `LM_A_BORROW_RETURN_BY_VERSION` value was 1, whereas for clients older than 11.14.0 the effective `LM_A_BORROW_RETURN_BY_VERSION` value was 0.*

For more information about the borrow-data cache management callback functions `ls_borrow_in`, `ls_borrow_init`, and `ls_borrow_out`, see the *C/C++ Function Reference*.

License Transferring Using the Transferrable Counted Model—Windows Only

The Transferrable Counted Model (TCM), a mobile licensing approach, introduced to transfer licenses from a parent licensing server to a remote licensing machine. While using this approach, license counting can be performed after the child server disconnection.

A license can be issued as a transferrable license using the BORROW keyword in a FEATURE line of a license file, if it is intended to be served on a machine that is not connected to a license server, such a license may be distributed to a remote machine via. a parent license server. A transferred license may subsequently be served on a machine which is not connected to the parent license server.

When transferring licenses, a user initializes the transfer and specifies an expiration date for returning the transferred license to the floating licensing pool. On the Windows platforms, a transferred license is cached in the system registry. On the Linux platforms, a transferred license is cached in a local secured data file. The system can now be disconnected from network after transfer. The parent license server keeps the transferred license checked out via. the linger mechanism.

Running a local license server on a remote machine enables to serve the cached transferred licenses and also counted on the server. These licenses are checked out by the FlexEnabled application.

If the vendor daemon in primary machine is enabled via. the `ls_borrow_return_early` setting in the machine/lsvendor.c, transferred licenses can be returned early i.e. before the transfer period expired.

The licensing server returns the transferred license to the pool of accessible licenses when one of the following occurs: the transfer period expired, the license is returned early, or the transfer cache no longer authorizing licenses. No clock synchronization is required between the parent license server and remote machine.

Note ▪ *While checking out a transferred license from the transfer cache through a local license server, the requested license should be in the same case as that of the license transferred from the license server. Usage from the transfer cache through a local license server is case-sensitive.*

Enabling License Transferring

Procedure for Software Publishers

To enable transferring in a FlexEnabled application, publishers need to follow the below:

Task ***To enable license transferring:***

1. Start the parent licensing server with a FEATURE line that contains the BORROW keyword in a license file.
2. Set the lmpath for the parent machine from the remote machine.
3. Set the expiry for transfer of license from remote machine using lmtransfer utility.

   For example, to set the expiry:

   ```
   lmtransfer vendorname -set dd-mmm-yyyy [hh:mm]
   ```

4. Call the **lc_transfer_lic()** API through the lmflex from the remote machine.

   For example, In lmflex file call the **lc_transfer_lic()** API in place of **lc_checkout()**:

```
/* if (lc_checkout(lm_job, feature, "1.0", nlic, LM_CO_NOWAIT, &code, LM_DUP_NONE)) */
 if(lc_transfer_lic(lm_job, feature, "1.0", nlic, &code))
 {
       lc_perror(lm_job, "license transfer failed");
       cleanup(initHandle);
       exit (lc_get_errno(lm_job));
 }
```

5. Start the child server on the remote machine with the following command:

```
lmgrd.exe -c license_file - servTransferred
```

Procedure for License Administrators

A license administrator can initiate license transferring in one of these ways:

- Running the lmtransfer utility.
- Using a transferring interface in an application, if implemented programmatically.

See the *License Administration Guide* for detailed instructions a user would use to transfer a license.

Support for License Transferring

See the ls_server_borrow_reread_interval section for more information.

See the *License Administration Guide* for more information about the *lmtransfer* utility and *C/C++ Function Reference* for more information about the following API functions and and attributes:

- *lc_transfer_lic*
- *lc_transfer_lic_return*
- *LM_A_TRANSFER_LIC*
- *LM_A_SERVEDTRANSFER_EXPIRE*

Considerations While Transferring the License

To decrease the likelihood that the local secured transfer data is counterfeited, each time a new version of an application is built, link the application with a newly generated version of `lm_new.o` (UNIX) or `lm_new.obj` (Windows) object files. Refer to the *Development Environment Guide* for suggested methods of generating the object file.

The transferred license state persists in a transfer-data cache managed by the license server. In the event, a license server is stopped and restarted while licenses are transferred, the restarted license server is refreshed with the contents of the transfer-data cache. This mechanism prevents reuse of currently transferred licenses. The transferred license state persists only on the license server from which transferring is initiated. The persistent state does not transfer to other license servers configured for three-server redundancy.

When a license is transferred from a license server, the transfer information is recorded in a file/registry on the client side, to be used for subsequent checkouts. This transfer information is encrypted using keys which are automatically generated each time `lmnewgen` is run when you build the FlexNet Publisher Licensing Toolkit. Each run of `lmnewgen` generates keys that are different from the previous run for security purposes. However, this means that a transfer record created by one FlexNet Publisher version of the FlexEnabled client will not be readable by another FlexNet version of the same FlexEnabled client.

For example, if version 1 of your FlexEnabled client is built using FlexNet Publisher v11.5, and version 2 of your product is built using FlexNet Publisher v11.8, then version 2 of your product may not be able to use the transferred license checked out by version 1 of your product.

This is the default behavior of FlexNet Publisher, which can be overridden by specifying the `-bfixed` option to `lmnewgen` during the toolkit build. This may make the transfer cache encryption more vulnerable, since a single version of the transfer cache can enable multiple versions of your product.

Limitation

This feature is not supported for the following list:

- Package and Suite
- UPGRADE keyword in options file
- Multiple vendors
- Transferring multiple features at a time
- lmadmin
- lmgrd started in service mode
- Java and C# clients

In future releases, TCM will be supported for all the above lists.

24

FlexNet Publisher Parameter Limits

Limitations such as string lengths are listed here. Items that are unlimited are also listed for clarification. This chapter lists the following:

- Parameter General Information
- License File Limits
- Options File Limits
- lc_set_attr limits
- Other API Limits
- Other API Limits
- Vendor Daemon Limits
- lmadmin (or lmgrd) Limits
- Subnet, Domain, Wide-Area Network Limits
- LM_LICENSE_FILE, *VENDOR_LICENSE_FILE*

Parameter General Information

This section contains the following:

- Internationalization Support
- Path Name Formats

Internationalization Support

FlexNet Publisher provides internationalization support for the following parameter categories:

- Host names
- Display names
- User names
- File names (Windows only)
- Feature and package names
- Contents of `NOTICE`, `ISSUER`, `VENDOR_STRING`, `SN`, `asset_info`, `dist_info`, `user_info`, and `vendor_info` fields
- Comments

These parameters can contain characters in any language and any codepage, including the UTF-8 encoding of Unicode. The only codepages that cannot be represented are the UCS-2 encoding of Unicode and EBCDIC.

See Also
Internationalization Support

Path Name Formats

FlexNet Publisher recognizes the following path name specifications:

- Universal naming convention (UNC)—`\\server\myvendor\bin...`
- MS-DOS naming convention—`C:\myvendor\bin\...`
- UNIX naming convention—`/myvendor/bin/...`

Path name support for any given FlexNet Publisher component is limited to those formats supported by the particular operating system on which the component resides.

The @ symbol cannot be used in path names for license file locations.

