Reference Manual

VERSION 9.5

AUGUST 2004

macrovision™

Printed in the USA.
August 2004

Table of

Contents

Preface

About This Manual

This manual, the *FLEXlm Reference Manual*, provides a comprehensive reference to the advanced features of FLEX*lm*® from the software developer's perspective, including a complete description of three application programming interfaces:

- Trivial API, used with single-process applications that checkout just one feature. It is provided in two formats: macro-based and function-based.
- Simple API, used with applications that require checking out more than one feature or license at a time.
- FLEXible API, the most complete API available for license management.

All documentation is provided online in the `htmlman` directory and can be accessed through any HTML browser.

Product Information

FLEX*lm* is a software licensing package that allows licensing a software application on a concurrent-usage as well as on a per-computer basis. FLEX*lm* allows the implementation of a wide variety of *license policies* by the developer of an application.

With FLEX*lm*, you, the application developer, can restrict the use of your software packages to a:

- Single specified computer
- Specified number of users on a network of one or more computer systems

FLEX*lm* is available on UNIX and Windows. FLEX*lm* features include:

- Operation in a heterogeneous network of supported computer systems
- Transparent reconnection of applications when their license server process becomes unavailable, including conditions of license server node failure

- Simple configuration by using a single license file per network
- Configuration controls for system administrators
- Administration tools for system administrators
- Independent features from one or multiple vendors with independent vendor security codes
- A wide variety of license policies and license styles, including:
  - Floating licenses
  - Node-locked licenses
  - Named-user licenses
  - Demo licenses
  - Counted and uncounted licenses
  - Optional license expiration dates
  - Several vendor-definable fields for each application feature.

Other Product Documentation

The *FLEXlm Programmers Guide* provides an introduction to FLEX*lm*, instructions for evaluating FLEX*lm* on UNIX and Windows systems and guidelines for integrating FLEX*lm* into your application.

The *FLEXlm Java Programmers Guide* contains guidelines for using FLEX*lm* Java and the FLEX*lm* Java API reference.

The *FLEXlm End Users Guide* contains information relevant to users of products that utilize FLEX*lm* as their license management system, including descriptions of the license administration tools which are bundled with FLEX*lm*. It describes setup and administration of a FLEX*lm* licensing system.

Typographic Conventions

The following typographic conventions are used in this manual:

- The first time a new term is used it is presented in *italics*.
- Commands and path, file, and environment variable names are presented in a `fixed_font`.
- Other variable names are in an *`italic_fixed_font`*.
- API function calls are in a sans-serif font.

Contacting Technical Support

Technical Support is available to customers with current support contracts and prospects. Please include the following information with your inquiry:

- Product Name
- Product Version
- Operating System Name and Version

Contact the support center for your area:

http://www.macrovision.com/services/support/

Chapter 1

Introduction

1.1 FLEX*lm* APIs

Most of the important functionality and flexibility in FLEX*lm* is contained in the license file; all license file attributes are available to all APIs.

The application program interfaces to FLEX*lm* via a set of functions that request (checkout) and release (checkin) licenses of selected feature(s).

There are four major FLEX*lm* APIs available to the developer:

- Trivial API - including both function-based and macro-based versions
- Simple API
- FLEXible API
- Java API (based on the FLEXible API)

Macrovision recommends using the Trivial API; if, however, the application requires FLEX*lm* functionality not provided in the Trivial API, use the Simple API. For complete flexibility, use the FLEXible API. The Simple, Trivial, and FLEXible APIs are documented in this manual; the Java API is documented in the *FLEXlm Java Programmers Guide*.

1.1.1 Trivial and Simple APIs

In the Trivial and Simple APIs, a licensing “policy” is selected as an argument to the license request call.

The Simple API must be used instead of the Trivial API when:

- A single process needs to separately license sub-functionality—that is, when two or more feature names may be checked out.
- The checkout call needs to be able to check out more than one license of a feature.

1.1.2 FLEXible API

Most commonly, the FLEXible API is required for:

- Asynchronous queuing, especially in GUI-based applications where queueing is required.
- To obtain a list of users of a given feature.

1.1.3 Migrating to the FLEXible API

If you need the functionality of the FLEXible API, but have been using either the Simple or Trivial API, it is not difficult to migrate. The following changes will have to be made:

1. Remove the include directive for `lmpolicy.h` and add:

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

2. Declare the following two variables:

   ```
   LM_HANDLE *job;
   VENDORCODE code;
   ```

3. Before the checkout call add:

   ```
   lc_new_job(0, 0, &code, &job);
   lc_set_attr(job, LM_A_LICENSE_DEFAULT,
         (LM_A_VAL_TYPE)licpath);
   ```

   where *`licpath`* is the same value as the last argument to CHECKOUT() or lp_checkout().

4. Change the CHECKOUT() or lp_checkout() call to:

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

SEE ALSO

- Example code in `machind/lmflex.c`

1.2 FLEX*lm* Terms and Definitions

The following terms are used as defined to describe FLEX*lm* concepts and software components:

Client application	An application program requesting or receiving a license.
Counted license	A license with a non-zero license count. A license server is required to manage a counted license. Counted licenses usually float on a network.
Daemon	A process that "serves" clients. Sometimes referred to as a *server* or *service* process.
Debug log file	One or more ASCII text files written by the license server daemons, `lmgrd` and the vendor daemons it manages. A debug log file contains status and error messages useful for debugging the license server.
Feature	Any functionality that needs to be licensed. The meaning of a feature will depend entirely on how it is used by an application developer. For example, a feature could represent any of the following: • An application software system consisting of hundreds of programs • A single program (regardless of version) • A specific version of a program • A part of a program • A piece of data (restricted via the access functions)
Feature line	A line in a license file that licenses a particular feature. A feature line begins with one of the keywords FEATURE, UPGRADE, PACKAGE, or INCREMENT.

Floating license	A license that can authorize usage of an application by one of a group of users on a network. A license server is required to manage a floating license. Floating licenses must be counted.
Heartbeats	Periodic messages sent from a client application to a license server to solicit replies that ensure that the license server is still running. Heartbeats can be implemented automatically or called explicitly from a client application. If called explicitly, they are referred to as manual heartbeats.
License	The legal right to use a feature. FLEX*lm* can restrict licenses for features by counting the number of licenses already in use for a feature when new requests are made by the client application software. FLEX*lm* can also restrict software usage to particular nodes or user names.
License file	A text file specific to an end-user site that contains descriptions of 1) license server node(s), 2) vendor daemons, and 3) licenses (features) for all supported products.
License-file list	A list of license files separated with a colon ":" on UNIX and a semi-colon ";" on Windows. A license-file list can be accepted in most places where a license file is appropriate. When a directory is specified, all files matching `*.lic` in that directory are automatically used, as if specified as a list.
License key	Optional 12- to 20-character hexadecimal license signature used to authenticate a license.
License server	An `lmgrd` and one or more vendor daemon processes. License server refers to the processes, not the computer on which they run.

License server machine	A computer system on which license server processes run. The license server machine will host all site-specific information regarding all floating feature usage. Multiple license server machines used for three-server redundancy can logically be considered the license server machine. Also known as license server node.
`lmgrd`	The background process, or license manager daemon, that sends client processes to the correct vendor daemon on the correct machine. The same license manager daemon process can be used by any application from any vendor because this daemon neither authenticates nor dispenses licenses. `lmgrd` processes no user requests on its own, but forwards these requests to a vendor daemon.
Node-locked license	A license that can authorize use of an application running on a single specific machine, as opposed to on a network. Node-locked licenses are usually, but not necessarily, uncounted. Node-locked, uncounted licenses do not require a license server.
Options file	A text file implemented for a particular vendor's software by a FLEX*lm* license administrator at an end-user site to customize the behavior of the vendor daemon or to enable or restrict the use of the licenses managed by the vendor daemon at that site.
Report log file	A text file written by a single vendor daemon. Report logs are not human readable; the data is compressed and authenticated for use with SAM*report* and FLEX*bill*, two Macrovision products that produce reports on license usage.
Signature	A secure 12- to 120-character hexadecimal number which "authenticates" the readable feature line in the license file, ensuring that the text in feature line has not been modified.

Uncounted license	A license that does not restrict the number of uses of an application. Uncounted licenses must be node-locked to a machine hostid—node-locked, uncounted licenses do not require a license server.
Vendor daemon	The server process that dispenses licenses for the requested features. This binary is built by an application's vendor (from libraries supplied by Macrovision) and contains the vendor's unique encryption seeds.
Vendor name	The name of the vendor as found in `lm_code.h`. Used as the name of the vendor daemon.

Chapter 2

The License File: Overview

This chapter provides an overview of the license file format and examples of license file types. Once you have an idea of the license file format for your FLEX*lm*-enabled product, proceed to Chapter 3, “The License File: Syntax,” for specific syntactical information.

2.1 Format of the License File

A license file consists of the following sections:

- Server Information

  This section appears in the license file if a license server is used (that is, if any features are *counted*). The following types of lines can appear in this section:

  - SERVER lines

    The SERVER line(s) contain information about the node(s) where `lmgrd` is running.

  - VENDOR line

    The vendor-specific VENDOR line contains information about the vendor daemon that runs on the license server node(s).

  - USE_SERVER line

    A USE_SERVER line, if used, usually follows the SERVER line and indicates that a client application should not process the rest of the license file itself, but should check out the license directly from the license server.

- Package Information

  This section defines how individual components are grouped into packages. There is only one type of line in this section:

  - PACKAGE line

- License Rights Information

  This section includes information which 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 lines are found in this section.

  - FEATURE line

    Defines the individual licensed feature.

  - INCREMENT line

    Adds additional entitlement to an existing feature or increment line for the same feature.

  - UPGRADE line

    Upgrades entitlement for an existing feature.

Note: See the *FLEXlm Programmers Guide* for information on `lmcrypt` and `makekey`, the license generation utilities. Also see Section 6.4.4, "lc_cryptstr()," for generating licenses with a C function call.

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

The SERVER hostids and everything on a FEATURE line, except the vendor daemon name and lowercase *`keyword=value`* pairs, are input to the authentication algorithm to generate the signature for that FEATURE line. If the authenticated portions are edited, an LM_BADCODE error will result when the FLEX*lm*-enabled product tries to checkout a license for that feature.

In summary, the only data items in the license file that are editable by the end user are:

- Host names on SERVER lines
- 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 lines

Any amount of white space can separate the components of license file lines; data can be entered via any plain text editor. Vendors can therefore distribute license data via fax or telephone.

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

2.2 Hostids for FLEX*lm*-Supported Machines

A hostid is a means used to uniquely identify a specific machine. There are two different contexts in which a hostid is used:

- A FLEX*lm* license

  The license is bound to one or more hostids. They are used to define which end-user machines are licensed to run your product.

- A FLEX*lm* license server

  A hostid used to define which machine is authorized to run a license server that serves licenses to your FLEX*lm*-enabled product.

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 FLEX*lm* utility, `lmhostid`, prints the default hostid that FLEX*lm* expects to use on any given platform. FLEX*lm* supports platform-specific as well as vendor-defined hostids. In addition, there are a number of special hostid types which apply to all platforms. These special hostid types can be used on either a SERVER line or a FEATURE line, wherever a hostid is required. The appendix entitled “Hostids for FLEX*lm*-Supported Machines” in the *FLEXlm End Users Guide* contains a listing of platform-specific and special hostids. Vendor-defined hostids are discussed in Chapter 15, “Vendor-Defined Hostid Types.”

The decision to use a particular hostid is made in the license file rather than the FLEX*lm*-enabled product. However, there are occasions when the FLEX*lm*-enabled product may want to determine a hostid type specified in a particular FEATURE line, for example, to detect the usage of special hostid types such as DEMO and ANY. The FLEXible API function, lc_auth_data(), called within the product, provides this information.

SEE ALSO

- Section 6.4.1, “lc_auth_data()”
- Section 8.1.1, “l_new_hostid()”

- Section 8.1.6, “lc_free_hostid()”
- Section 3.2, “SERVER Lines”
- Section 3.5, “FEATURE /INCREMENT Lines”
- Section D.4.3, “Intel Pentium III Hostid (HOSTID_INTEL)”

2.3 Types of License Files

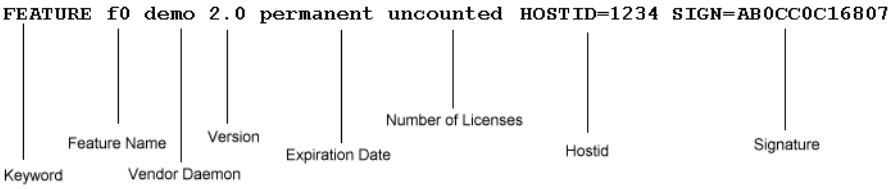
The information in the license file affects how the contents are interpreted by FLEX*lm*. The license file supports

- Node-locked licenses
- Demo/evaluation licenses
- Network licenses—either one license server or three server redundant configurations

Examples of these types are presented below. In addition, see the `examples/licenses` for additional examples.

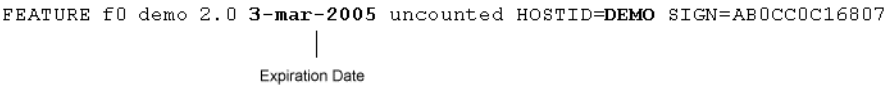
Following are license file examples, starting with the simplest. In the examples, the changes from the previous example are in **bold** text.

2.3.1 Simple Uncounted License



- Uncounted licenses have unlimited use on the hostid specified. Uncounted licenses 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 through 2.0 (inclusive).

2.3.2 Expiring Demo License



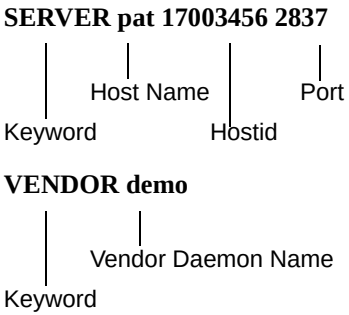
- This license expires on 3 March, 2005.
- Specify the year with four digits, e.g. “1-jan-2001.”
- The “DEMO” hostid indicates that this license allows “f0” to run on any system. In addition, the client application can also detect that it is in demo mode, and could behave differently.

2.3.3 Simple Floating (Counted) License

```
SERVER pat 17003456 2837
VENDOR demo
FEATURE f1 demo 2.0 permanent 9 SIGN=DBCC10416777
```

- SERVER and VENDOR lines required.
- Unexpiring.
- Floating — runs on any node. No hostid on FEATURE line.
- Limited to nine concurrent licenses.
- Server restricted to hostid “`17003456.`” To remove this restriction, use hostid of “ANY” (e.g., SERVER pat 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 here:



- Host name can be changed by the end user. If host name is `this_host`, clients running on the same node as the server work fine. 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.

- Since a *vendor_daemon_path* is not specified in the VENDOR line, lmgrd uses the current directory or 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.

2.3.4 Floating with Three Server Redundancy

```
SERVER pat 17003456 2837
SERVER lee 17004355 2837
SERVER terry 17007ea8 2837
VENDOR demo
FEATURE f1 demo 1.0 1-jan-2005 10 SIGN=1AEEFC8F9003
FEATURE f2 demo 1.0 1-jan-2005 10 SIGN=0A7E8C4F561F
```

- A three server redundant configuration is specified. This is a a set of three server nodes all running the same operating system, any two of which must be running for FLEX*lm* to function.
- Three SERVER lines and one VENDOR line required.
- All three servers communication via the same TCP/IP port.
- Two features are licensed: f1 and f2.
- Expires on January 1, 2005.
- Floating — runs on any node. No hostid on FEATURE lines.
- Limited to ten concurrent licenses for each feature.

2.3.5 Mixed Floating (Counted) and Uncounted

```
SERVER speedy 08002b32b161
VENDOR demo
FEATURE f1 demo 2.0 permanent 1 HOSTID=80029a3d SIGN=7B9F02AC0645
INCREMENT f1 demo 2.0 permanent 2 HOSTID=778da45 SIGN=6BAFD2BC1C3D
FEATURE f0 demo 2.0 permanent uncounted HOSTID=FLEXID=8-12345678 \
        SIGN=AB0CC0C16807
```

- Checkouts of “f0,” since it is *uncounted*, may not communicate with the server — they only verify that the client is on node “FLEXID=8-12345678 (i.e., the node has the hardware key with id 8-12345678 attached)” and that the version is <= 2.0. If USE_SERVER is specified, or either *VENDOR*_LICENSE_FILE or LM_LICENSE_FILE is set to @*host* (or *port*@*host* if a port number is specified on the SERVER line), then checkouts do require a server and their usage is logged.
- The “f0” line does not require the SERVER or VENDOR lines, and in fact could reside in another license file altogether.

2.4 License in a Buffer

The license file does not need to be located on disk—it can be specified in the program itself. The license path in CHECKOUT(), or lp_checkout() can specify the actual license, as in this example:

```
CHECKOUT(LM_RESTRICTIVE, "f1", "1.0",
        "START_LICENSE\n\
         FEATURE f1 demo 1.0 permanent \
         uncounted HOSTID=ANY \
         VENDOR_STRING=\"Acme Inc\" SIGN=50A35101C0F3\n\
         END_LICENSE");
```

The license must begin with START_LICENSE\n and end with \nEND_LICENSE, where the embedded newlines are required.

This can also be a license-file list; as in the following example:

```
CHECKOUT(LM_RESTRICTIVE, "f1", "1.0",
        "path_to_license_file:START_LICENSE\n\
         FEATURE f1 demo 1.0 permanent \
         uncounted HOSTID=ANY \
         VENDOR_STRING=\"Acme Inc\" SIGN=50A35101C0F3\n\
         END_LICENSE"
```

In this example, *path_to_license_file* is first in the list, followed by the license in the string.

Specifying a license in a buffer is particularly useful when selling libraries if a separate license file is not desirable, or as a final “fail-safe” license in the event that the license server is not running.

Chapter 3

The License File: Syntax

This chapter is a reference for license file syntax. It is divided up into sections which correspond to the different license file sections as outlined in Section 2.1, “Format of the License File.” For an overview of license files, see Chapter 2, “The License File: Overview.”

Table 3-1 lists the license file sections described in this chapter.

Table 3-1:License File Sections

Section	Synopsis
General Syntax Issues	Syntax rules common to all sections.
Server Information	
SERVER Lines	Contains information about the machines running the license server, `lmgrd`.
VENDOR Line	Contains information about the vendor daemon that runs on the license server machines.
USE_SERVER Line	Forces the FLEX*lm*-enabled product to access servers in preceding SERVER lines for licenses.
Package Information	
PACKAGE Lines	Provides a way to license a set of components into one package.
License Rights Information	

Table 3-1:License File Sections (Continued)

Section	Synopsis
FEATURE /INCREMENT Lines	Describes the licensed features available for the specified vendor's products.
UPGRADE Lines	Creates a new version of the license, thereby replacing the older one.

3.1 General Syntax Issues

3.1.1 Comment Lines

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

3.1.2 Line Continuation

Long lines are broken up by line the continuation character, "\." The following line is an example of using the line continuation character.

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

3.2 SERVER Lines

SERVER *host hostid* [*port*]

A SERVER line specifies the name and hostid of the license server machine and, optionally, the TCP/IP port number through which to communicate to `lmgrd`. A license file may have one or three SERVER lines. The SERVER node name in the license file can be any network alias for the node.

host	String returned by the UNIX `hostname` or `uname -n` commands, or an IP address in ###.###.###.### format. This can be edited by the license administrator. 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 allows the product to be installed and to start the license server. Clients on the same host as the license server will work fine. Clients on other nodes will 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. Note that `lminstall` and lc_convert() will automatically change `this_host` to the real host name when appropriate.

hostid	String returned by the `lmhostid` command (case insensitive). Only one hostid is allowed. Alternate special hostids can also be specified here, including ANY, HOSTNAME=*host*, etc. See the *FLEXlm End Users Guide* for information about expected and special hostids, and Chapter 15, "Vendor-Defined Hostid Types," for vendor-defined hostids. **WARNING**: If the INTERNET hostid is used on the SERVER line, wildcards should not be allowed in the IP address. If wildcards are used, the customer could easily start license managers on more than one node and obtain "extra" licenses.
port	TCP port number to use. This can be edited by the license administrator. If not specified, FLEX*lm* will automatically use the next available port number in the range 27000-27009. Applications, when connecting to a server, try all numbers in the range 27000-27009. The port number is required if the license is for a three-server redundant license server. Using a port number in the range 27000-27009 is recommended when specifying a port number. SERVER lines specifying servers in a three-server redundant license server system configuration require a port number to be specified; Macrovision recommends using port numbers outside the range of 27000 through 27009.

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-file list can be used if hostids are not identical, but refer to the same machine.

SEE ALSO

- *FLEXlm End Users Guide*
- Section 3.5.10, “HOSTID”
- Chapter 15, “Vendor-Defined Hostid Types”

3.3 VENDOR Line

```
VENDOR vendor [vendor_daemon_path] \
       [[options=]options_file_path] [[port=]port]
```

The VENDOR line specifies the name and location of a vendor daemon, as well as the location of the end user’s options file.

vendor	Name of the vendor daemon used to serve at least some feature(s) in the file.
vendor_daemon_path	Path to the executable for this daemon. If blank, `lmgrd`’s `PATH` environment variable, plus the current directory, is used by `lmgrd` to find the daemon process to start.
options_file_path	Path to the end-user options file for this 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 end-user options file.
port	Vendor daemon port number. The default, if *port* is not specified, is chosen by the operating system at run-time. Sites with Internet firewalls need to specify the port number the daemon uses. If a port number is specified on the VENDOR line, there may be a delay restarting the vendor daemon until all the clients have closed their connections to the vendor daemon.

Note: A port number must be specified in the VENDOR line when you are connecting to the vendor daemon through a firewall.

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
```

3.4 USE_SERVER Line

```
USE_SERVER
```

USE_SERVER takes no arguments and has no impact on the server. When the client 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 one the server uses
- Requires only SERVER and USE_SERVER lines

3.5 FEATURE /INCREMENT Lines

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

A feature line describes the license to use with a FLEX*lm*-enabled product. It is composed of five parts:

- `FEATURE` or `INCREMENT` - required
- 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 3-2 summarizes the FEATURE/INCREMENT fields and keywords.

Table 3-2:FEATURE/INCREMENT Fields and Keywords

Keyword
Positional Fields — in required order
Feature Name
Vendor Daemon Name
Feature Version
Expiration Date
Number of Licenses
Authenticated Vendor Keywords
BORROW
DUP_GROUP
FLOAT_OK
HOST_BASED
HOSTID
ISSUED
MINIMUM
OVERDRAFT
PLATFORMS
START
SUITE_DUP_GROUP
SUPERSEDE

Table 3-2:FEATURE/INCREMENT Fields and Keywords (Continued)

Keyword
TS_OK
USER_BASED
VENDOR_STRING
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, set `ls_use_all_feature_lines` set in `lsvendor.c` for your vendor daemon The original behavior of FEATURE line is then unavailable to that application. Notify your end user if you set `ls_use_all_feature_lines`.

There are two formats for FEATURE; pre-v3.0 and current. The older format is still understood and correct with new clients and servers, but the current format is more flexible.

LICENSE POOLS

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

- feature name
- version
- DUP_GROUP
- FLOAT_OK
- HOST_BASED
- HOSTID
- PLATFORM
- USER_BASED
- VENDOR_STRING (if configured by the vendor as a pooling component)

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 up 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.

Now, in contrast, this next 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 are different.
- Since a license pool is not formed, this independent tracking *only* works with INCREMENT, not FEATURE, because the server only recognizes the first FEATURE line for a given feature name. Subsequent ones are ignored.

A single checkout request can not 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 3 licenses for v1.0 and one with 4 licenses for v2.0. A request for 5 licenses for version v1.0 is denied because neither pool has 5 licenses.

FEATURE/INCREMENT Example

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

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

They result in four licenses for v1.0 *or* five licenses for v2.0, depending on their order in the file. Now, consider:

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

- These result in four licenses for v1.0 *and* five licenses for v2.0 and below being available, giving a total of nine licenses for “`f1`.”
- INCREMENT 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-2005 uncounted HOSTID=DEMO \
        SIGN=123456789012
```

This feature has unlimited usage on any hostid, requires no license server and is, therefore, could 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-2005.

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

- Chapter 15, “Vendor-Defined Hostid Types”
- Section 8.3.9, “ls_use_all_feature_lines”
- Section 13.2.4, “ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade”
- Section 8.2.5, “LM_A_CRYPT_CASE_SENSITIVE”

3.5.1 Feature Name

`feature` is the name given to the feature by the vendor. Legal feature names in FLEX*lm* must contain only letters, numbers, and underscore characters. Letters in the feature name are case insensitive by default. If case sensitivity is desired, see Section 8.2.10, “LM_A_LICENSE_CASE_SENSITIVE.”

3.5.2 Vendor Daemon Name

`vendor` is the vendor daemon name from a VENDOR line. This vendor daemon serves this *`feature`*.

3.5.3 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.

3.5.4 Expiration Date

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

```
{dd-mmm-yyyy | permanent}
```

For example, `22-mar-2005`. 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`.

3.5.5 Number of Licenses

Number of licenses for this feature; a value greater than 0 denotes a *counted* license. Use "uncounted" or 0, for unlimited use of node-locked licenses.

3.5.6 BORROW

```
BORROW[=n]
```

Optional field. Enables license borrowing for a particular FEATURE/INCREMENT line (see the *FLEXlm Programmers Guide* for more information about license borrowing). *n* is the maximum number of hours that the license can be borrowed for. The default maximum borrow period is 168 hours, or one week. BORROW licenses are susceptible to extra uses, should a user stop and restart the licenses server while licenses are borrowed. A lower value of *n* minimizes the affects of the possible use of extra licenses. A software vendor may choose to issue BORROW licenses only to their "trusted" customers. (The maximum borrow period is limited by the maximum value of a 32-bit integer: 2 billion hours.)

3.5.7 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, and the DUP_MASK is the OR of the combination. 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.

3.5.8 FLOAT_OK

`FLOAT_OK[=server_hostid]`

Optional field. Enables mobile licensing via FLEX*id* with FLOAT_OK for a particular FEATURE/INCREMENT line (see the *FLEXlm Programmers Guide* for more information about mobile licensing). This FEATURE/INCREMENT line must also be node-locked to a FLEX*id*.

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 only be run on the machine with the hostid that `lmhostid` returns equal to the *server_hostid* specified with FLOAT_OK.
- A user can run on the license server machine, 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.
- The hostid on the FLOAT_OK FEATURE line must be only a single hostid. For multiple dongles, use individual FEATURE lines for each dongle.

3.5.9 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 end-user options file. The purpose is to limit the use to a particular number of hosts, but allow the end user to determine which hosts. If *=n* is specified, then the number of hosts which 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.

3.5.10 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 FLEX*lm* utility, `lmhostid`, 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 machine 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, e.g.:

```
HOSTID="12345678 FLEXID=6-876d321a HOSTNAME=joe"
```

If a list of hostids is used, the feature is granted on any one of the hostids in the list.

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 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

- *FLEXlm End Users Guide* for information about platform-specific and special hostids.
- Section 3.2, “SERVER Lines”

3.5.11 ISSUED

```
ISSUED=dd-mmm-yyyy
```

Optional field. Date that the license was issued. Can be used in conjunction with SUPERSEDE.

3.5.12 ISSUER

```
ISSUER="..."
```

Optional field. Issuer of the license.

3.5.13 MINIMUM

```
MINIMUM=n
```

Optional field. If in lc_checkout(...*num_lic*...), *num_lic* is less than *n*, then the server will checkout *n* licenses.

3.5.14 NOTICE

```
NOTICE="..."
```

Optional field. A field for intellectual property notices.

3.5.15 OVERDRAFT

```
OVERDRAFT=n
```

Optional field. The OVERDRAFT policy allows you to specify a number of additional licenses which your end user will be allowed to use, in addition to the licenses they have purchased. This is useful if you want to allow your customers to not be denied service when in a “temporary overdraft” state. Usage above the licensed limit will be reported by the SAM*report* 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 at least three members of interest: `lic_in_use`, `lic_avail`, and `overdraft`. If `lic_in_use` > `lic_avail` – `overdraft`, then you are in an “overdraft state.”