On systems where directory names use non-ASCII characters, these characters must be encoded in UTF-8 when defining the default license file location using **lc_set_attr** and `LM_A_LICENSE_DEFAULT`. For example oxE9 for the character é.

License File Limits

The limits on names for the major parameters employed in the license file are shown in the table below:

Table 24-1 ▪ Parameter Limits

Parameter	Limit
Max. number of VENDOR lines	Unlimited
Vendor daemon name length	10 bytes
Display name length	1024 bytes
Host name length	1024 bytes

Table 24-1 ▪ Parameter Limits

Parameter	Limit
User name length and format	1024 bytes User names cannot contain spaces.
Maximum number of FEATURE/INCREMENT/ UPGRADE lines allowed	Unlimited
Maximum length of each FEATURE/INCREMENT/ UPGRADE/PACKAGE	4096 bytes
FEATURE name length	30 bytes
Version (including date-based version)	10 characters, in floating point format For example: `123.4567, 2.10, 2016.12`
Latest expiration date	31-dec-9999 Using `permanent` is recommended to indicate that a license does not expire.
License count in FEATURE/INCREMENT/UPGRADE/ PACKAGE lines	1 through 2,147,483,647
Maximum total license count for a given feature (accounting for all FEATURE and INCREMENT lines for that feature in the same license pool)	2,147,483,647
Number of users	1 through 2,147,483,647
OVERDRAFT=*n*	1 through 2,147,483,647
HOST_BASED=*n*	1 through 2,147,483,647
USER_BASED=*n*	1 through 2,147,483,647
MINIMUM=*n*	1 through 32767
BORROW=*n*	1 through 2,147,483,647
Other optional FEATURE attributes (for example, NOTICE=, VENDOR_STRING=)	Limited only by the total length of the `FEATURE` line.
Comment line	4096

Note ▪ *Support for maximum parameter sizes can be limited when those parameters are used as arguments on the FlexNet Publisher utility command line. The operating system may impose limits on the total command line length, reducing the maximum size that can be used for parameters.*

There is a limit on the number of license files that can be used by a license server manager. This limit is on the number of characters in the combined license file paths:

- **UNIX**—163840 characters
- **Windows**—20800 characters

When this limit is exceeded error -184 is reported: see "Flexible API Error Return Values" in the *C/C++ Function Reference*.

This limit also applies to the license search paths used by FlexEnabled applications. When the license search path exceeds the maximum, a license checkout will fail with error -184. For details of how license file locations are specified, see Locating the License File.

Options File Limits

The line length limit is 4,000 characters. There are no other string size limitations on anything in this file. Note that `GROUP`s can be made arbitrarily large by listing the `GROUP` more than once: FlexNet Publisher concatenates such entries.The total number of reservations supported per options file is 10000.

lc_set_attr limits

The table below shows the **lc_set_attr** limits:

Table 24-2 ▪ lc_set_attr Limits

Function	Limit
LM_A_DISPLAY_OVERRIDE	1024 bytes
LM_A_HOSTNAME_OVERRIDE	1024 bytes
LM_A_USERNAME_OVERRIDE	1024 bytes
LM_A_CHECKOUT_DATA	1024 bytes
LM_A_MULTIPLE_CHECKOUT_DATA	1024 bytes
LM_A_CHECK_INTERVAL	>20 seconds

Other API Limits

The table below shows additional API limits:

Table 24-3 ▪ Additional API Limits

API	Limit
Maximum length for entire specification of vendor-defined hostid: *`LABEL=hostid`*	1024 bytes
Maximum number of licenses in one lc_checkout request	4,294,967,296
Maximum long error message length	1024 bytes (length of string returned from **lc_errstring**)

Vendor Daemon Limits

Number of Applications Per Vendor Daemon

When using TCP/IP ports, each FlexEnabled application connected to a vendor daemon uses one or more sockets. The number of sockets any one FlexEnabled application requires is dependant on FlexNet Publisher implementation details; each job handle consumes one socket. The number of sockets available to the vendor daemon is defined by the per-process system limit for file descriptors. The total number of sockets used by the license server `lmadmin` (or `lmgrd`) and the vendor daemon together is slightly larger than the total number needed by the FlexEnabled applications.

In practice, we encourage users to put license servers on systems configured with enough file descriptors per process to support the number of FlexEnabled applications connecting to the vendor daemon, which may require reconfiguring the kernel to increase the number of file descriptors per process.

Multiple vendor daemons can be run on a single network, making the number of TCP/IP systems effectively unlimited.

Number of Vendor Daemons Per Node

A *particular* vendor daemon can only be run once per node. This is a security mechanism to prevent extra licenses from being granted.

There is no limit to the number of *different* vendor daemons that can be run per node.

lmadmin (or lmgrd) Limits

The following table shows the `lmadmin` (or `lmgrd`) limits.

Table 24-4 ▪ `lmadmin` (or `lmgrd`) Limits

Description	Limit
lmadmin or lmgrd processes per node	Unlimited
Default port number range	27000-27009 If the value **0** is specified, `lmadmin` (or `lmgrd`) uses one of the available default ports. ***Note ▪*** *Those producers concerned about potential DoS attacks based on knowledge of common license server ports may want to consider specifying a server port outside the range 27000 to 27009.*
License files per lmadmin or lmgrd process	Unlimited

Subnet, Domain, Wide-Area Network Limits

FlexNet Publisher has no limitations regarding subnets (because it does not use *broadcast* messages).

If the host name in the license file is fully qualified (`name.domain.suf`) or is an IP address (`###.###.###.###`), then there are no limitations with regard to Internet domains.

There are no other limitations regarding wide-area networks.

LM_LICENSE_FILE, *VENDOR*_LICENSE_FILE

The following table shows the **lm_license_file** and **vendor_license_file** limits.

Table 24-5 ▪ License Path Limitations

Description	Limit
Number of licenses in path	Unlimited

25

Obsolete Features

The variables, and features listed in this section are obsolete, but are still provided in the toolkit for backward compatibility. Their functionality has been replaced with more current features; where possible, that current feature is noted in the tables below.

Obsolete Vendor Daemon Variables

The following vendor daemon variables are obsolete, and exist only for compatibility with earlier FlexNet Publisher versions. They should be used with care—contact Revenera Technical Support if you have questions about their use.

Table 25-1 ▪ Obsolete Vendor Daemon Variables

Variable	Description
ls_a_behavior_ver	Sets the behavior of the FlexEnabled application to the given version of FlexNet Licensing.
ls_a_crypt_case_sensitive	Controls license file case sensitivity in the vendor daemon with license keys. See: ● ls_a_license_case_sensitive
ls_a_lkey_long	Enables 64-bit license keys. License keys are obsolete. See: ● SIGN
ls_a_lkey_start_date	Enables embedded start dates in license keys. License keys are obsolete. See: ● START
ls_enforce_startdate	Enforces start date to be used in check outs. See: ● START

Table 25-1 ▪ Obsolete Vendor Daemon Variables

Variable	Description
ls_send_oldest_signature	Used with cross-version license files and affects which signature is returned when the version of the FlexNet Publisher client library linked into the FlexEnabled application does not match that of any available license signature. See **ls_single_xver_signature**.
ls_tell_startdate	Compares start date with system date.
ls_use_featset	Enables requirement of `FEATURESET` line. `FEATURESET` is obsolete.
ls_ba_heartbeat_interval	Enables Hostid validation interval for AMZN Cloud server hostid. The interval is specified in minutes; means no heartbeat.
ls_ba_heartbeat_retry_count	Controls the number of retry count for AMZN Cloud server hostid validation.
ls_ba_heartbeat_retry_interval	Enables Hostid validation retry interval for AMZN Cloud server hostid. The retry interval is specified in minutes; means no heartbeat

Obsolete License File Features

The following obsolete license file features are described:

- License Key Length and Start Date
- FEATURESET Line
- Intel Pentium III Hostid (HOSTID_INTEL)

License Key Length and Start Date

This section applies only to licenses generated with license keys rather than signatures (`SIGN=` keywords).

The license key is the set of hexadecimal digits which appear on `FEATURE/INCREMENT/UPGRADE/PACKAGE` lines and authenticates the text, making the line secure.

For example:

```
FEATURE f2 demo 1.0 permanent uncounted 6E06CC47D2AB HOSTID=1234
                                        ^^^^^^^^^^^^
                                               license key
```

If a vendor daemon or client library is configured to authenticate a license using the license key (`LM_STRENGTH_LICENSE_KEY`), those components can authenticate the license using either the 12 or 20-character license key. That is, a vendor daemon or client library configured for non-license key (`LM_STRENGTH_DEFAULT`, or `LM_STRENGTH_`{133,163,239}BIT) will not authenticate the license with either size of license key; those components will only authenticate the license using their respective value of the SIGN= field.

The license key is 12 characters by default. Previous to version 6, license keys are 20 characters; they are also referred to as long license keys. Both 12- and 20-character license keys are accepted. 20-character license keys can be enabled, via one of:

- `lc_set_attr(job, LM_A_LKEY_LONG, (LM_A_VAL_TYPE)1)`
- `lc_set_attr(job, LM_A_BEHAVIOR_VER, (LM_A_VAL_TYPE)` *behavior*) where *behavior* is `LM_BEHAVIOR_V5_1` or less

12-character license keys impact licensing in two ways:

- Instead of a 64-bit security key on each feature line, there is a 48-bit security key.
- A start date in not embedded in the license key.

You can specify a start date in two ways:

- For both 12- and 20-character license keys, use the optional "START=" attribute for `FEATURE/INCREMENT/UPGRADE` lines. This is the preferred method for a start date.

For 20-character license keys, embed the start date in the license key.

Implementing Long License Keys and Embedded Start Dates

Here is how to turn on long license keys with embedded start dates in applications, license generators, and vendor daemons:

- In an application, set long license keys with:

  ```
  lc_set_attr(job, LM_A_LKEY_LONG, (LM_A_VAL_TYPE) 1);
  ```

  and embedded start dates with:

  ```
  lc_set_attr(job, LM_A_LKEY_START_DATE, (LM_A_VAL_TYPE) 1);
  ```

- Make the same changes to the `lmcrypt.c` source in the `machind` directory.
- For the vendor daemon (`lsvendor.c` in `machind` directory) set:

  ```
  ls_a_lkey_long = 1;             /* long license keys */
  ls_a_lkey_start_date = 1;       /* hidden start dates */
  ```

After these changes are made, 20-character license keys with embedded dates are generated and 12-character license keys are not generated. The start date is determined in one of three ways:

- To use the date the license is digitally signed; use **`0`** for the license key field in the feature definition line.

  Example:

  ```
  FEATURE f2 demo 1.0 permanent uncounted 0 HOSTID=1234
  ```

- To specify a start date other than the date the license is signed, use

  ```
  start:dd-mmm-yyyy
  ```

  for the license key field in the feature definition line.

  Example:

  ```
  FEATURE f2 demo 1.0 permanent uncounted start:03-jan-2013 HOSTID=1234
  ```

- To keep the date the same in an already existing license key, leave the license key as is.

  ```
  FEATURE f2 demo 1.0 permanent uncounted 6E06CC47D2AB30542134 HOSTID=1234
  ```

Compatibility Issues

- Version 6 applications (even those accepting 12-character license keys) will accept licenses with long, 20-character license keys.
- Applications prior to version 6 will not accept licenses with 12-character license keys.
- License generators, like `lmcrypt`, issue long license keys when `verfmt` is set to a version less than 6.
- `LM_VER_BEHAVIOR`, in `machind/lm_code2.h`, set to `LM_BEHAVIOR_V5_1` (or older) will set license keys to be long and start dates in the license keys. However, this can be overridden in the code with **lc_set_attr(**`job, LM_A_LKEY_LONG, (LM_A_VAL_TYPE)0`**)** and **lc_set_attr(**`job, LM_A_LKEY_START_DATE, (LM_A_VAL_TYPE)0`**)**, which must be set in the application, license generator, and vendor daemon.

Existing companies can successfully use short license keys (and may very well want to), but must obey the following rules:

- If a site wants to use older products, then you must use `-verfmt` to create a license with long keys. Both old and new products will accept these licenses.
- If a site is completely converting to products using FlexNet Publisher version 6.0 or later, licenses with short keys can be shipped.
- New users can receive licenses with short keys.

FEATURESET Line

The use of `FEATURESET` is discouraged. The `FEATURESET` line is required only if `ls_use_featset` is set in `lsvendor.c`.

`FEATURESET` *vendor key*

where:

Table 25-2 ▪ FEATURESET Line

Line	Description
vendor	Name of the vendor daemon used to serve at least some features in the file.
key	License key for this `FEATURESET` line. This license key encrypts the license keys of all `FEATURE` lines that this vendor daemon supports, so that no `FEATURE` lines can be removed or added to this license file.

The `FEATURESET` line allows the vendor to bind together the entire list of `FEATURE` lines supported by one vendor daemon. If a `FEATURESET` line is used, then all the `FEATURE` lines must be present in the same order in the license file. This is used, for example, to insure that a user uses a complete update as supplied, without adding old `FEATURE` lines from the vendor.

See Also
The License File: Overview

Intel Pentium III Hostid (HOSTID_INTEL)

Requirements

- Windows
- CPU hostid must be enabled

Note ▪ *In May 2000, Intel announced their intention to discontinue support for CPUID.*

Enabling the CPU Hostid

On most systems, this is enabled in the BIOS Setup, which you usually enter by pressing the DEL key when the system is first booting up. If this is unavailable, it likely means that the system is not a Pentium III or higher.

Hostid Length

The true CPUID is a 96-bit value, in the format—

```
####-####-####-####-####-####
```

—where the #s are uppercase hexadecimal characters. According to Intel, all 96 bits (24 hexadecimal characters) are required to achieve a nearly unique hostid. It is likely, however, that using the last 16 or 8 hexadecimal characters are very nearly unique. Therefore, it is recommended that unless absolute uniqueness is required, the 32-bit format should normally be used so that the license file is shorter and more readable. The 64-bit version is a compromise between the two.

The required length is determined by what is put in the license file. If you want to use 96-bit CPUID, then that is what should go in the license.