3.5.16 PLATFORMS

```
PLATFORMS="plat1 ... platn"
```

Optional field. This allows you to restrict usage to particular hardware platforms. The platforms are defined with FLEX*lm* platform names and are the same as used to license FLEX*lm* itself: `sun4_u5`, `i86_n3`, etc. The FLEX*lm Release Notes* contain the currently supported platforms and their associated FLEX*lm* platform names.

Note that the platform name can be overridden with:

```
lc_set_attr(job, LM_A_PLATFORM_OVERRIDE, (LM_A_VAL_TYPE)str);
```

Note that the trailing digit in the FLEX*lm* 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 “f2”s can be used on Windows systems.

If the checkout fails because it’s on the wrong platform, the error returned is LM_PLATNOTLIC: “This platform not authorized by license.”

SEE ALSO

- Section 8.2.14, “LM_A_PLATFORM_OVERRIDE”
- FLEX*lm Release Notes* located in the `machind` directory of the FLEX*lm* SDK.

3.5.17 SIGN

```
SIGN=signature
```

Required field. Signature for this FEATURE line. *signature* is from 12-20 characters long and is produced by lc_cryptstr() in `lmcrypt` or `makekey`, or by a vendor-defined utility that calls lc_crypstr(). When using `lmcrypt`, put `SIGN=0` at the end of each FEATURE line, and `lmcrypt` will replace the `0` with the correct signature.

3.5.18 SN

```
SN=serial_num
```

Optional field. Useful for differentiating otherwise identical INCREMENT lines. Its only use by FLEX*lm* is to be encrypted in the signature. Similar to HOSTID.

3.5.19 START

START=*dd-mmm-yyyy*

Optional field. Feature start date. If the license is used before this date, the checkout fails with LM_TOOEARLY.

3.5.20 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

- Section 3.5.7, “DUP_GROUP”
- Section 3.7, “PACKAGE Lines”

3.5.21 SUPERSEDE

SUPERSEDE[="*feat1 ... featn*"]

Optional field. Replaces existing lines in a license file. Without the optional list of features, allows vendors to sum up a set of INCREMENT lines in a single, new FEATURE (or INCREMENT) line, which supersedes all INCREMENT lines for the same feature name with previous START or ISSUED dates. 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 (e.g., ISSUED=1-jan-2005). If the ISSUED date is set, then SUPERSEDE uses it, otherwise it uses the START= date.

For example

```
INCREMENT f1 ... 1 ... ISSUED=1-jan-2005
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-2007
```

The second line supersedes the first, and causes FLEX*lm* to ignore the first line.

```
FEATURE f1 ... 1 ... ISSUED=1-jan-2003
FEATURE f2 ... 1 ... ISSUED=1-jan-2003
FEATURE f3 ... 4 ... SUPERSEDE="f1 f2" ISSUED=2-jan-2003
```

“f3” supersedes “f1” and “f2” and causes FLEX*lm* to support only “f3.”

Multiple INCREMENT lines for the same feature both 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:

```
INCREMENT f1 ... 1 ... ISSUED=1-jan-2005
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-2007
INCREMENT f1 ... 3 ... SUPERSEDE ISSUED=1-jan-2007
```

will result in a license pool containing 7 licenses for “f1”, collectively superseding the first INCREMENT line.

3.5.22 TS_OK

```
TS_OK
```

Optional field. FLEX*lm* detects when a node-locked uncounted license is running under Windows Terminal Server. If you want to allow users to run on Terminal Server client machines, the TS_OK keyword must be added to the feature line. Without the TS_OK keyword, a user running on a Terminal Server client machine will be denied a license. See Section 18.4, “Windows Terminal Server Support,” for more information.

3.5.23 USER_BASED

```
USER_BASED[=n]
```

Optional field. If USER_BASED appears, then licenses can only be used by users INCLUDEd for this feature in the end-user options file. The purpose is to limit the use to a particular number of users, but allow the end user to determine which users. If *=n* is specified, then the number of users which can be INCLUDEd 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.

3.5.24 VENDOR_STRING

```
VENDOR_STRING="..."
```

Optional field. Vendor-defined license data. If a checkout is conditional on the contents of the vendor string, then LM_A_CHECKOUTFILTER is the best 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

- Section 6.4.1, “lc_auth_data()”
- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 13.2.4, “ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade”

3.5.25 asset_info

```
asset_info="..."
```

Optional field. Additional information provided by the software end user’s license administrator for asset management. Not encrypted into the feature’s signature.

3.5.26 dist_info

```
dist_info="..."
```

Optional field. Additional information provided by the software distributor. Not encrypted into the feature’s signature.

3.5.27 sort

```
sort=nnn
```

Optional field. Used to override the default sorting order of FEATURE/INCREMENT lines; *nnn* specifies the relative sort order. The default sort order value 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.

Licenses are automatically sorted when they are processed by FLEX*lm*; the default sorting rules are is as follows:

1. License file. Automatic sorting does not occur across files in a license-file list.

2. Feature name.
3. FEATURE before INCREMENT.
4. Uncounted before counted.
5. Version, lower versions before higher versions.
6. Issued date, in reverse order, newest first. The date is taken from ISSUED= or START=.
7. Original order is otherwise maintained.

To turn off automatic ordering, 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().

3.5.28 user_info

```
user_info="..."
```

Optional field. Additional information provided by the software end user's license administrator. Not encrypted into the feature's signature.

3.5.29 vendor_info

```
vendor_info="..."
```

Optional field. Additional information provided by the software vendor. Not encrypted into the feature's signature.

3.6 UPGRADE Lines

```
UPGRADE feature vendor from_feat_version to_feat_version \
        exp_date num_lic [options ... ] SIGN=sign
```

All the data is the same as for 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-2005 5 SIGN=9BFAC03164ED
UPGRADE f1 demo 1.0 2.0 1-jan-2005 2 SIGN=1B9A30316207
```

result in 3 licenses of v1.0 of "f1" and 2 licenses of v2.0 of "f1."

And, the three lines:

```
INCREMENT f1 demo 1.0 1-jan-2005 5 SIGN=9BFAC03164ED
INCREMENT f1 demo 2.0 1-jan-2005 4 SIGN=8BF3fb031643
UPGRADE f1 demo 1.0 3.0 1-jan-2005 2 SIGN=1B9A30316207
```

result in 3 licenses for v1.0, the original 4 licenses for v2.0, and 2 licenses for v3.0 (taken from the original group of five v1.0 licenses).

Now consider this scenario:

```
INCREMENT f1 demo 1.0 1-jan-2005 5 SIGN=9BFAC03164ED
INCREMENT f1 demo 2.0 1-jan-2005 4 SIGN=8BF3fb031643
UPGRADE f1 demo 1.0 3.0 1-jan-2005 10 SIGN=afb303162056
```

This results in 9 licenses for v3.0 (5 v1.0 plus 4 v2.0 all upgraded to v3.0) 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-2005 8 SIGN=9BFAC03164ED
UPGRADE f1 demo 1.0 2.0 1-jan-2005 5 SIGN=afb303162056
UPGRADE f1 demo 1.0 3.0 1-jan-2005 3 SIGN=afb303162056
```

This upgrade scenario results in breaking up the one license pool of 8 licenses for v1.0 into two license pools: one with 5 licenses for v2.0 and one with 3 licenses for v3.0. A checkout request of 6 licenses for v1.0 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 v1.0.

3.7 PACKAGE Lines

```
PACKAGE package vendor [pkg_version] COMPONENTS=pkg_list \
        [OPTIONS=SUITE|SUITE_RESERVED] \
        [SUPERSEDE["p1 p2 ..."]ISSUED=date] \
        SIGN=pkg_sign
```

where:

package	Name of the package. The corresponding FEATURE/INCREMENT/UPGRADE line must have the same name.
vendor	Name of the vendor daemon that supports this package (*VENDOR_NAME* in `lm_code.h`).
pkg_version	Optional version of the package. If specified, the corresponding FEATURE/INCREMENT/UPGRADE line must have the same version.
pkg_sign	Signature generated by one of the license generators: `makepkg`, `lmcrypt`, or the vendor's customized license generator.
pkg_list	A space-separated 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 left out, their values come from the corresponding FEATURE/INCREMENT/UPGRADE 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`	This is what distinguishes a package suite from a package used to facilitate distribution. With `OPTIONS=SUITE`, the package FEATURE is checked out in addition to the component feature being checked out. See Section 3.7.3, “PACKAGE SUITE Example,” 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 Section 3.8, “Decimal Format Licenses,” 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 ISSUED dates prior to *dd-mmm-yyyy*.
`ISSUED=dd-mmm-yyyy`	Optional field, but if used, use with SUPERSEDE. Replaces all PACKAGE lines for the same package name with ISSUED dates prior to *date*.

The purpose of the PACKAGE line is to support different licensing needs:

1. To provide a way of distributing one line for a license file that has a large number of features, which largely share the same FEATURE line arguments.
2. To license a product suite.

A PACKAGE line, by itself, does not license anything—it requires a corresponding FEATURE/INCREMENT line to license the whole package. A PACKAGE line can be shipped with a product, independent of any licenses. Later, you can issue one or more corresponding FEATURE/INCREMENT lines that will enable the package. PACKAGE lines can be kept in a separate license file. The path to the package file should be specified in the application to support this transparently, via LM_A_LICENSE_DEFAULT.

3.7.1 PACKAGE Example

```
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 above PACKAGE and FEATURE line, note the following:

- The FEATURE line *enables* the PACKAGE line.
- The 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, since it contains no customer-specific fields.
- PACKAGE lines can be shipped in a separate file that never needs end-user editing, so long as the file is include in the license-file list.
- These PACKAGE and FEATURE lines, together, are a more efficient way of delivering the following 7 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
```

3.7.2 PACKAGE SUITE_RESERVED Example

With the OPTIONS=SUITE_RESERVED keyword, package component licenses are granted to a fixed number of users at any one time. In addition, once a user checks out a license for one component, licenses for the other package components are reserved by the license server for that user. A set of package components remain reserved for the given user until all licenses for the set are checked back in.

The SUITE_DUP_GROUP FEATURE-line keyword must be specified, in conjunction with OPTIONS=SUITE_RESERVED PACKAGE-line keyword, in order to get 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 \
        SUITE_DUP_GROUP=U DUP_GROUP=NONE 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 once one component is checked out, the others in the package are reserved for that user. This concept is illustrated in the following table with two users.

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`.

License Checkout Attempt	User	Requested Feature	Result	Explanation
#7	u2	`write`	granted	User `u2` rightfully gets the second `write` license.
#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.

Now, consider this variation where one user grabs both package licenses:

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

License Checkout Attempt	User	Requested Feature	Result	Explanation
#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 example depicted above, 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.

DUPLICATE GROUPING CONSIDERATIONS

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.

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 \
        SUITE_DUP_GROUP=U 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

- Section 7.3.10, “LM_A_LICENSE_DEFAULT”

3.7.3 PACKAGE SUITE Example

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 component of the package rather than being applied to one specific feature.

Various license sharing scenarios are created within the package suite paradigm using feature and suite duplicate grouping. 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 Section 3.5.7, “DUP_GROUP,” and Section 3.5.20, “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 \
        SUITE_DUP_GROUP=U 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.

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 \
        SUITE_DUP_GROUP=U DUP_GROUP=NONE 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. Once the licenses are consumed, no further licenses are available for checkout. This concept is illustrated in the following table.

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	

License Checkout Attempt	User	Requested Feature	Result	Explanation
#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 `paint` 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, via `OPTIONS=SUITE` on the PACKAGE line, named `office` with three components: `write`, `paint`, and `draw`. One license is available and shared among the three components. Once the license is consumed for one of the components, no further licenses are available for checkout. This concept is illustrated in the following table.

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

3.8 Decimal Format Licenses

Licenses can be represented in decimal format, to make license delivery easier for customers without access to email. Decimal has the advantage that it's simpler to type in, and often the licenses are much shorter.

To generate a decimal format license, use the `-decimal` argument for `lmcrypt` or `makekey`.

To convert an existing license to decimal, use `lmcrypt -decimal`, or

```
lminstall -i infile -o outfile -odecimal
```

If needed, decimal lines can be mixed with readable format lines in a license file.

End users will normally use the `lminstall` command to install decimal format licenses. Note that `lminstall` converts the decimal lines to readable format. `lminstall` does not, however, know where your application expects to find the license file. You will need to make the license file location clear to the end user. Refer to the *FLEXlm End Users Guide* for more information on `lminstall`.

3.8.1 Decimal Format Limitations

PACKAGE lines cannot be represented in decimal format. These can be shipped separately, shipped in the license file in readable format, or (preferably) pre-installed as part of the normal application installation. PACKAGE lines are not available in decimal format because they would be excessively long, because they consist mostly of component names.

FEATURESET lines also cannot be represented in decimal format.

Very long FEATURE lines will be extremely long in decimal format. If a license is very long in the normal format (say > 100 characters), it could be up to three times longer in decimal format, defeating the purpose of the format.

Feature names that include "-" cannot be represented in decimal format. This is unsupported by FLEX*lm*, although some companies have used it.

3.8.2 Example Decimal Licenses

COUNTED LICENSE:

```
SERVER this_host 12345678
VENDOR demo
FEATURE f0 demo 1.0 permanent 1 SIGN=A7F6DFD8C65E
FEATURE f1 demo 1.0 permanent 1 SIGN=AA8BD581EE65
```

Decimal format:

```
demo-f0-16641-00780-63392-57302-22216-00830-23011-18641-4
demo-f1-16641-00780-35488-34267-28385-54
```

Note that the first decimal line includes the SERVER/VENDOR information, and the second (and any subsequent lines) are much shorter.

DEMO LICENSE:

```
FEATURE f2 demo 1.0 1-jun-2005 uncounted HOSTID=DEMO \
        SIGN=6E06CC47D2AB
```

Decimal format:

```
demo-f2-23169-24979-00024-12403-47718-23830-1
```

3.8.3 Format of a Decimal License

Decimal format licenses have a fixed format which is easy to recognize:

```
vendor-feature-#####-#####-[...]
```

vendor	Vendor daemon name.
feature	Feature name.
#####	Groups of five decimal numbers (0-9) separated by a hyphen. The last group may be less than five digits.

The line includes a checksum, which can detect all single-digit errors and most multi-digit errors in lines that are typed incorrectly.

3.8.4 Hints on Using the Decimal Format

There are some “tricks” that are used internally to make decimal lines shorter. Knowledge of these can be useful when designing FEATURE lines.

TEXT IN OPTIONAL ATTRIBUTES

Text in the optional feature attributes are normally three times longer in the decimal format than in the “normal” format. For example: VENDOR_STRING=“limit 3” would require about 21 characters in the decimal version. There’s a trick to making this shorter: If the text portion is a decimal or hex number, then it’s stored compressed in the decimal version, and the conversion is about 1:1 instead of 1:3.

For example: VENDOR_STRING=12345 consumes about five characters in the decimal format. VENDOR_STRING=abcd (valid hex characters) will also consume about five characters in the decimal format. Knowing this, you might choose to “encode” information in the VENDOR_STRING in a numeric format. This enhancement only applies to numbers <= 0xffffffff. For example, VENDOR_STRING=12345678901234 will require about 14*3 = 42 characters in the decimal format.

Note: Mixed-case hex characters will not be stored efficiently. VENDOR_STRING=abcD will take about twelve decimal characters, instead of five.

FEATURE NAMES

Avoid underscore "_" in feature names; it's hard to distinguish from a hyphen "-." For example:

```
demo-prod_1a-10449-31786-63556-56877-09398-10373-137
```

This is hard to read, and if the user mixes up the "-" and "_", the license will be invalid. Since you also can't use "-" in a feature name, this means that feature names won't have any kind of separator. Therefore, in the example, we suggest simply "prod1a."

EXPIRATION DATES

For non-expiring licenses, use "permanent" or "1-jan-0" as the expiration date. Some older format, but still valid, expiration dates are not supported in the decimal format. For example: "3-mar-0" is functionally identical to "permanent," but because the decimal format supports only "permanent" or "1-jan-0," "3-mar-0" is unsupported. Dates farther in the future require many decimals to represent. Therefore 1-jan-9999 takes about 14 characters while "permanent" requires about 1.

SEE ALSO

- `lminstall` in the *FLEXlm End Users Guide*

Chapter 4

Trivial API

4.1 Overview of the Trivial API

The Trivial API provides a basic interface to the FLEX*lm* client library routines. It is differentiated from the other FLEX*lm* APIs by exposing only scalar data types and requiring that only one feature can be checked out at a time from a single application. This API is not thread-safe because it internally passes context between calls to checkout/heartbeat/checkin, and between error/information/warning functions and other internal functions.

This API is available in two formats:

- Function-based (Windows only)

  This format is for applications that must invoke functions rather than C macros. This implementation is available to applications that link the FLEX*lm* client library dynamically. The function-based Trivial API consists of functions that call the FLEXible API. The only required header file is `lmpolicy.h`. Function names start with the `lt_` prefix.

- Macro-based

  This format is for applications that can invoke C macros. This implementation is available to applications that link the FLEX*lm* client library either statically or dynamically. The macro-based Trivial API consists of macros that call the Simple API. The only required header file is `lmpolicy.h`, and no other macros or function calls are needed. Macros are named with uppercase letters.

The Trivial API is divided into the following functional categories:

- License acquisition
- Heartbeat management
- Error and warning processing

Note: You cannot mix function-based and macro-based Trivial API calls nor Trivial API calls with any other API calls.

Programming examples using the Trivial API and instructions for building your licensed application are located in the *FLEXlm Programmers Guide*.

4.1.1 License Acquisition

These are the basic functions which acquire and release a license. They are required and constitute the minimum Trivial API implementation.

Function	Description
lt_checkin(), CHECKIN()	Releases a license and frees all FLEX*lm* memory.
lt_checkout(), CHECKOUT()	Acquires a license.

4.1.2 Heartbeat Management

This function is used for manual heartbeat implementations. If automatic heartbeats are specified by the call to CHECKOUT()/lc_checkout() in the FLEX*lm*-enabled application, this function is not explicitly used.

Function	Description
lt_heartbeat(), HEARTBEAT()	Sends a heartbeat, manually, to the license server.

4.1.3 Error and Warning Processing

These functions provide optional error and warning processing.

Function	Description
lt_errstring(), ERRSTRING()	Returns a string describing the most recent error.

Function	Description
lt_perror(), PERROR()	Presents current error message to user.
lt_pwarn(), PWARN()	Presents current warning message to user.
lt_warning(), WARNING()	Returns a string describing the most recent warning.

4.2 Trivial API Descriptions

4.2.1 lt_checkin(), CHECKIN()

SYNTAX

```
(void) lt_checkin()
(void) CHECKIN()
```

DESCRIPTION

Releases the license for the feature and frees memory associated with the checkout.

4.2.2 lt_checkout(), CHECKOUT()

SYNTAX

```
status = lt_checkout(policy, feature, version,
                     license_file_list)
status = CHECKOUT(policy, feature, version, license_file_list)
```

DESCRIPTION

Acquires a license for a feature.

PARAMETERS

`(int) policy`	See Section 7.1, “Trivial and Simple API License Policies.” Example: LM_RESTRICTIVE.
`(char *) feature`	The feature name to check out.

`(char *) version`	The version of the feature to check out. This is a string in floating-point format (e.g., 12345.123). If the license in the license file has the same version number or a higher version number, the checkout will succeed. Macrovision recommends that this version number be a license version level and *not* the application's version number. This version number should only be changed when you want old licenses to no longer work with a new version of the software.
`(char *) license_file_list`	A location in your installation hierarchy which indicates the expected license file location. This is a directory containing one or more license files with a `.lic` extension. If 0, this argument is unused. See the *FLEXlm Programmers Guide* for more information on how this location is used by the licensed application.

Upon success, the path to the license file used is set in *VENDOR*`_LICENSE_FILE` in the registry on Windows (`\HKEY_LOCAL_MACHINE\Software\FLEXlm License Manager`) and `$HOME/.flexlmrc` on UNIX.

RETURN

`(int) status`	0 if successful; otherwise, the FLEX*lm* error number.

SEE ALSO

- Section 7.1, "Trivial and Simple API License Policies"
- Section 7.2, "Trivial and Simple API Policy Modifiers"

4.2.3 lt_errstring(), ERRSTRING()

SYNTAX

```
string = lt_errstring()
string = ERRSTRING()
```

DESCRIPTION

Returns a string describing the last FLEX*lm* error set.

RETURN

`(char *) string`	An explanatory string.

4.2.4 lt_heartbeat(), HEARTBEAT()

SYNTAX

```
status = lt_heartbeat()
status = HEARTBEAT()
```

DESCRIPTION

lt_heartbeat()/HEARTBEAT() sends heartbeat messages to and receives acknowledgments from the license server. By default, these activities are handled automatically by FLEX*lm* via a separate, dedicated application thread. See Section 12.1, “Automatic Heartbeats,” for more details on automatic heartbeat messages.

This function provides manual control of heartbeat messages; thereby, overriding the automatic mechanism. To use lt_heartbeat()/HEARTBEAT(), you must first turn off the automatic mechanism by setting the LM_MANUAL_HEARTBEAT policy modifier in the lt_heartbeat()/HEARTBEAT() call.

If the license server goes down and later comes back up, lt_heartbeat()/HEARTBEAT() automatically reconnects and checks the license out. Each call to lt_heartbeat()/HEARTBEAT() makes one attempt to reconnect. On failure, it returns the cumulative number of failed attempts to reconnect to the license server, in which case, applications should at a minimum notify the user of the failure. This provides a way for applications to monitor the number of reconnects and take appropriate action, such as exiting, when a predetermined limit is reached. In addition, applications may want to exit if reconnections succeed more than three or four times in a relatively short period (e.g. ten minutes), which may indicate a user restarting the license server in an attempt to acquire extra licenses.

Do not call lt_checkout()/CHECKOUT() on failure from lt_heartbeat()/HEARTBEAT()—this is not necessary and will cause problems if attempted.

RETURN

`(int)` *`status`*	0 if successful; otherwise, it returns the number of failed attempts to reconnect to the server.

SEE ALSO

- Section 7.2.1, “LM_MANUAL_HEARTBEAT”
- Section 7.2.2, “LM_RETRY_RESTRICTIVE”
- Chapter 12, “Heartbeats”

4.2.5 lt_perror(), PERROR()

SYNTAX

```
(void) lt_perror(string)
(void) PERROR(string)
```

DESCRIPTION

Presents *`string`* and a description of the most recent error to the user. On Windows this appears in a dialog.

PARAMETERS

`(char *)` *`string`*	A string describing the error context.

4.2.6 lt_pwarn(), PWARN()

SYNTAX

```
(void) lt_pwarn(string)
(void) PWARN(string)
```

DESCRIPTION

Presents *`string`* and a description of the most recent warning to the user. On Windows this appears in a dialog. This is useful with policy set to LM_LENIENT or LM_FAILSAFE. Nothing is printed if there is no warning.

PARAMETERS

(char *) *string*	A string describing the error context.

4.2.7 lt_warning(), WARNING()

SYNTAX

```
string = lt_warning()
string = WARNING()
```

DESCRIPTION

Returns a string describing the last FLEX*lm* warning.

RETURN

(char *) *string*	An explanatory string. This is useful with policy set to LM_LENIENT or LM_FAILSAFE.

Chapter 5

Simple API

5.1 Overview of Simple API Functions

The Simple API can do nearly everything the FLEXible API can do. Use this API if your application requires checking out more than one feature name at a time or if you need to acquire more than one license for a feature. This API requires that you include the `lmpolicy.h` header file.

The functions are divided into the following categories:

- License acquisition
- Heartbeat management
- Error and warning processing

Note: You cannot mix calls to Simple API functions with calls to any other API functions.

Programming examples using the Simple API and instructions for building your licensed application are located in the *FLEXlm Programmers Guide*.

5.1.1 Checkin and Checkout Functions

These are the basic functions which acquire and release a license. They are required and constitute the minimum Simple API implementation.

Function	Description
lp_checkin()	Releases a license and frees all FLEX*lm* memory.
lp_checkout()	Acquires a license.

5.1.2 Heartbeat Function

This function is used for manual heartbeat implementations. If automatic heartbeats are employed in the client application, this function is not explicitly called.

Function	Description
lp_heartbeat()	Sends a manual heartbeat to the license server.

5.1.3 Error and Warning Processing Functions

These functions provide optional error and warning processing. They are not required in the minimal implementation.

Function	Description
lp_errstring()	Returns a string describing the most recent error.
lp_perror()	Presents current error message to user.
lp_pwarn()	Presents current warning message to user.
lp_warning()	Returns a string describing the most recent warning.

5.2 Simple API Function Descriptions

5.2.1 lp_checkin()

SYNTAX

```
(void) lp_checkin(lp_handle)
```

DESCRIPTION

Releases a license, and frees memory associated with the corresponding checkout. lp_checkin() should be called even if the checkout fails, in order to free associated memory and resources.

PARAMETER

`(LP_HANDLE *) lp_handle`	The handle from the lp_checkout() call.

5.2.2 lp_checkout()

SYNTAX

```
#include "lmpolicy.h"
LP_HANDLE *lp_handle;
status = lp_checkout(LPCODE, policy, feature, version, num_lic,
                           license_file_list, &lp_handle)
```

DESCRIPTION

Acquires a license for a feature.

PARAMETERS

`(LPCODE_HANDLE *) LPCODE`	From the `lmpolicy.h` header file. Use the literal `LPCODE`.
`(int) policy`	See Section 7.1, “Trivial and Simple API License Policies.” Example: LM_RESTRICTIVE.
`(char *) feature`	The desired feature name to check out.

`(char *) version`	The version of the feature to check out. This is a string in floating-point format (e.g., 12345.123). If the license in the license file has the same version number or a higher version number, the checkout will succeed. Macrovision recommends that this version number be a license version level and *not* the application's version number. This version number should only be changed when you want old licenses to no longer work with a new version of the software
`(int) num_lic`	The number of licenses to check out. Usually this number is 1.
`(char *) license_file_list`	A location in your installation hierarchy which indicates the expected license file location. This is a directory containing one or more license files with a `.lic` extension. If 0, this argument is unused. See the *FLEXlm Programmers Guide* for more information on how this location is used by the licensed application.
`pointer to (LP_HANDLE *) lp_handle`	This is the return handle, and is used for subsequent calls that apply to this checkout, e.g., lp_checkin(), lp_errstring(), etc. If lp_checkout() is called more than once, separate `lp_handle` variables must be declared and used, and the corresponding handle must be used with the other lp_*xxx*() (Simple API) calls.

Upon success, the path to the license file used is set in *VENDOR*_LICENSE_FILE in the registry on Windows (\HKEY_LOCAL_MACHINE\Software\FLEXlm License Manager) and $HOME/.flexlmrc on UNIX.

RETURN

(int) *status*	0 if successful; otherwise, the FLEX*lm* error number.

To check out two features:

```
LP_HANDLE *lp_handle1;
LP_HANDLE *lp_handle2;
lp_checkout(LPCODE, LM_RESTRICTIVE, "f1", "1.0", 1,
            "a/b/c/license.lic", &lp_handle1);
lp_checkout(LPCODE, LM_RESTRICTIVE, "f2", "1.0", 1,
            "a/b/c/license.lic", &lp_handle2);
```

5.2.3 lp_errstring()

SYNTAX

string = lp_errstring(lp_handle)

DESCRIPTION

Returns a string describing the previous FLEX*lm* error.

PARAMETER

(LP_HANDLE *) lp_handle	The handle from the lp_checkout() call.

RETURN

(char *) *string*	Error description.

5.2.4 lp_heartbeat()

SYNTAX

```
status = lp_heartbeat(lp_handle, num_reconnects, num_minutes)
```

DESCRIPTION

lp_heartbeat() sends heartbeat messages to and receives acknowledgments from the license server. By default, these activities are handled automatically by FLEX*lm* via a separate, dedicated application thread. See Section 12.1, "Automatic Heartbeats," for more details on automatic heartbeat messages.

This function provides manual control of heartbeat messages; thereby, overriding the automatic mechanism. To use lp_heartbeat(), you must first turn off the automatic mechanism by setting the LM_MANUAL_HEARTBEAT policy modifier in the lp_checkout() call.