Converting from 96-Bit to 32-Bit

The 32-bit hostid is simply the last nine characters from the 96-bit version. Similarly, the 64-bit is the last 19 characters:

Table 25-3 ▪ 96-bit to 32-bit Conversion Information

Length	Example
96-bit	1B34-A0E3-8AFA-6199-9C93-2B2C
64-bit	8AFA-6199-9C93-2B2C
32-bit	9C93-2B2C

lmtools and lmhostid

`lmhostid` takes the following arguments:

Table 25-4 ▪ `lmhostid` Arguments

Argument	hostid Description
-cpu	32-bit hostid
-cpu32	32-bit hostid
-cpu64	64-bit hostid
-cpu96	96-bit hostid

Table 25-5 ▪ Obsolete Keywords

Keywords	hostid Description
LMB_	baremetal hostids
VMW_UUID	Virtual Machine HostID
HPV_UUID	Hyper-V HostID

Security Issues

Where available, the CPUID is the preferred hostid, because it is likely to be the most secure hostid. Revenera has taken extra precautions in the applications and vendor daemons to make this hostid especially secure.

It is not believed that the CPUID length is important to security. A duplicate 32-bit or 64-bit hostid will likely be so rare as to be insignificant.

lmbind

lmbind was used for bare-metal binding on license servers running in virtual machines. It and all the keywords it supported (LMB_*) are now deprecated. For more details refer to previous version documents.

Note ▪ *Support for lmbind has now been completely removed from FlexNet Publisher (release 11.14.0 onwards).*

Obsolete Libraries

UNIX Libraries for Single-Threaded Applications

The following UNIX libraries for single-threaded applications have been deprecated in FlexNet Publisher (release 11.15.1 onwards):

- `liblmgr_nomt_pic.a`
- `liblmgr_nomt_pic_trl.a`
- `liblmgr_nomt.a`
- `liblmgr_nomt_trl.a`

Non-TRL Libraries

The following non-TRL libraries have been deprecated in FlexNet Publisher (release 11.15.1 onwards):

- **Windows**—`lmgr.lib`
- **UNIX**—`liblmgr.a`, `liblmgr_pic.a`

Glossary

The following glossary terms are used in this product.

Term	Meaning
activatable license	A license right in trusted storage that can be activated to a FlexEnabled client. See also license group.
activated feature line	See feature definition line.
activation	In the FlexNet Publisher Licensing Toolkit, this is a process used to load license rights into trusted storage. Different activation types include: ● programmatic activation ● manual activation ● short-code activation ● local activation In FlexNet Operations, this is a process used to generate license rights in either trusted storage and in license files.
Activation API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that enable you to initiate transactions and manage license rights in trusted storage. The Activation API is used to develop the functionality in an activation utility.

Term	Meaning
activation client	The system that works with an activation server to receive license rights. This system can be either a license server or a FlexEnabled client. The activation client initiates an activation, repair, or return transaction with an activation server, using an activation utility. The following transaction types can be initiated by an activation client: ● activation transaction ● return transaction ● repair transaction
activation request	A message sent by an activation utility to an activation server that initiates an activation transaction.
activation response	A message sent from an activation server to an activation client, in response to an activation request. This message defines the license rights.
activation server	A system that responds to requests for license rights from an activation client. The activation server can be one of the following: ● a license server ● FlexNet Operations ● in a test environment, the Manual Response Generator (responsegen.exe) An activation server can also perform repair and return transactions.
activation specification record (ASR)	A signed XML file used with local activation and short-code activation that contains information about license rights, configuration settings for the publisher's trusted storage segment, and additional information required by the activation transaction.
activation transaction	A sequence of activation request and activation response messages, and related actions that result in license rights being loaded into trusted storage.
activation utility	A utility that initiates an activation transaction, repair transaction, or return transaction by sending a request to an activation server. This utility also processes the response received from the activation server. It uses the Activation API to perform these capabilities. It is written by the software publisher to meet specific business process requirements. This application can be either separate from or embedded in the FlexEnabled application.
anchor	A link between the system and the trusted storage that is used to detect whether trusted storage has been tampered with, deleted, or restored. Anchors are also used to ensure that information in trusted storage about trial licenses persists beyond the expiration date.
ASR file	See activation specification record (ASR).

Term	Meaning
binding	A capability used with trusted storage where one or more properties of a system, combined together into a unique identifier, are used to ensure that license rights held in trusted storage are not copied to another system.
borrowing	A capability available in FlexEnabled applications that enables a user to temporarily obtain a license in such a way that the license can be used locally for a limited period of time while the user is disconnected from the local network.
bulk entitlement	An entitlement created for an order of multiple copies of a single product or suite. Bulk entitlements are used for sales through reseller channels.
checkin	The transaction that a FlexEnabled application performs to return a license after it has been checked out. This operation is not the same as an activation transaction. FlexEnabled applications must perform a checkout and checkin transaction to use a license.
checkout	The transaction that a FlexEnabled application performs to request a license. This transaction is different than activating a license. FlexEnabled applications must perform a checkout and checkin transaction to use a license.
client system	A general term used to define any system that interacts with a server in a client-server relationship. When using the FlexNet Publisher Licensing Toolkit there are two types of clients: ● activation client ● FlexEnabled client
composite transaction	Available in FlexNet Publisher 11.8, composite transactions allow multiple actions to be combined into a single transaction with FlexNet Operations. Also called multi-action or V2 transactions. From FlexNet Publisher 11.14.0, composite transactions are also supported for all license servers.
concurrent license	A license that can be shared among users by allowing each user to check out and then return the license. These licenses are served to FlexEnabled applications from a license server. If all licenses are being used, an additional user cannot run the FlexEnabled application until one of the other users check in their license. When one user finishes using the license, another user can begin using it. Also called floating license or served license rights.
config response	A response to any request from an activation client that contains information about how the publisher's trusted storage segment must be configured (called the trusted storage segment configuration). This response is processed by the activation utility and results in a segment of trusted storage being created or re-configured for that publisher.
consumer	An individual or small business who purchases a license for their individual use. Compare with enterprise.

Term	Meaning
counted	A type of license right or license model that defines a multiple quantity of licenses and allows concurrent use of the licenses. This term is taken from the fact that the license server counts the number of licenses that are being used.
debug log file	A file used by the license server to record status and error messages that are useful for debugging the license server. Each license server can have one or more of these files.
demo license	See trial license.
duplicate grouping	A capability in the FlexNet Publisher Licensing Toolkit used to uniquely identify a single client (or client checkout request) for the purpose of counting the number of licenses that are used. Duplicate grouping defines a set of rules under which multiple checkout requests can share the same license(s).
embedded activation	See local activation.
encryption seed	A set of values defined by a software publisher that are used when creating a publisher-specific vendor daemon.
end-user	A person who uses a FlexEnabled application. This role is not the same as a license administrator.
end-user system	See client system.
end-user license administrator	See license administrator.
enterprise	An organization that purchases licenses on behalf of a number of individual users. Compare with consumer.
entitlement	A representation of the license model and product that a customer has bought. It is a definition only. The customer will need to fulfill the entitlement so that the license rights can be populated to license files or trusted storage. In FlexNet Operations, there are two types of entitlements: ● bulk entitlement ● simple entitlement Each entitlement is uniquely identified by an entitlement ID.
entitlement ID	In the FlexNet Publisher Licensing Toolkit, an attribute in trusted storage that stores a value used to identify the customer's entitlement. In FlexNet Operations, a value that uniquely identifies the entitlement.
expiring license	License rights granted to a customer for a limited duration. When the license rights have expired, the licenses can no longer be checked out.