If the license server goes down and later comes back up, lp_heartbeat() automatically reconnects and checks the license out. On failure, it returns the cumulative number of failed attempts to reconnect to the license server, each call to lp_heartbeat() making one attempt. In which case, applications should at a minimum notify the user of the failure. Applications can set a limit to the number of reconnects before exiting. In addition, applications may want to exit if reconnections succeed more than three or four times in a relatively short period (e.g. ten minutes), which may indicate a user restarting the license server in an attempt to acquire extra licenses.

PARAMETERS

`(LP_HANDLE *) lp_handle`	The handle from the lp_checkout() call.
`(int *) num_reconnects`	Pointer to `int`. If null, this argument is ignored. If non-null, and the client has just successfully reconnected to the server, the return value will be 0 (success), and *`num_reconnects`* is set to the number of times the client has reconnected in the last minutes. If this is a large number, it may indicate attempted theft.

(int) *num_minutes*	If 0, this argument is ignored. If non-zero, it's used to detect when a server is being started and stopped many times in a short period, which can indicate attempted theft. The reporting period is set with *num_minutes*.

RETURN

(int) *status*	0 if successful; otherwise, it returns the number of failed attempts to reconnect to the server.

SEE ALSO

- Section 7.2.1, "LM_MANUAL_HEARTBEAT"
- Chapter 12, "Heartbeats"

5.2.5 lp_perror()

SYNTAX

```
(void) lp_perror(lp_handle, string)
```

DESCRIPTION

Presents *string* and a description of the most recent error to the user. On Windows this appears in a dialog.

PARAMETERS

(LP_HANDLE *) lp_handle	The handle from the lp_checkout() call.
(char *) *string*	A string describing the error context.

5.2.6 lp_pwarn()

SYNTAX

```
(void) lp_pwarn(lp_handle, string)
```

DESCRIPTION

Presents *string* and a description of the most recent warning to the user. On Windows this appears in a dialog; on other systems, it prints to stderr. This is useful with policy set to LM_LENIENT or LM_FAILSAFE. Nothing is printed if there is no warning.

PARAMETERS

`(LP_HANDLE *) lp_handle`	The handle from the lp_checkout() call.
`(char *) string`	A string describing the error context.

5.2.7 lp_warning()

SYNTAX

```
string = lp_warning(lp_handle)
```

DESCRIPTION

Returns a string describing the last FLEX*lm* warning.

PARAMETERS

`(LP_HANDLE *) lp_handle`	The handle from the lp_checkout() call.

RETURN

`(char *) string`	An explanatory string. This is useful with policy set to LM_LENIENT or LM_FAILSAFE.

Chapter 6

FLEXible API

This is the most powerful API available for license management. As such, it contains many options enabling considerable flexibility. Where possible, new licensed applications should use the Trivial or Simple APIs which are documented in Chapter 4, “Trivial API” and Chapter 5, “Simple API,” respectively. There is, however, no reason to change APIs in applications which already use the FLEXible API. Some FLEX*lm* functionality is available only in this API. For example, the C interface to license generation, lc_cryptstr(), is available only in the FLEXible API.

6.1 FLEXible API Function Summary

This section lists the popularly used FLEXible API functions. Full descriptions for them are found later in this chapter. For a grouping by function category, see Section 6.2, “FLEXible API Functions by Category.”

Programming examples using the FLEXible API and instructions for building your licensed application are located in the *FLEXlm Programmers Guide*.

Table 6-1: FLEXible API Function Summary

Function Name	Description
lc_auth_data()	Gets the license file line for a checked-out feature.
lc_checkin()	Returns a license of a feature to the license pool.
lc_checkout()	Requests a license of a feature.
lc_cryptstr()	Generates a valid signature in a feature line.
lc_err_info()	Returns error information to an LM_ERR_INFO structure pointer.

Table 6-1: FLEXible API Function Summary (Continued)

Function Name	Description
lc_errstring()	Returns the FLEX*lm* error string for the most recent FLEX*lm* error.
lc_expire_days()	Returns the number of days until a license expires.
lc_feat_list()	Gets the list of all features in the license file
lc_first_job()	Locates the first job on the list of jobs.
lc_free_job()	Frees the memory associated with a job.
lc_get_attr()	Retrieves a FLEX*lm* attribute.
lc_heartbeat()	Exchanges heartbeat messages with the license server.
lc_hostid()	Provides the hostid in string format.
lc_idle()	Informs the license server when the licensed application is idle.
lc_init()	Creates a job handle for use by lc_cryptstr().
lc_log()	Logs a message in the debug log file.
lc_new_job()	Initializes FLEX*lm* and creates a new job.
lc_next_job()	Walks the list of jobs.
lc_perror()	Prints a FLEX*lm* error message.
lc_set_attr()	Sets a FLEX*lm* attribute.
lc_status()	Returns the status of the requested feature.
lc_userlist()	Provides a list of who is using a feature.
lc_vsend()	Sends a message to the vendor daemon and returns a result string.

SEE ALSO

- Section 8.1, “Advanced FLEXible API Functions”
- Section D.1, “Obsolete FLEXible API Functions”

6.2 FLEXible API Functions by Category

Incorporating the FLEXible API into your application involves adding calls to functions from several different categories. In the following sections, the API functions are divided into categories to help guide you through the implementation process. Each category includes the API functions and related attributes that are used with the lc_set_attr() and lc_get_attr() functions. Full descriptions for the FLEXible API functions are presented later in this chapter and the attributes are described in Section 7.3, “FLEXible API Attributes set by lc_set_attr().”

6.2.1 Checkin and Checkout

The licensed application interfaces to FLEX*lm* via a set of routines that request (check out) and release (check in) licenses of selected feature(s). Table 6-2 lists the API functions and attributes related to checking in and checking out a feature.

Table 6-2: Checkin and Checkout

Function or Attribute Name	Description
Functions	
lc_auth_data()	Gets the license file line for a checked-out feature.
lc_checkin()	Returns a license of a feature to the license pool.
lc_checkout()	Requests a license of a feature.
lc_status()	Returns the status of the requested feature.
Attributes	
LM_A_BORROW_EXPIRE	Sets the date and time when a borrowed license expires.
LM_A_CHECK_BADDATE	Checks system date on the client node.
LM_A_LICENSE_DEFAULT	The expected license file location.
LM_A_LINGER	Controls the license linger time.

Table 6-2: Checkin and Checkout (Continued)

Function or Attribute Name	Description
LM_A_PROMPT_FOR_FILE (Windows Only)	Prompts the user for a license-file path or server name.

6.2.2 Job Handling

Each licensed application must establish at least one connection to a license server. This connection is called a “job” and is managed by a job handle. Job handling API functions manage job handles. Table 6-3 lists these functions.

Table 6-3: Job Handling

Function Name	Description
lc_first_job()	Locates the first job on the list of jobs.
lc_free_job()	Frees the memory associated with a job.
lc_new_job()	Initializes FLEX*lm* and creates a new job.
lc_next_job()	Walks the list of jobs.

6.2.3 Heartbeat Management and Communication

Heartbeat exchange is the mechanism by which the licensed application keeps track of the license server status. The vendor implements heartbeats as either automatic or manual. If manual heartbeats are implemented, the API functions and attributes in Table 6-4 are used.

Table 6-4: Heartbeat Management and Communication

Function or Attribute Name	Description
Functions	
lc_heartbeat()	Exchanges heartbeat messages with the license server. Used for manual heartbeat implementations.

Table 6-4: Heartbeat Management and Communication (Continued)

Function or Attribute Name	Description
lc_idle()	Informs the license server when the licensed application is idle.
Attributes	
LM_A_CHECK_INTERVAL	Controls the licensed application's detection of license server failures.
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL	Specifies the number of and the interval between automatic reconnection attempts.
LM_A_TCP_TIMEOUT	Specifies amount of time to wait, after TCP disconnection, before license is automatically checked in.
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX	Pointer to a function that receives control if reconnection fails.
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX	Pointer to a function that is called just before a reconnection attempt.
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX	Pointer to a function that is called after a successful reconnection.

6.2.4 Informational

Table 6-5 lists the informational functions and attributes. They provide status and usage information not critical to the license management process.

Table 6-5: Informational

Function or Attribute Name	Description
Functions	
lc_expire_days()	Returns the number of days until a license expires.
lc_feat_list()	Gets the list of all features in the license file
lc_log()	Logs a message in the debug log file.
lc_userlist()	Provides a list of who is using a feature. Helpful to explain why a license is denied.
lc_vsend()	Sends a message to the vendor daemon and returns a result string.
Attributes	
LM_A_BORROW_STAT	Provides an programming interface to the lmborrow functionality.
LM_A_LF_LIST	Lists all license files searched for features.
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO	Retrieves information from the vendor daemon.
LM_A_VERSION, LM_A_REVISION	Retrieves information regarding the version and revision of the FLEX*lm* client libraries.

6.2.5 Error and Warning Reporting

These functions provide a way to report errors and warnings from the licensed application to the user. These functions and attributes are listed in Table 6-6.

Table 6-6: Error and Warning Reporting

Function or Attribute Name	Description
Functions	
lc_err_info()	Used for internationalization.
lc_errstring()	Returns the FLEX*lm* error string for the most recent FLEX*lm* error.
lc_perror()	Prints a FLEX*lm* error message to stderr on UNIX and to a message box on Windows.
Attributes	
LM_A_LONG_ERRMSG	Presents error messages in the long format.
LM_A_PERROR_MSGBOX (Windows Only)	Presents the error message in an error dialog box.
LM_A_WINDOWS_MODULE_HANDLE (Windows only)	Enables dialogs and error messages to be displayed properly (used with DLL only)

6.2.6 License File Installation

Vendors may want to supply a utility along with their FLEX*lm* licensed application that performs license file installation. Table 6-7 lists the API functions and attributes related to license file installation.

Table 6-7: License File Installation

Function or Attribute Name	Description
Functions	
lc_hostid()	Provides the hostid in string format.
Attributes	
LM_A_FLEXLOCK	Turns on FLEX*lock* capability.
LM_A_FLEXLOCK_INSTALL_ID	Tags the FLEX*lock* operation with a random number.

6.2.7 License File Generation

These functions and attributes are provided to the vendor who wishes to develop a proprietary license file generation utility. Table 6-8 lists them.

Table 6-8: License File Generation

Function or Attribute Name	Description
Functions	
lc_cryptstr()	Generates a valid signatures for a license file.
lc_init()	Creates a job handle for use by lc_cryptstr().
Attributes	
LM_A_LICENSE_FMT_VER	Specifies the version format of the license that lc_cryptstr() generates.

6.3 Commonly Used FLEXible API Functions

Table 6-9 gives lists the most commonly used FLEXible API functions. Full descriptions for the FLEXible API functions are presented later on in this chapter.

Table 6-9: Commonly Used Functions

Function Name	Description
lc_auth_data()	Gets the license file line for a checked-out feature. Often used to display license information to the user.
lc_checkin()	Returns a license of a feature to the license pool.
lc_checkout()	Requests a license of a feature.
lc_errstring()	Returns an explanatory error string for the most recent error.
lc_expire_days()	Returns the number of days until a license expires.
lc_free_job()	Frees a job allocated with lc_new_job().
lc_heartbeat()	Sends heartbeat from client application to license server. Used for manual heartbeat implementations.
lc_new_job()	Initializes FLEX*lm* and creates a license job.
lc_perror()	Prints an error message to stderr or error message dialog on Windows.
lc_set_attr()	Sets a FLEX*lm* client attribute.

These and additional functions are documented in this chapter, and other functions are documented in Appendix D, “Obsolete FLEXible API Features.” It is rare that an application will require the functions in this appendix, and care should be used when calling them.

6.4 FLEXible API Function Descriptions

6.4.1 lc_auth_data()

SYNTAX

```
conf = lc_auth_data(job, feature)
```

DESCRIPTION

Gets authenticated information corresponding to the specific license-file line for a feature that has been checked out. This is the only way to retrieve authenticated information for a checked out feature. Use a separate call to lc_auth_data() for each checked out feature.

The following code excerpt demonstrates using this function. In this example, the application obtains the feature line for the feature just checked out in order to process the NOTICE field. For this example, consider the following feature line from the license file:

```
FEATURE f1 demo 1.0 permanent 4 NOTICE="ab" \
        SIGN=xxxxxxxxxxxx
```

The code appears as follows in the application:

```
CONFIG *conf;

/* after the call to lc_new_job() */
if (lc_checkout(job, feature, ...))
{
     /* error processing...*/
}
else
{
     conf = lc_auth_data(job, feature);
     /* Use conf->lc_notice for intellectual property info. */
     if(conf->lc_notice[0] == 'a')
     {
        /* "a" processing */
     }
     else if(conf->lc_notice[1] == 'b')
     {
        /* "b" processing */
     }
}
```

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(char *) *feature*	The desired feature.

RETURN

(CONFIG *) *conf*	The CONFIG struct, or NULL if error. The CONFIG struct is defined in the header file lmclient.h.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOFEATURE	Feature not found.

Note: If you want a checkout to depend on information in the license file, call lc_set_attr() with LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX, or LM_A_CHECKOUTFILTERLAST_EX before the call to lc_checkout().

SEE ALSO

- lmclient.h for the CONFIG struct definition
- Section 6.4.11, “lc_get_attr()”
- Section 6.4.3, “lc_checkout()”
- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 8.2.4, “LM_A_CHECKOUTFILTERLAST_EX”

6.4.2 lc_checkin()

SYNTAX

```
(void) lc_checkin(job, feature, keep_conn)
```

DESCRIPTION

Checks in the licenses of the specified feature. For TCP clients, the daemon will detect the fact that the client exited, and return any licenses that were checked out back to the available pool. For TCP, this function is called if the application has need of a feature for a period of time, then no longer needs it. The second parameter is used for TCP clients to tell FLEX*lm* to keep the connection open to the server for cases where another feature will be needed shortly after this one is released. If the communications protocol is TCP, there is no appreciable time delay incurred in returning the license if the program exits rather than returning the license via lc_checkin(). For reporting purposes in the report log file, it is preferable to check in a license with lc_checkin() rather than simply exiting, because these two types of events are recorded differently in the report log file.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) feature`	The feature name to be checked in, or LM_CI_ALL_FEATURES.
`(int) keep_conn`	If non-zero, means “Keep connection to server.” If 0, drops TCP connection.

RETURN

None.