Term	Meaning
feature	In the context of a software application, this is a single unit of capability. In the FlexNet Publisher Licensing Toolkit, this is an identifier used to associate a license model to a unit of capability in a software application. The software publisher defines which features in their software application map to a single feature in the FlexNet Publisher Licensing Toolkit. For example, a feature in the FlexNet Publisher Licensing Toolkit could represent any of the following: ● The entire software application. ● A subset of menu items. ● A specific set of data available from the software application. ● A process in the software application. The software publisher also defines the license model for each FlexNet Publisher Licensing Toolkit feature. This gives software publishers the ability to define a different license model for each feature in the software application.
feature bundle	A collection of features in FlexNet Operations. It exists primarily to enable modularity and re-use.
feature definition line	An entry in a license file or fulfillment record that describes a feature. Each line begins with the keyword FEATURE, INCREMENT, or UPGRADE. In license files, feature definition lines are used with SERVER and VENDOR lines to define the license model. In trusted storage, feature definition lines are used with the fulfillment record properties to define the license model.
feature line	See feature definition line.
file-based activation	See manual activation.
FlexEnabled application	A software application that uses the FlexNet Publisher Licensing Toolkit to implement its license models.
FlexEnabled client	A client system where a FlexEnabled application is installed.
Flexible API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that allow FlexEnabled clients to access license rights.
FlexNet Agent	A utility that runs on a license server and allows communication between the license server and other FlexNet products.
FlexNet Manager	This is a license server management and license management decision support solution in the FlexNet product family that allows companies to centrally manage license servers and license usage across the enterprise.
FlexNet Operations	A product in the FlexNet family of products that manages entitlements and generates license rights.

Term	Meaning
floating license	A license that can be shared among users and is served to FlexEnabled applications from a license server. When a user finishes using the license, another user can begin using it. Also called concurrent license or served license rights.
fulfill	The action of getting license rights.
fulfillment record	In the FlexNet Publisher Licensing Toolkit, a data structure specific to trusted storage that defines license rights.
heartbeat	A type of message sent periodically between two systems (for example, a FlexEnabled application and a license server) to identify whether each is running and able to communicate with the other.
hop count	In the FlexNet Publisher Licensing Toolkit, the number of times that license rights in a single fulfillment record have been transferred from one license server to another.
hostid	In the FlexNet Publisher Licensing Toolkit, the value of a specific system attribute that uniquely identifies the host. For example, if IP address is a hostid type, then the hostid might be 129.87.33.101. Hostids are used in node-locked license.
hostid type	A type of attribute that is available to uniquely identify a system. For example, disk serial number, IP address, or MAC address.
hybrid license	In the FlexNet Publisher Licensing Toolkit, a type of license right in the license group on trusted storage that can function as either a concurrent license or as an activatable license.
internet activation	See programmatic activation.
keyboard activation	See short-code activation.
license administrator	A person in an enterprise who is responsible for installing the FlexEnabled application and administering the licenses and license servers.
license file	A text file, usually with the `.lic` extension, that contains license certificates from one or more publishers.
license-file list	See license search path.
license generator	A general term used to describe any mechanism a software publisher uses to generate license rights for their customers. The following FlexNet capabilities and products can function as a license generator: ● FlexNet Operations: generates license rights for trusted storage and license files. ● lmcrypt: generates license rights for license files.

Term	Meaning
license group	A mechanism used to describe licenses rights held in server-side trusted storage. These groups are not used with license rights held in license files. ● **activatable**—A license right in trusted storage that can be activated to a FlexEnabled client. ● **concurrent**—license rights in this group can be shared among users by having a FlexEnabled client to check out, and then return the license. ● **hybrid**—these license rights have the same characteristics as both the activatable license group and concurrent license group. They can be activated to client-side trusted storage on a FlexEnabled client or checked out by a FlexEnabled client. Although you see these group names used to describe license rights in client-side trusted storage, the capabilities of the license rights do not change. Once license rights are in client-side trusted storage, you cannot transfer, serve, or activate them to another system. The FlexEnabled application can only check out and check in these licenses. The group names are used when a license right is returned to the license server, so that it is assigned to the correct license group on the license server.
license model	The set of characteristics that determine the conditions under which (for example, how, when, where, and by whom) a FlexEnabled application can be used.
license rights	Information in trusted storage or license files that defines the license model and controls user access to a specific FlexEnabled application.
license search path	A property that the FlexEnabled application uses to locate where license rights are stored. Sometimes known as a license-file list.
license server	A set of files from the FlexNet Publisher Licensing Toolkit, that work together to manage and serve licenses to FlexEnabled applications. A license server is required to support a concurrent license model. The license server consists of a license server manager, one or more vendor daemons, zero or more options files, zero or more debug logs, and, optionally, a report log. These components are available in the FlexNet Publisher Licensing Toolkit. If the license server supports license rights held in trusted storage, it also includes an activation utility. It may, optionally, act as an activation server.
license server system	See license server.
license server manager (lmadmin or lmgrd)	A license server manager `lmadmin` or `lmgrd` executable that forwards connections from a FlexEnabled application to the correct vendor daemon. Because the vendor daemons are responsible for authenticating requests and serving licenses, a single license server manager can be used with multiple vendor daemons on the license server.

Term	Meaning
license sharing	A capability available in the FlexNet Publisher Licensing Toolkit where multiple license requests from the same user, host, or display results in only one license being used. Also known as duplicate grouping.
licensing client	See FlexEnabled client.
lmadmin	The license server manager introduced in version 11.6 which includes a Web-based GUI.
lmgrd	A command-line-based license server manager.
local license rights	License rights that are stored on the same system where the FlexEnabled application is running, rather than on a license server.
local activation	A type of activation where license rights are loaded using an ASR file. The ASR file must reside with the FlexEnabled application. This does not require user intervention.
machine ID	A value, generated using attributes of the system, that is used to bind trusted storage to that system.
maintenance	A maintenance represents a customer's right to obtain updates and upgrades to a particular licensed product or suite for a specified duration. Used in FlexNet Operations.
manual activation	An type of activation where license rights are transmitted from an activation server to another system using a set of request and response messages that are formatted as XML. These messages are saved to and read from physical XML files. An activation utility loads the license rights from an XML file into trusted storage.
Manual Response Generator (responsegen.exe)	A tool included in the FlexNet Publisher Licensing Toolkit that enables software publishers to test the activation process offline using XML request and response messages stored as files.
node-locked license	A license that can be used only when the FlexEnabled application is run on a specific system or by a specific user (as defined by the hostid).
options file	A configuration file available on the license server that license administrators can use to either change license server behavior or implement certain licensing policies. Each software publisher has its own separate options file that works with a specific vendor daemon.
orderable	Anything that can be purchased by a customer. Orderables can include suites, products, documentation, and maintenances.

Term	Meaning
package	In the FlexNet Publisher Licensing Toolkit, a set of features that can be grouped and licensed together.
part number	A unique identifier, typically correlated with an order system, assigned by a publisher to a product.
permanent license	License rights granted to a customer that do not expire.
product	Anything that can be sold by a publisher and can be licensed.
product ID	In the FlexNet Publisher Licensing Toolkit, one of the values that uniquely identifies the license rights in trusted storage. See also entitlement ID.
programmatic activation	A type of activation where license rights are transmitted from one system to another across a network using a set of request and response messages that are formatted as XML. This process is the identical to manual activation, except that it does not generate a physical XML file. An activation utility loads the license rights from the XML message into trusted storage. This is an automated process and does not require user intervention.
publisher ID	A value used by trusted storage to uniquely identify the software publisher's logical and physical licensing components. Revenera distributes this identifier to software publishers. It is used to prepare certain FlexNet Publisher Licensing Toolkit files that access trusted storage. This value is not the same as the vendor daemon.
quorum	A condition specific to three-server redundancy that is met when at least two of the three license servers are running and can communicate with each other using heartbeats.
rehost	The process of reissuing a license for a different system. This process requires software publishers to perform certain activities to redefine the license rights so that they will work properly on the new system.
repair request	A message sent by an activation utility to an activation server that initiates a repair transaction. The activation server provides the information that repairs the license rights in trusted storage. An activation utility is an application that initiates the repair transaction.
repair response	A message sent from an activation server to an activation client, in response to a repair request. This message contains information that is used to the repair the license rights.
repair transaction	A sequence of repair request and repair response messages, and the related actions that result in license rights being repaired in trusted storage.
report log file	A file that runs on a license server that contains data about the features used by a single vendor daemon. Report logs are encrypted and cannot be read by a person, but are used by FlexNet Manager to produce reports.

Term	Meaning
responsegen	See Manual Response Generator (responsegen.exe).
return request	A message sent by an activation utility to an activation server that initiates a return transaction. The activation server receives the license rights returned by an activation client. An activation utility is an application that initiates the return transaction.
return response	A message sent from an activation server to an activation client, in response to a return request.
return transaction	A sequence of return request and return response messages, and the related actions, that result in license rights being returned from trusted storage to the activation server.
served license rights	License rights that are stored on a license server and are checked out when needed by FlexEnabled applications that run on client systems. These license rights are used for concurrent licensing.
short code	A multi-character code used with a short-code activation transaction, a repair transaction, or a return transaction.
short-code activation	A type of activation where license rights are loaded from an ASR file using an activation transaction that requires a specific code (called a short code). Often the user enters this code into a prompt from the application. This is also is used for licensing via automated telephony.
signature	A secure multi-character string added to the license certificate that ensures it has not been modified.
simple entitlement	An entitlement for a single, specific customer.
single-action transaction	This is the default transaction type for FlexNet Publisher versions 11.7 and earlier. Each transaction can include only a single action. Also known as legacy or V1 transactions. From FlexNet Publisher version 11.14, single actions are deprecated.
suite	A set products grouped together that are functionally complementary and associated with one or more license models.
supersede	A capability in the FlexNet Publisher Licensing Toolkit, used to replace the characteristics of old license rights with new characteristics.
three-server redundancy	A capability in the FlexNet Publisher Licensing Toolkit that enables license administrators to configure three license servers in a failover cluster.
triad	A set of three license servers which operate together to support three-server redundancy.

Term	Meaning
trial license	License rights granted to a user for a limited duration so that the user can evaluate the software application.
trusted storage	A tamper-proof area on the system that stores license rights. This storage location is different than license files and requires a different mechanism for loading and managing license rights. Compare with license file.
trusted configuration	A set of attributes that define the security configuration of a segment of trusted storage. Each software publisher can configure their segments of trusted storage differently. This configuration defines the settings for binding, anchoring, and windback detection.
trusted configuration file	A file that stores the attributes that define a software publisher's trusted storage segment configuration.
trusted storage configuration	See trusted configuration.
uncounted	A license right attribute that allows an unlimited number of concurrent uses of a FlexEnabled application. Compare with counted.
unserved license rights	These are license rights that are not served from a license server. These license rights are located with the FlexEnabled application. Also known as local license rights.
upgrade	A process where a user installs a newer version of a FlexEnabled application and also updates the license rights so that they can use this newer version.
upsell	The process of transitioning from a product of lesser functionality to one of greater functionality in the same product line.
vendor daemon	One of the files that is a part of the license server. This executable is customized and built by the software publisher using files in the FlexNet Publisher Licensing Toolkit. The vendor daemon is responsible for communicating with the FlexEnabled application and issuing licenses.
Vendor Certificate Generator	A publisher-built component, that is a part of FlexNet Operations, that creates license certificates or signed feature lines in fulfillment records. This functions similarly to lmcrypt.
vendor name	A value used to uniquely identify a software publisher's logical and physical licensing components. It is used to prepare certain files in the FlexNet Publisher Licensing Toolkit that manage license rights in both trusted storage and license files. Revenera distributes this identifier to software publishers. This value is not the same as the publisher ID.
windback	A condition where the system clock has been set to an earlier date or time. One reason users change the system clock is to extend the license duration.

Term	Meaning
windback detection	A capability in the FlexNet Publisher Licensing Toolkit that detects whether a user has reset the system clock. This prevents a user from extending the duration of the license.
windback tolerance	An acceptable amount of windback to a system clock that won't trigger windback detection. A certain tolerance is normally allowed to account for time zone differences or Daylight Saving Time.

Index

Symbols

@host 24

A

activatable license 253
activated feature line, See feature definition line
activation 253
    short code 262
Activation API 253
activation application 254
activation client 254
activation request 254
activation response 254
activation server 254
activation specification record 254
activation transaction 254
anchor 254
ANY hostid 167
application attributes for migration to TRL 152
asset_info 131
automatic heartbeats 52

B

Batch checkout 219
billing for license usage 143
binding 255
BORROW 121
borrowing 236, 255
building vendor daemon
    UNIX 32
    Windows 32
bulk entitlement 255

C

CAPACITY 121
checkin 255
checkin callback, vendor-defined 64
checkin filter, vendor-defined 64
checkout 255
    example using LM_CO_AVAIL_NOWAIT flag 213, 223
    example using LM_CO_NOWAIT flag 212
checkout denials
    avoiding 209
checkout filter, vendor-defined 65
checkout option flag
    LM_CO_AVAIL_NOWAIT 209, 219
    LM_CO_NOWAIT 209
checkout, COAVAIL See COAVAIL 209
chroot and ls_do_checkroot 78
client heartbeats
    and TIMEOUT 188
Client reconnection, with COAVAIL 210, 222
client system 255
cloud-licensing, hostids 169
COAVAIL
    client reconnection 210, 222
    example using LM_CO_AVAIL_NOWAIT flag 213, 223
    introduction 209
    limitations 216, 227
    usage examples 210, 222
common vendor daemon 85
    benefits 85
    configuring 90
    deploying 94
    license files 92
    planning 88
    terminology 85