6.4.3 lc_checkout()

SYNTAX

```
status = lc_checkout(job, feature, version, num_lic, flag,
                     code, dup_group)
```

DESCRIPTION

Checks out one (or more) license(s) of the specified feature. If the process that calls lc_checkout() exits in any manner, then the checked-out license will be returned for re-use by another user. For TCP clients, the resulting checkin is immediate.

Place the call to lc_checkout() in an executable that is active whenever the user is using the feature. If *flag* is specified as LM_CO_WAIT, then the process will wait until the number of licenses requested for this feature are available. The license file must specify a version that is greater than or equal to the version in the call to lc_checkout().

If the license file is *counted*, that is, if the number of users specified on the FEATURE line is non-zero, lc_checkout() will request the license from a license server. If the number of users on the FEATURE line is *uncounted*, it will grant permission based on the contents of the license file only—hostid, version, expiration date, etc.

- Before a call to the checkout() function, it is recommended that the application first indicate the expected license file location, with:

  ```
  lc_set_attr(job, LM_A_LICENSE_DEFAULT, \
          (LM_A_VAL_TYPE)license_file_list)
  ```

  The *license_file_list* is a location in your installation hierarchy which indicates the expected license file location. This is a directory containing one or more license files with a `.lic` extension. If 0, this argument is unused. See the *FLEXlm Programmers Guide* for more information on how this location is used by the licensed application.
- Multiple checkout requests from the same process in the same license job will not result in additional licenses being checked out, unless a new request specifies more licenses than were previously checked out. That is, two calls to lc_checkout(`...,1,...`) will result in only one license being checked out, not two. A second call to request two licenses would result in a total of two licenses.
- For improved security, it is recommended that the parameters *feature*, *version*, etc., be "hidden"—the string should not be directly declared in source code. It should be built up chars or smaller strings, and then created via sprintf(). That way, it is more difficult for users to change the license being checked out by altering the string in the binary.

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().
`(char *)` *feature*	The ASCII feature name desired.

(char *) *version*	The version of the feature desired in floating point format, maximum of ten characters (e.g., “12345.123” or “123.456789”). This value must be <= the version number in the license file for the checkout to succeed. Letters are not allowed in versions. For example, “v1.0” is illegal.
(int) *num_lic*	The number of licenses to check out. (Must be > 0.)
(int) *flag*	The checkout option flag.

Possible values for *flag* are:

LM_CO_NOWAIT	Do not wait—non-blocking.
LM_CO_WAIT	Wait, return when license is granted—*blocking*.
LM_CO_QUEUE	Queue request, return immediately. This request will give you the license if it is available. You can find out if you hold the license by calling lc_status(). If there are multiple license pools, this queues from only the first license pool in the list.

LM_CO_LOCALTEST	With this flag, the license file is tested for errors that would prevent a license checkout; a license is not consumed when lc_checkout() is used with this flag. In particular, LM_MAXUSERS and LM_USERSQUEUED are not detected, but other checkout errors are detected. For example, you might want to gray out a menu item if a particular feature is not present in the license file. If you call lc_checkout() with the LM_CO_LOCALTEST flag, use the function lc_test_conf() to retrieve the license file line for the tested feature (lc_auth_data() won't work with the LM_CO_LOCALTEST flag). This can only be done after the most recent call to lc_checkout(). Testing a license file with this flag causes a connection to the license server system that lasts as long as the current job. Therefore, the job in which the test is performed is irrevocably connected to a specific license server system, which may not have any count available for subsequent license requests. Consider making this test in a separate job.
`pointer to (VENDORCODE) code`	From lc_new_job().
`(int) dup_group`	Duplicate grouping mask for this feature.

Requests for licenses from "duplicates" can either be "grouped" or "not grouped." Grouping duplicates allows license requests from separate processes to use a single license if the process's USER, HOST, DISPLAY, and/or VENDOR_DEFINED field are the same. Duplicate grouping is valid only with counted licenses.

The *dup_group* parameter allows you to select what to compare to constitute a group from the set {USER HOST DISPLAY VENDOR}. Any of the four fields that are not set to compare will automatically match; thus not setting any of the four fields yields a site license, since all users on all hosts on all displays are the same as far as the comparison is concerned.

The following examples illustrate the use of the duplicate grouping capability:

Value:	**Meaning:**
LM_DUP_NONE	Every process gets a new license.
LM_DUP_USER	All requests from this user name share the same license.
LM_DUP_HOST	All requests from this host name share the same licenses. This is a “floating node-locked” license.
LM_DUP_DISP	All requests from this display share the same license. (Useful for display or GUI based products, like a window system.)
LM_DUP_VENDOR	All requests with the same vendor-defined data, use the same license. (Useful for sharing licenses among otherwise unrelated processes.) If LM_DUP_VENDOR is used, LM_A_CHECKOUT_DATA must be set.
LM_DUP_USER | LM_DUP_HOST	All requests from this user name on this host name use the same license.

LM_DUP_USER | LM_DUP_DISP	All requests from this user name on this display use the same license. (One user, displaying on a single node, using several nodes to run the software.)
LM_DUP_USER | LM_DUP_HOST | LM_DUP_DISP	All requests from this user name on this host name using this display use the same license.
LM_DUP_USER | LM_DUP_VENDOR	All requests from this user name with the same vendor data use the same license. If LM_DUP_VENDOR is used, LM_A_CHECKOUT_DATA must be set.
LM_DUP_SITE	All requests from any user on any node on any display with any vendor data use the same license (site license).

RESERVE AND DUP_GROUP

There is an important interaction between RESERVE and the duplicate grouping mask. A license reservation for an entity not contained in the duplicate grouping mask in the call to lc_checkout() (e.g., a USER reservation) when the duplicate grouping mask is set to `LM_DUP_HOST | LM_DUP_DISP`) can cause an interesting interaction at run-time.

To understand why this is the case, consider the following example:

- Your software groups duplicates based on USER and DISPLAY.
- Your end user has a license file with a single license.
- Your end user reserves this license for HOST “nodea.”
- User “joe” on display “displaya” on HOST “nodea” checks out a license. He gets the license, since his HOST matches the reservation.
- User “joe” on display “displaya” on HOST “nodeb” checks out a license. He also gets a license, since he is grouped with the first license as a duplicate.

- The first user (joe/displaya/nodea) checks in his license.

At this point in the example, the situation appears to be inconsistent. The second user continues to hold the reservation, which means that a user on "nodeb" is using a license reserved for "nodea." Once this second user checks in the license, the reservation will return to the pool of reservations to be used by anyone on "nodea."

FLEX*lm* was designed to allow this potential temporary inconsistency rather than the alternative, which is to have an unusable reservation.

REGISTRY AND $HOME/.FLEXLMRC

Environment variables can be taken either from the environment or from the registry (on Windows) or `$HOME/.flexlmrc` (UNIX). After a successful checkout, the *`VENDOR`*`_LICENSE_FILE` variable is set for the location in the registry (Windows) or `$HOME/.flexlmrc` (UNIX). This way, all subsequent checkouts for features from this vendor will automatically use the license that worked previously. Note that this location is added to all other locations the application may look for the license.

This automatic registry update can be turned off with:

```
lc_set_attr(job, LM_A_CKOUT_INSTALL_LIC, (LM_A_VAL_TYPE)0);
```

RETURN

`(int)` *`status`*	0—OK, license checked out <> 0—Error

ERROR RETURNS

LM_BADCODE	Signature in license file does not match other data in file.
LM_BADHANDSHAKE	Authentication handshake with daemon failed.
LM_BADPARAM	FLEX*lm* function argument is invalid or there is an invalid setting in `lm_code.h`.

LM_BADSYSDATE	System clock has been set back. This error can only occur when the FEATURE line contains an expiration date.
LM_BAD_VERSION	Version argument is invalid floating point format.
LM_BORROW_TOOLONG	Cannot borrow a license for that many hours. The user has specified a borrow period longer than the license allows.
LM_BUSYNEWSERV	License server busy starting another copy of itself—retry.
LM_CANTCONNECT	Cannot establish a connection with a license server.
LM_FEATQUEUE	Feature is queued. lc_status() will indicate when it is available.
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_LOCALFILTER	Checkout request filtered by the vendor-defined filter routine.
LM_MAXLIMIT	Checkout exceeds MAX specified in options file.
LM_MAXUSERS	All licenses in use. Applications usually need to test for both LM_MAXUSERS and LM_USERSQUEUED instead of only LM_MAXUSERS.
LM_NO_SERVER_IN_FILE	No license server specified for counted license.
LM_NOBORROW_SUPP	License BORROW support not enabled, but borrowing was requested by the user.
LM_NOFEATURE	Cannot find feature in the license file.

LM_NOSERVSUPP	Server has different license file than client—client's license has feature, but server's does not.
LM_OLDVER	License file does not support a version this new.
LM_PLATNOTLIC	This platform is not authorized by the license—running on a platform not included in PLATFORMS="..." list.
LM_SERVBUSY	License server busy—the request should be retried. (This is a rare occurrence.)
LM_USERSQUEUED	Like LM_MAXUSERS, but also indicates that there are already some users queued. Applications usually need to test for both LM_MAXUSERS and LM_USERSQUEUED instead of only LM_MAXUSERS.

Note: If you want a checkout to depend on information in the license file, call lc_set_attr() `LM_A_CHECKOUTFILTER`, `LM_A_CHECKOUTFILTER_EX`, or `LM_A_CHECKOUTFILTERLAST_EX` before the call to lc_checkout().

If you want to display information to the user from a license file line (but not have a checkout depend on the returned information), call lc_auth_data() after a successful checkout. Because lc_auth_data() returns only features which have been successfully checked out, the returned data is authenticated. If you call lc_checkout() with the `LM_CO_LOCALTEST` flag, use the alternate function lc_test_conf() to retrieve the desired license file line instead of lc_auth_data().

SEE ALSO

- `machind/lmflex.c`
- Section 6.4.1, "lc_auth_data()"
- Section 8.2.2, "LM_A_CHECKOUT_DATA"
- Section 6.4.20, "lc_set_attr()"

- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 8.2.4, “LM_A_CHECKOUTFILTERLAST_EX”
- Section 7.3.1, “LM_A_APP_DISABLE_CACHE_READ”
- Section 7.3.6, “LM_A_CKOUT_INSTALL_LIC”
- Section 7.3.10, “LM_A_LICENSE_DEFAULT”
- Section 8.2.9, “LM_A_HOST_OVERRIDE”
- Section 8.2.8, “LM_A_DISPLAY_OVERRIDE”
- Section 8.2.16, “LM_A_USER_OVERRIDE”
- Section 6.4.2, “lc_checkin()”
- Section 3.5, “FEATURE /INCREMENT Lines”
- Section 6.4.21, “lc_status()”
- Section 11.4, “Multiple Jobs”

6.4.4 lc_cryptstr()

SYNTAX

```
status = lc_cryptstr(lmjob, str, return_str, code, flag,
                     filename, errors)
```

DESCRIPTION

Signs the given license certificate. This function is used by license certificate generation utilities such as `lmcrypt` and, as such, is not intended to be incorporated into your application. You pass, via the *`str`* parameter, a license certificate template, which is a complete, valid unsigned (`SIGN=0`) license certificate containing one or more FEATURE, INCREMENT, PACKAGE, or UPGRADE lines. A signed license certificate is returned, via *`return_str`*. For example, consider the following license certificate template as input via *`str`*:

```
FEATURE f1 demo 1.5 01-jan-2005 uncounted HOSTID=08002b32b161 \
        SIGN=0
```

lc_cryptstr() calculates the signature and returns the complete, valid license certificate in *`return_str`*:

```
FEATURE f1 demo 1.5 01-jan-2005 uncounted HOSTID=08002b32b161 \
        SIGN=2C817A5100D8
```

If *flag* has LM_CRYPT_ONLY set, then the function returns the signature for the first FEATURE, INCREMENT, PACKAGE, or UPGRADE line in the certificate. If LM_CRYPT_ONLY is cleared, then the entire certificate is signed and returned as a string. If *flag* has LM_CRYPT_FORCE set, then the signature is recomputed for every line, even if the signature is defined.

Comment lines are retained in the *return_str* output.

If you want to specify a start date, then add the following keyword to the license line template using the following syntax:

```
START=dd-mmm-yyyy
```

Example:

```
START=1-jan-2005
```

The output is compatible with the LM_A_LICENSE_FMT_VER setting, which defaults to the current FLEX*lm* version.

The complete source for the `lmcrypt` command is provided in the Software Development Kit package in `machind/lmcrypt.c`. This as well as the coding example below provide usage information for lc_crypstr().

CODING EXAMPLE

```
[global include and variable info:]
#include "lmprikey.h"
#include "lmclient.h"
#include "lm_code.h"
#include "lmseeds.h"
#include "lm_attr.h"

LM_HANDLE *lm_job;
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
        VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
[...]

[C code:]
char *errors;
char *return_str;
int flag = LM_CRYPT_FORCE;
char *filename = "myfile.lic";
char str[MAX_CONFIG_LINE * 100]; /* if maximum license is 100
                                    lines */
[...]
[set up str variable with valid license syntax]
[...]

/* License Generator initialization */

/* The LM_CODE_GEN_INIT() is called once during license generator
initialization. Do not call this macro more than once during
the execution of the license generator */
```

```
LM_CODE_GEN_INIT(&code);
if (lc_init(0, VENDOR_NAME, &code, &lm_job))
{
    /* present error */
}
/* Initialization complete */
[...]
if (lc_cryptstr(lm_job, str, &return_str, &code, flag,
                filename, &errors))
{
    /* present error, and if non-null, print it out */
}
if (return_str)
{
     /* return_str is the correct license-file string */
}
```

LINKING YOUR LICENSE GENERATOR UTILITY

For Windows platforms:

Once you have successfully compiled your license generator utility, link with the libraries listed in Table 6-10. These libraries are in the *platform* directory of your software development kit. Additionally, refer to the rules for building the `lmcrypt` license generator in *platform*`/makefile`.