COMPONENTS 133
components, for license file-based licensing 13
COMPOSITE 171
    hostid 167
composite hostid 171
    defining 171
    initializing in application 171
    initializing in vendor daemon 172
    issuing the license 175
    steps to create 171
composite transaction 255
concurrent license 255
config response 255
configuration, signature 146
configuring vendor daemon 31
consumer 255
counted license 256
CPUID hostid 249
customizing
    FlexNet Publisher behavior 187
    vendor daemon 32

D

daemon, vendor 263
date-based version 120
debug log file 256
    managing 33
    starting 31
default license search path 23
default license server ports 111
DEMO hostid 167
demo license, See trial license
detecting OVERDRAFT usage 144
disabling
    lmdown 29
    lmremove 29
DISPLAY
    hostid 168
dist_info 131
domain limits 243
DUP_GROUP 121
dup_group 138
duplicate grouping 256
    and PACKAGE SUITE 136, 138
    suite 136, 138

E

embedded activation, See local activation
embedded start date 247
encryption seed 256
ENCRYPTION_SEEDS 146
end user 256
END_LICENSE 24, 106
end-user license administrator, See license administrator
end-user machine, See client system
enterprise 256
entitlement 256, 259
entitlement ID 256
environment variables
    LM_LICENSE_FILE 40
    VENDOR_LICENSE_FILE 40
eport
    VENDOR line 113
error codes, for LM_CO_AVAIL_NOWAIT flag 216, 226
error messages
    limits 242
example application
    Flexible API 19
expiration date 120
expiring license 256

F

feature 257
feature bundle 257
feature definition line 253, 257
feature definition lines 101
FEATURE line
    asset_info 131
    authentication 102
    BORROW 121
    CAPACITY 121
    COMPOSITE 171
    dist_info 131
    DUP_GROUP 121
    expiration date 120
    feature name 119
    FLOAT_OK 122
    HOST_BASED 122
    HOSTID 123
    ISSUED 124
    ISSUER 124
    license count 120
    MINIMUM 124
    NOTICE 124
    ONE_TS_OK 124
    OVERDRAFT 125
    PLATFORMS 125
    signature 126
    SN 126
    sort 132
    START 126
    SUITE_DUP_GROUP 126
    SUPERSEDE 127
    SUPERSEDE_SIGN 128
    syntax 114

TS_OK 128
USER_BASED 130
user_info 132
vendor name 119
vendor_info 132
VENDOR_STRING 130
version 119
feature name 119
FEATURE/INCREMENT line, See feature definition line
FEATURESET 248
file
license 258
options 260
report log 261
file-based activation, See manual activation
FlexEnabled application 257
FlexEnabled client 257
Flexible API 257
Flexible API example 19
FLEXID 166
FlexNet Agent 257
FlexNet ID dongle
definition 166
hostid 166
FlexNet ID dongle with FLOAT_OK 234
FlexNet Licensing Service
installing (Linux) 180
installing (OS X) 182
installing (Windows) 179
FlexNet Operations 257
FLOAT_OK 122
floating license 258
format of license file 101
fulfill 258
fulfillment record 258

H

HEARTBEAT_INTERVAL
SERVER line 112
heartbeats 258
automatic 52
automatic vs. manual 52
process 51
hop count 258
HOST_BASED 122
host, SERVER line 110
HOSTID 123
hostid 166, 258
ANY 167
cloud-licensing 169
COMPOSITE 167
CPUID 249
definition 163
DEMO 167
determining with lmhostid 102, 163
different platforms 102
DISPLAY 168
FlexNet ID dongle 166
HOSTID_INTEL 249
HOSTNAME 168
ID 168
Intel Pentium III+ 249
INTERNET 168
platform-specific 163
SERVER line 111
special 167
table of expected 102
TPM 164, 166
vendor-defined 171, 176
virtualization 169
hostid type 258
HOSTID_INTEL hostid 249
HOSTNAME hostid 168
hybrid license 258

I

ID hostid 168
INCREMENT line 114
installanchorservice.exe 182
installing
FlexNet Licensing Service (Linux) 180
FlexNet Licensing Service (OS X) 182
FlexNet Licensing Service (Windows) 179
installs.exe 31
Intel Pentium III+ hostid 249
internationalization 195
INTERNET
hostid 168
internet activation, See programmatic activation
INTERNET hostid
on SERVER line 111
IPv6
support overview 159
ISSUED 124
ISSUER 124

K

keyboard activation, See short-code activation

L

lc_auth_data()
and demo licensing 142
lc_checkout() limits 242

lc_init_simple_composite() 171
lc_set_attr()
    limits 242
lc_vsend()
    and ls_vendor_msg 70
length, signature 147
lenient licensing 143
libraries
    non-TRL 151
    TRL-only 150
license
    borrowing 236
    counted 256
    served 262
    unserved 263
    version 120
license administrator 258
license count
    in license file 120
license example
    expiring uncounted 104
    floating, counted 104
    node-locked, uncounted 103
    PACKAGE 135
    PACKAGE SUITE variations 138
license file 258
    case folding 197
    comment lines 110
    decimal format limits 241
    FEATURE line 114
    format 101
    generating with lmcrypt 185
    INCREMENT line 114
    limits 240
    line continuation 110
    locating 22
    PACKAGE line 133
    SERVER line 110
    signature 103
    specifying for license server 28
    UPGRADE line 132
    USE_SERVER line 114
    UTF-8 encoding 197
    VENDOR line 112
license file directory
    redundancy 40
license file generator
    lmcrypt 185
license generator 258
license group 259
license in a buffer 106
license key
    12-character 247
    20-character 247
    length 246
    length and start-date 247
    long 247
    version behavior 248
license model 259
license models
    expiring uncounted demo 141
    lenient licensing 143
    limited functionality demo 142
license rehosting 233
license rights 259
    local 260
license search path 259
    and equivalent hostids 112
    default 23
    redundancy 40
    specification 24
license server 259
    default ports 111
    improving performance 114
    license search path 40
    license-file directory 40
    lmgrd
        run in foreground 29
    sockets used 242
    specifying license files 28
license server debug log
    starting for lmgrd 28
license server manager 259
license server manager, definition 259
license server system 259
license sharing 260
license usage
    billback 143
    in report log 143
license-file list, See license search path
Licenses
    Concurrent 22
licensing client 260
licensing, time zone 201
limitations, of COAVAIL checkout 216, 227
limits
    decimal license 241
    domain 243
    error message 242
    lc_checkout() requests 242
    lc_set_attr() 242
    license file 240
    LM_LICENSE_FILE 244
    lmgrd 243
    subnet 243
    vendor daemon 242
    vendor-defined hostid 242
    wide-area network 243