Table 6-10: License Generator Utility Libraries

FLEXlm Client Libraries		**Windows System Libraries**
/MT	**/MD**	

`lmgr.lib`
`libcrvs.lib`
`libsb.lib`
`flock.lib`
	`lmgr_md.lib`
	`libcrvs_md.lib`
	`libsb_md.lib`
	`flock_md.lib`
		`oldnames.lib`
		`kernel32.lib`
		`user32.lib`
		`netapi32.lib`
		`advapi32.lib`
		`gdi32.lib`
		`comdlg32.lib`
		`comctl32.lib`
		`wsock32.lib`
		`For /mt:`
		`libcmt.lib`
		`For /md:`
		`msvcrt.lib`

For UNIX platforms:

Once you have successfully compiled your license generator, link with a command similar to the following:

```
cc -o lic_gen lic_gen.o -Lplatform liblmgr.a libcrvs.a
libsb.a $(OSLIBS)
```

with the following notes:

- *lic_gen* is your license generator utility that incorporates lcryptstr().
- $*platform* is platform directory in the production software development kit which contains the necessary FLEX*lm* client libraries.
- `$(OSLIBS)` is a list of platform specific libraries. Refer to the rules for building the `lmcrypt` license generator in *platform*`/makefile` for the list specific to your platform.

PARAMETERS

`(LM_HANDLE *)` *lm_job*	From lc_init().
`(char *)` *str*	Set *str* to a complete valid license file, where the signatures are replaced with `SIGN=0`.
`pointer to (char *)` *return_str*	Resulting license file string. Malloc'd by lc_cryptstr() and freed by the calling program. Pass the address of a `char` pointer.
`pointer to (VENDORCODE)` *code*	From LM_CODE() macro.

(int) *flag*	Mask which can be binary OR'd (|) with the following flags: LM_CRYPT_ONLY—If true, only return signature for first FEATURE in *str*. LM_CRYPT_FORCE—If set, recompute the signature for *every* line, even if the signature is already present on the line. LM_CRYPT_IGNORE_FEATNAME_ERRS—If set, no warnings returned about invalid feature names. LM_CRYPT_DECIMAL—Output will be decimal format. Otherwise, readable format.
(char *) *filename*	For error reporting, or (char *)0. This name will appear in the error message as the file name.
pointer to (char *) *errors*	For error reporting, or (char **)0. If there are errors, the return value is non-zero and errors is set to an explanatory string. Malloc'd by lc_cryptstr(), and freed by the calling program (use lc_free_mem() on Windows). Pass the address of a char pointer. If a warning occurs, *errors* is set to a warning string, but the return value is 0 (success).

RETURN

(int) *status*	0 == success, !0 indicates an error occurred.

ERROR RETURNS

Because different errors can occur on every line of the input *str*, lc_cryptstr() must be able to report all these errors independently, and does so via the *errors* parameter. The *errors* parameter is used for both errors and

warnings. If there is an error, lc_cryptstr() returns non-zero, and no signatures are generated in *return_str*. If there are only warnings, the return value from lc_cryptstr() is success (0), but *errors* is set to a warning message.

Here is an example of error reporting:

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 letters
```

With this error, no signature is generated and *return_str* will be the same as the input *str*.

SEE ALSO

- Section 8.1.3, “lc_check_key()”
- Section 8.1.5, “lc_convert()”
- Section 6.4.11, “lc_get_attr()”
- Section 6.4.15, “lc_init()”
- Section 7.3.11, “LM_A_LICENSE_FMT_VER”
- `machind/lmcrypt.c`
- `machind/makekey.c`

6.4.5 lc_err_info()

SYNTAX

`err_info` = `lc_err_info(`*`job`*`)`

DESCRIPTION

Returns a pointer to a `LM_ERR_INFO` struct, which contains all necessary information to present an error message to the user. This is the supported method for internationalization and localization of FLEX*lm* error messages.

The format of `LM_ERR_INFO` is:

`(int)` *`maj_errno`*	The FLEX*lm* error number. See `lmerrors.h` and `lm_lerrs.h` in the `machind` directory for English versions of the error messages.
`(int)` *`min_errno`*	The minor error number. This allows a support person with access to the FLEX*lm* source code to pinpoint the location where the error occurred.
`(int)` *`sys_errno`*	The most recent system `errno` (or Winsock error on Windows).
`(char *)` *`feature`*	The name of the feature that the error applies to.
`(char **)` *`lic_files`*	A null-terminated array of `char` pointers of the license files used when the error occurred.

(char *) *context*	This is a string which gives additional information about the error. Its contents depends on the type of error, but is not language dependent. Refer to machind/lcontext.h for information needed for translation.

This information allows applications to present error messages in any language and in any desired format. The FLEX*lm* SDK provides the English text of all messages. The three types of messages that need to be translated are

- short — located in machind/lmerrors.h
- long — located in machind/lm_lerr.h
- context — located in machind/lcontext.h

Use the value *err_info.maj_errno* as an index into these files in order to retrieve the corresponding message.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().

RETURN

(LM_ERR_INFO *) *err_info*	Pointer to the LM_ERR_INFO struct, outlined above.

SEE ALSO

- Section 6.4.19, “lc_perror()”

6.4.6 lc_errstring()

SYNTAX

string = lc_errstring(*job*)

DESCRIPTION

Returns the FLEX*lm* error string for the most recent FLEX*lm* error, along with the major and minor error number. If a UNIX error is involved, the UNIX error description will also be included in the message, along with the UNIX `errno`. For internationalization of error messages, use lc_err_info().

This memory is managed by the FLEX*lm* library. Do not attempt to free it. This string is freed and reset when another FLEX*lm* error occurs, so it's only valid between calls to FLEX*lm* functions. Check that the call to the previous FLEX*lm* function has returned an error before calling lc_errstring().

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().

RETURN

(char *) *string*	The FLEX*lm* error string text.

EXAMPLES

```
No such feature exists (-5,116)
Cannot find license file, (-1,73:2), No such file or directory
```

SEE ALSO

- Section 6.4.19, "lc_perror()"
- Section 6.4.5, "lc_err_info()"

6.4.7 lc_expire_days()

days = lc_expire_days(*job*, *conf*)

DESCRIPTION

Returns the number of days until a license expires.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(CONFIG *) *conf*	A FEATURE line from the license file. If security is important and the FEATURE line information should be authenticated, use lc_auth_data() to obtain the CONFIG pointer after a successful checkout. If the FEATURE line information needs to be authenticated before a checkout is done, use lc_checkout() with the LM_CO_LOCALTEST flag, then lc_test_conf() can be used to obtain the CONFIG pointer. If data authentication is not needed, lc_next_conf() can be used to get the CONFIG pointer.

RETURN

(int) *days*	LM_FOREVER — Unexpiring license. > 0 — Number of days until expiration. ==0 — The license will expire tonight at midnight. < 0 — FLEX*lm* errno.

ERROR RETURNS

LM_BADPARAM	*conf* is 0.
LM_LONGGONE	The feature has already expired.

SEE ALSO

- Section 6.4.3, “lc_checkout()”
- Section 8.1.10, “lc_next_conf()”

6.4.8 lc_feat_list()

SYNTAX

```
list = lc_feat_list(job, flags, dupaction)
```

DESCRIPTION

Gets the list of all features in the license file.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(int) flags`	LM_FLIST_ALL_FILES for all license files. If 0, only the first license in the license-file list is used.
`(void) (*dupaction)()`	Action routine called when a duplicate feature is found. This routine is called upon the second occurrence of any feature name. If specified as NULL, no call is made.

RETURN

`(char **) list`	List of features. *list* is a pointer to a NULL-terminated array of feature string pointers. Both the pointers and the string data are malloc’d; this memory is freed upon a subsequent call to lc_feat_list(). Do not free this data. If NULL, an error has occurred.

ERROR RETURNS

LM_CANTMALLOC	The call to malloc() failed.
LM_NOFEATURE	Specified feature not found.

6.4.9 lc_first_job()

SYNTAX

```
job = lc_first_job(job)
```

DESCRIPTION

lc_first_job() is used to find the first job in a list of jobs.

CODING EXAMPLE

```
LM_HANDLE *job
job = lc_first_job(job);
while (job)
{
      /*processing*/
      job = lc_next_job(job);
}
```

PARAMETER

(LM_HANDLE *) *job*	A pointer to the first job is returned into this parameter.

ERROR RETURNS

None.

SEE ALSO

- Section 6.4.10, “lc_free_job()”
- Section 6.4.17, “lc_new_job()”
- Section 6.4.18, “lc_next_job()”
- Section 11.4, “Multiple Jobs”

6.4.10 lc_free_job()

SYNTAX

```
(void) lc_free_job(job)
```

DESCRIPTION

lc_free_job() frees the memory associated with a job, which has been allocated by lc_new_job(). Calls to this function are needed only by an application that uses a large number of jobs over its lifetime.

lc_free_job() removes everything associated with *job*; this includes any timers, memory and licenses.

For dynamically linked Windows applications or FLEXenabled DLLs, it is required to call lc_cleanup() after the last call to lc_free_job().

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().

RETURN

None.

ERROR RETURNS

LM_BADPARAM	No such job.

SEE ALSO

- Section 6.4.15, “lc_init()”
- Section 6.4.17, “lc_new_job()”
- Section 6.4.20, “lc_set_attr()”
- Section 8.1.4, “lc_cleanup() (Windows only)”
- Section 11.4, “Multiple Jobs”

6.4.11 lc_get_attr()

SYNTAX

```
#include "lm_attr.h"
status = lc_get_attr(job, attr, value)
```

DESCRIPTION

Retrieves a FLEX*lm* attribute. *attr* describes which attribute to retrieve, and the value is a pointer to the value for the attribute. See lm_attr.h for *attr* constants and value types. Attribute descriptions can also be found in Section 7.3, “FLEXible API Attributes set by lc_set_attr(),” and Section 8.2, “Advanced FLEXible API Attributes.”

Types of char * are handled a little differently than other types. Types of int or short are declared, and a pointer to the declared variable is passed as an argument. Types of char * are declared as char *, and the variable itself is passed.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(int) *attr*	Which attribute to get.

RETURN

(short *) *value*	Value of the attribute. *value* must be a pointer to the correct attribute type and should be cast to a short *. Return value is set in *value*.
(int) *status*	0—OK, <>0, error.

ERROR RETURNS

LM_NOSUCHATTR	No such attribute exists.

LM_NOADMINAPI	LM_A_VD_GENERIC_INFO or LM_A_VD_FEATURE_INFO only—request was made to other company's vendor daemon.
LM_NOSERVSUPP	LM_A_VD_GENERIC_INFO or LM_A_VD_FEATURE_INFO only—pre-v4.0 server does not support these requests.

SEE ALSO

- Section 6.4.20, "lc_set_attr()"
- Section 7.3, "FLEXible API Attributes set by lc_set_attr()"

6.4.12 lc_heartbeat()

SYNTAX

```
status = lc_heartbeat(job, num_reconnects, num_minutes)
```

DESCRIPTION

lc_heartbeat() sends heartbeat messages to and receives acknowledgments from the license server. By default, these activities are handled automatically by FLEX*lm* via a separate, dedicated application thread. This function provides manual control of heartbeat messages; thereby, overriding the automatic mechanism.

The purpose of heartbeat messages is to:

- Keep license server informed that the application is still using its license — otherwise the license server may timeout and check in the license.
- Keep the application informed that the license server is continually running — unchecked license server stops and starts may indicate unauthorized license usage.

lc_heartbeat() performs the following activities:

- It attempts to read the acknowledgement from the previous heartbeat.
- It sends out a heartbeat message or, if there is no acknowledgement from the previous one, it makes one attempt to reconnect to the license server.
- If reconnection is successful, re-checks out the original complement of licenses from the license server.

- It informs the application of a number of states that may indicate attempted tampering with the license server.

To use lc_heartbeat(), you must first turn off the automatic mechanism before the first license checkout:

```
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);
```

Heartbeat messages ensure that licenses are re-checked out from a restarted server. Heartbeats are not needed for the server to retain a client's license—the server retains the license until the client exits. If lc_heartbeat() is called, the client will automatically reconnect and re-checkout from a license server that has restarted. It also informs the application of a number of states that may indicate attempted tampering with the license server.

- The return value, if non-zero, indicates that the license server is down, and how many reconnect attempts have been made. This information can be used, for example, to inform the end user the license server is down and possibly to deny use after a specified number of failures.
- The arguments *num_reconnects* and *num_minutes*, if used, can indicate the rate a server has been stopped and started, possibly signifying attempted theft.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(int *) *num_reconnects*	Pointer to int. If null, this argument is ignored. If non-null, and the client has just successfully reconnected to the server, the return value will be 0 (success), and *num_reconnects* is set to the number of times the client has reconnected in the last minutes. If this is a large number, it may indicate attempted theft.

(int) *num_minutes*	If 0, this argument is ignored. If non-zero, it's used to detect when a server is being started and stopped many times in a short period, which can indicate attempted theft. The reporting period is set with *num_minutes*.

RETURN

(int) *status*	If non-zero, the license server is currently down, and is the number of failed attempts to reconnect.

HOW LC_HEARTBEAT() WORKS

lc_heartbeat() sends a heartbeat message to the license server. It then reads the acknowledgement from the previously sent heartbeat. The first heartbeat is sent after the application first connects to the license server via lc_checkout(). In this manner, there is normally no delay in lc_heartbeat().

If lc_heartbeat() is unable to read an acknowledgement from the license server, it makes one attempt to reconnect to the license server according to the following parameters:

- Each subsequent call to lc_heartbeat() makes one reconnection attempt for a total of five attempts (default). The LM_A_RETRY_COUNT attribute is used to override the total number of attempts.
- Calls to lc_heartbeat() more frequent than 30 seconds are ignored. That is, a reconnection is attempted at most every 30 seconds even when lc_heartbeat() is called more frequently.
- The application can optionally define a pre-reconnection callback function via the LM_A_USER_RECONNECT attribute.
- If a reconnection occurs before five attempts and the application defines a post-reconnection callback function via the LM_A_USER_RECONNECT_DONE attribute, it is called.
- If a reconnection fails to occur after five attempts, the a callback exit handler, if specified with the LM_A_USER_EXITCALL attribute, is called. Otherwise, the application exits with the error message, "Lost License, cannot re-connect" (UNIX — stderr, WINDOWS—dialog box).

LC_HEARTBEAT() AND USER TIMEOUT OPTION

If lc_heartbeat() is not called for an extended period, then the application may lose its license. This can happen for two reasons: the timeout, set with the LM_A_TCP_TIMEOUT attribute, has expired or the end user has set a TIMEOUT for this feature in the end-user options file. In both cases, the license server has a timeout associated with the license which gets invoked if lc_heartbeat() is not called within the timeout interval. Make sure that LM_A_TCP_TIMEOUT is large enough to accommodate your usage of lc_heartbeat(). Similarly, make sure `ls_minimum_user_timeout` in `lsvendor.c` is large enough so that users will not timeout applications that are in use.

If the license is inadvertently released, the next lc_heartbeat() will automatically re-acquire the license, if there is still a license available.

SEE ALSO

- Chapter 12, “Heartbeats”
- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 7.3.16, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL”
- Section 7.3.17, “LM_A_TCP_TIMEOUT”
- Section 7.3.18, “LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX”
- Section 7.3.19, “LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX”
- Section 7.3.20, “LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX”
- Section 13.2.9, “ls_minimum_user_timeout”
- TIMEOUT options file keyword in the *FLEXlm End Users Guide*

6.4.13 lc_hostid()

SYNTAX