LM_A_CHECKOUTFILTER
and VENDOR_STRING 130
LM_A_VD_FEATURE_INFO
and OVERDRAFT 144
LM_BADCODE 102
LM_CO_AVAIL_NOWAIT flag
description 209, 219
error codes 216, 226
example 213, 223
limitations 216, 227
LM_CO_NOWAIT flag
description 209
example 212
lm_code.h
LM_STRENGTH 33
TRL_KEYs 33, 151
VENDOR_KEYs 150
VENDOR_NAME 33
LM_LICENSE_FILE 40
limits 244
LM_SEEDS 147
LM_STRENGTH 33, 147, 150
lmadmin 15, 260
lmcrypt 185
lmdown
disabling 29
restricting access 29
lmflex.c 19
lmgrd 28, 260
command-line syntax 28
limits 243
starting 30
starting debug log 28
starting on Windows 31
upgrading 31
lmhostid 102, 163
lmremove
disabling 29
restricting access 29
lmreread
restricting access 29
lmtools.exe 188
lmutil
lmborrow 188
lmdiag 188
lmdown 188
lmhostid 189
lminstall 189
lmlicvalidator 189
lmnewlog 189
lmobfslog 189
lmpath 189
lmremove 189
lmreread 189
lmstat 189
lmswitch 189
lmswitchr 189
lmver 189
lmutil.exe 188
local activation 260
local license rights 260
locating license file 22
-c license_file_list argument 23
LM_LICENSE_FILE 23
VENDOR_LICENSE_FILE 23
locating the license server
@host 24
license directory 24
license file 24
port@host 24
log file
debug 256
report 261
long license key 247
ls_a_behavior_ver 245
ls_a_check_baddate 62
ls_a_crypt_case_sensitive 245
ls_a_license_case_sensitive 73
ls_a_lkey_long 245
ls_a_lkey_start_date 245
ls_allow_vm
and virtualization hostid 169
LS_ATTR_CHECKOUT_DATA 66
ls_attr.h 65, 66
ls_ba_heartbeat_interval 246
ls_ba_heartbeat_retry_count 246
ls_ba_heartbeat_retry_interval 246
ls_checkout() and ls_outfilter 66, 69
ls_client_removal_on_reread 77
ls_compare_vendor_on_increment 62
ls_compare_vendor_on_upgrade 62
ls_daemon_periodic 63
ls_do_checkroot 78
ls_dump_send_data 79
ls_enforce_startdate 245
ls_flexid9_hasp4_support 63
ls_get_attr(), and ls_outfilter 65, 66
ls_get_status_is_master 63
ls_hud_hostid_case_sensitive 79
ls_incallback 64
ls_infilter 64
ls_min_lmremove 64
ls_minimum_user_timeout 64
ls_outfilter 65
ls_secure_comms 67
ls_send_oldest_signature 246
ls_show_vendor_def 68
ls_tell_startdate 246

ls_use_all_feature_lines 82
ls_use_featset 246
ls_user_init1 68
ls_user_init2 68
ls_user_init3 68
ls_user_lockfile 83
ls_user_lockfile2 83
ls_vendor_msg 70
lsvendor.c 32, 59

M

machineid 260
managing debug log file 33
manual activation 260
Manual Response Generator 260
MAX_OVERDRAFT 143
migration
    to TRL, full 149
    to TRL,partial 151
MINIMUM 124
mobile licensing
    borrowing 236
    FlexNet ID dongle with FLOAT_OK 234
    node-locked to FlexNet ID dongle 234
    node-locked to laptop 234
    node-locked to user name 236
    prepaid license pool fulfillment 236
model, license 259

N

newlink GlossaryC 255
NOTICE 124

O

ONE_TS_OK 124
options file 187, 260
    path on VENDOR line 113
OPTIONS=SUITE 134
OPTIONS=SUITE_RESERVED 134
OVERDRAFT 125
OVERDRAFT detection 144

P

package 261
PACKAGE line 101, 133
    and demo licensing 142
    COMPONENTS 133
    OPTIONS=SUITE 134
    OPTIONS=SUITE_RESERVED 134
package suite
    bundle 138
    sharing components 138, 139
    unlimited component usage 138
part number 261
permanent license 261
PLATFORMS 125
port
    SERVER line 111
    VENDOR line 113
port@host 24
ports
    license server 111
PRIMARY_IS_MASTER
    SERVER line 112
product 261
product version 119
programmatic activation 261
publisher id 261

Q

quorum 261

R

redundant license server
    license search path 40
    license-file directory 40
regular entitlement, See simple entitlement
rehost 261
rehosting, license 233
repair request 261
repair response 261
report log file 143, 261
responsegen 262
restricting access
    lmdown 29
    lmremove 29
    lmreread 29
return request 262
return response 262
return transaction 262

S

served license rights 262
server
    license 259
SERVER line 101
    HEARTBEAT_INTERVAL 112
    host 110
    hostid 111

Internet hostid 111
port 111
PRIMARY_IS_MASTER 112
syntax 110
short code 262
short-code activation 262
SIGN, keyword 147
SIGN= 126
SIGN2, keyword 147
signature 103, 126, 262
choosing strength 149
configuration elements 146
configurations 146
ENCRYPTION_SEEDS 146
keyword 147
length 147
LM_SEEDS 147
LM_STRENGTH 147
non-TRL 145
supported clients 147
TRL 145, 148
TRL_KEYS 147
types 145
simple entitlement 262
single-action transaction 262
SN 126
sockets
number used by license server 242
sort 132
specifying license in application 106
specifying license search path 24
START 126
START_LICENSE 24, 106
starting lmgrd 28, 30
Windows 31
subnet limits 243
suite 262
SUITE_DUP_GROUP 126, 138
SUPERSEDE 127
supersede 262
SUPERSEDE_SIGN 128

T

Terminal Server support 128
three-server redundancy 262
time zone licensing 201
TIMEOUT
setting 64
TIMEOUT option 188
TPM hostid 166
triad 262
trial license 263
TRL 148
application attributes 151, 152
backward compatibility 151
enabling in your application 150
full migration to 149
LM_STRENGTH_113BIT 149
LM_STRENGTH_163BIT 149
LM_STRENGTH_239BIT 149
migrating to 149
non-TRL libraries 151
partial migration to 151
signature type 145
TRL-only libraries 150
TRL_KEYS 147
TRL_KEYs 33, 151
trusted configuration file 263
trusted storage 263
trusted storage configuration 263
TS_OK 128
type, hostid 258

U

uncounted license
uncounted 263
uninstallanchorservice.exe 183
unserved license rights 263
unserved licenses 263
upgrade 263
UPGRADE line, syntax 132
upgrading
lmgrd 31
vendor daemon 32
upsell 263
USE_SERVER line 101, 114
USER_BASED 130
user_info 132
utilities on UNIX 188
utilities on Windows 188

V

Vendor Certificate Generator 263
vendor daemon 263
building on UNIX 32
building on Windows 32
configuring 31
customizing 32
limits 242
path on VENDOR line 113
upgrading 32
vendor daemon, common 85
VENDOR line 101
eport 113
options file path 113

port 113
syntax 112
vendor daemon path 113
vendor name 113
vendor name 119, 263
on VENDOR line 113
vendor_info 132
VENDOR_KEYs 150
VENDOR_LICENSE_FILE 40
VENDOR_NAME 33
VENDOR_STRING 130
and ls_compare_vendor_on_* 62
vendor-defined checkin callback 64
vendor-defined checkin filter 64
vendor-defined checkout filter 65
vendor-defined hostid
defining 171, 176
limits 242
version 119
virtualization 229
hostid 169
ls_allow_vm variable 169

W

wide-area network limits 243
windback 263
windback detection 264
windback tolerance 264