```
char buf[MAX_CONFIG_LINE];
status = lc_hostid(job, id_type, buf);
```

DESCRIPTION

Fills in *buf* with a hostid string specified by *id_type*. If *id_type* is HOSTID_DEFAULT, you get the default *id_type* on the system.

This function allows developers access to hostid information in string format. Note that lc_hostid() may return a space-separated list of hostids, if appropriate.

If a request is made for an invalid hostid type for a platform, lc_hostid() returns the value of the default hostid type for that platform. If a request is made for a valid type that is not currently present on the machine, such as HOSTID_FLEXID6 on Windows, lc_hostid() returns an error. The *FLEXlm End Users Guide* contains a list of currently supported platforms and their default hostid type.

PARAMETERS

`(LM_HANDLE *)` *`job`*	From lc_new_job().
`(int)` *`id_type`*	The requested hostid type. See *`id_types`* listed in the table below. For backward compatibility, other choices exist in `lmclient.h`, but are not applicable in this version of FLEX*lm*.
`(char *)`*`buf`*	A pointer to a `char` array of length MAX_CONFIG_LINE. If successful, the hostid string is returned here.

HOSTID TYPES

There are two groups of hostid types: those valid for a specific platform and those valid on all platforms. Table 6-11 lists the valid hostid types for a specific platform.

Table 6-11:Hostid Types

Operating System	FLEX*lm* Platform	Supported Hostid Types (*`id_type`*)	Description
AIX	All	HOSTID_LONG	32-bit hostid
Alpha OSF	All	HOSTID_ETHER	Ethernet address
FreeBSD	All	HOSTID_ETHER	Ethernet address

Table 6-11:Hostid Types (Continued)

Operating System	**FLEX*lm* Platform**	**Supported Hostid Types (`id_type`)**	**Description**
HP-UX	Non-Itanium	HOSTID_LONG	32-bit hostid
	Itanium	HOSTID_ID_STRING	String ID, maximum length is MAX HOSTID_LEN
IRIX	All	HOSTID_LONG	32-bit hostid
Mac OS	All	HOSTID_ETHER	Ethernet address
		HOSTID_FLEXID9	FLEXID=9 • Mfg: Aladdin Knowledge Systems • Device: HASP® 4 M1 USB memory key

Table 6-11:Hostid Types (Continued)

Operating System	FLEX*lm* Platform	Supported Hostid Types (`id_type`)	Description
Microsoft Windows	All	HOSTID_DISK_ SERIAL_NUM	Windows disk serial number
		HOSTID_ETHER	Ethernet address
	32-bit, only: 98/Me/NT/ 2000/XP/ Server 2003	HOSTID_FLEXID6	FLEXID=6. (Replaces FLEXID=7) • Mfg: Rainbow Technologies/SafeNet, Inc. • Device: Parallel-based iKey™ authentication token (No USB support on NT)
		HOSTID_FLEXID7	FLEXID=7 (No longer available, replaced by FLEXID=6) • Mfg: Rainbow Technologies/SafeNet, Inc. • Device: Parallel-based iKey™ authentication token
		HOSTID_FLEXID8	FLEXID=8 • Mfg: Dallas Semiconductor/Maxim Integrated Products, Inc. • Device: 1-Wire® parallel port adapter
		HOSTID_FLEXID9	FLEXID=9 • Mfg: Aladdin Knowledge Systems • Device: HASP® 4 M1 USB memory key (except NT)

Table 6-11:Hostid Types (Continued)

Operating System	**FLEX*lm* Platform**	**Supported Hostid Types (*id_type*)**	**Description**
Red Hat Linux	All	HOSTID_ETHER	Ethernet address
	Version 7 and greater, 32-bit only	HOSTID_FLEXID9	FLEXID=9 • Mfg: Aladdin Knowledge Systems • Device: HASP® 4 M1 USB memory key
SCO	All	HOSTID_ID_STRING	String ID, maximum length is MAX HOSTID_LEN
Solaris	All	HOSTID_LONG	32-bit hostid
		HOSTID_ETHER	Ethernet address
SuSE Linux	All	HOSTID_ETHER	Ethernet address
	Enterprise Server 8.1, 32-bit only	HOSTID_FLEXID9	FLEXID=9 • Mfg: Aladdin Knowledge Systems • Device: HASP® 4 M1 USB memory key

The following table lists the hostid types common to all platforms.

id_type	**Description**
HOSTID_COMPOSITE	Composite hostid, if defined. See Chapter 14, “Composite Hostids,” and Section 8.1.9, “lc_init_simple_composite().”
HOSTID_DEFAULT	Default hostid type on the system.
HOSTID_DISPLAY	Display name.
HOSTID_HOSTNAME	Node name.
HOSTID_INTERNET	Internet IP address.
HOSTID_USER	User name.

id_type	**Description**
HOSTID_VENDOR	Vendor-defined hostid, if defined. See Chapter 15, “Vendor-Defined Hostid Types.”

RETURN

(int) *status*	0 if successful, FLEX*lm* errno otherwise.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this *id_type*.

SEE ALSO

- machind/lmclient.h for valid hostid type definitions
- *FLEXlm End Users Guide*, Appendix A, “Hostids for FLEXlm-Supported Machines,” for a list of the default hostid for each supported platform

6.4.14 lc_idle()

SYNTAX

```
(void) lc_idle(job, flag)
```

DESCRIPTION

Informs FLEX*lm* when the process is idle. lc_idle() enables the end user feature inactivity TIMEOUT to allow idle licenses to be reclaimed. Use of lc_idle() is recommended for end users to take advantage of the TIMEOUT option. lc_idle() also affects vendor daemon timeout due to LM_A_TCP_TIMEOUT.

lc_idle() can be used to bracket a portion of the application code that prompts for user input, so that when the user is not using the application, the vendor daemon can detect the fact that the application is idle. lc_idle() only sets a flag internally in the application; it is therefore safe to call as often as necessary.

A typical use would be:

```
lc_idle(job, 1);       /* Process is idle now */
... wait for input from user...
lc_idle(job, 0);       /* Process is no longer idle */
```

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(int) flag`	0 if process is not idle, non-zero if process is idle.

RETURN

None.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”
- Section 7.3.17, “LM_A_TCP_TIMEOUT”
- Section 13.2.9, “ls_minimum_user_timeout”

6.4.15 lc_init()

SYNTAX

```
#include "lm_code.h"
status = lc_init(job, VENDOR_NAME, &code, &lm_job)
```

DESCRIPTION

lc_init() initializes FLEX*lm* and creates a license job. Subsequent calls to lc_init() create new license jobs; each job is independent. lc_init() is only to be used with license generators and not in licensed applications shipped to end users. Licensed applications use lc_new_job(), instead, for initialization and job creation because it offers enhanced security.

PARAMETERS

`(LM_HANDLE *)job`	Must be NULL on first call to lc_init(). Otherwise, the current `job` on all subsequent calls.
`(char *)vendor_id`	Vendor name as defined with `VENDOR_NAME` in `machind/lm_code.h`.
`pointer to (VENDORCODE *) code`	From LM_CODE() macro.

RETURN

`pointer to (LM_HANDLE *)` *`lm_job`*	Set to job for the current process. This is used as the first argument to all subsequent lc_*xxx*() functions.
`(int)` *`status`*	Value of lc_get_errno() after initialization is complete, 0 if successful.

ERROR RETURNS

LM_BAD_TZ	Time zone offset from GMT is > 24 hours (may imply a user is attempting to bypass an expiration date).
LM_BADPLATFORM	Vendor keys do not support this platform.
LM_BADKEYDATA	Bad vendor keys.
LM_BADVENDORDATA	Unknown vendor key type.
LM_CANTMALLOC	The call to malloc() failed.
LM_DEFAULT_SEEDS	Encryption seeds were left to default values, but the vendor daemon name is not `demo`.
LM_EXPIRED_KEYS	Vendor keys have expired.
LM_NOKEYDATA	Vendor key data not supplied.
LM_LIBRARYMISMATCH	`lmclient.h/liblmgr.a` version mismatch.
LM_NONETWORK	Networking software not available on this machine.
LM_OLDVENDORDATA	Old vendor keys supplied.

SEE ALSO

- Section 6.4.4, “lc_cryptstr()”
- Section 6.4.17, “lc_new_job()”

6.4.16 lc_log()

SYNTAX

```
(void) lc_log(job, msg)
```

DESCRIPTION

Logs a message in the debug log file, if the license is served by `lmgrd`.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) msg`	The message to be logged. The maximum length of *msg* is LM_LOG_MAX_LEN.

RETURN

None.

ERROR RETURNS

LM_NOSOCKET	Communications failure to daemon.
LM_CANTWRITE	Write error sending message to daemon.

6.4.17 lc_new_job()

SYNTAX

```
VENDORCODE code;
LM_HANDLE *job = (LM_HANDLE *)NULL;
status = lc_new_job(prevjob, lc_new_job_arg2, &code, &job);
```

DESCRIPTION

lc_new_job() initializes FLEX*lm* and creates a license job. Subsequent calls to lc_new_job() create new license jobs. Each license job is independent.

lc_new_job() is used with licensed applications. For license generators (like `lmcrypt` and `makekey`) use lc_init() instead.

All applications must link `lm_new.o` (`lm_new.obj` on Windows) into the application executable. Failing to link `lm_new.o` into the executable will result in unresolved references to `l_n36_buf`.

Note: The application must call lc_new_job() before calling any other FLEX*lm* function, including lc_set_attr() and lc_get_attr().

PARAMETERS

`(LM_HANDLE *) prevjob`	Must be NULL on first call to lc_new_job(). On subsequent calls, use any existing job previously initialized with lc_new_job().
`lc_new_job_arg2`	This second parameter is required for enhanced security for a DLL.

RETURN

`pointer to (VENDORCODE) code`	Pointer to `VENDORCODE` struct. Initialized by this function and used later as argument to lc_checkout().
`pointer to (LM_HANDLE *) job`	Set to job for the current process. This is used as the first argument to all subsequent lc_*xxx*() functions.
`(int) status`	Value of lc_get_errno() after initialization is complete, 0 if successful.

ERROR RETURNS

LM_BAD_TZ	Time zone offset from GMT is > 24 hours (may imply a user is attempting to bypass an expiration date).
LM_BADPLATFORM	Vendor keys do not support this platform.
LM_BADKEYDATA	Bad vendor keys.
LM_BADVENDORDATA	Unknown vendor key type.
LM_CANTMALLOC	The call to malloc() failed.

LM_DEFAULT_SEEDS	Encryption seeds were left to default values, but the vendor daemon name is not `demo`.
LM_EXPIRED_KEYS	Vendor keys have expired.
LM_NOKEYDATA	Vendor key data not supplied.
LM_LIBRARYMISMATCH	`lmclient.h/liblmgr.a` version mismatch.
LM_NONETWORK	Networking software not available on this machine.
LM_OLDVENDORDATA	Old vendor keys supplied.

SEE ALSO

- Section 6.4.15, “lc_init()”
- Section 6.4.10, “lc_free_job()”
- Section 11.4, “Multiple Jobs”

6.4.18 lc_next_job()

SYNTAX

```
LM_HANDLE *job
job = lc_first_job(job);
while (job)
{
       /*processing*/
       job = lc_next_job(job);
}
```

DESCRIPTION

lc_next_job() is used to walk the list of jobs. This only works properly if all calls to lc_new_job() have a pointer to the current job as the first parameter.

PARAMETERS

`(LM_HANDLE *)` *`job`*	Current job.

RETURN

`(LM_HANDLE *) job`	Next currently active job, or `(LM_HANDLE *)0` if end.

ERROR RETURNS

None.

SEE ALSO

- Section 6.4.9, “lc_first_job()”
- Section 6.4.10, “lc_free_job()”
- Section 6.4.17, “lc_new_job()”
- Section 11.4, “Multiple Jobs”

6.4.19 lc_perror()

SYNTAX

```
(void) lc_perror(job, string)
```

DESCRIPTION

Prints a FLEX*lm* error message, in the same format as the UNIX function perror(), e.g.:

```
"string": FLEXlm error-string
```

If a system error has also occurred, it will be included in the message.

On Windows systems, a dialog of type MB_OK will be displayed with the FLEX*lm* error message. The FLEX*lm* error messages are available by calling lc_errstring().

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) string`	The first part of the error message, as above.

RETURN

None.

SEE ALSO

- Section 6.4.5, “lc_err_info()”

- Section 6.4.6, “lc_errstring()”
- Section 7.3.13, “LM_A_LONG_ERRMSG”

6.4.20 lc_set_attr()

SYNTAX

```
#include "lm_attr.h"
status = lc_set_attr(job, attr, (LM_A_VAL_TYPE)value)
```

DESCRIPTION

Sets a FLEX*lm* attribute. *attr* describes which attribute to set, and the value is the value for the attribute. See the header file `lm_attr.h` for *attr* constants and value types.

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().
`(int)` *attr*	Which attribute to set.
`(LM_A_VAL_TYPE)` *value*	Value to set it to. Values should be of the appropriate type for the particular attribute (see `lm_attr.h`), but should be cast to `LM_A_VAL_TYPE`.

RETURN

`(int)` *status*	0—OK, !=0, error.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_BADPARAM	Specified parameter is incorrect.
LM_NOCONFFILE	Specified license file cannot be found (LM_A_LICENSE_FILE or LM_A_LICENSE_FILE_PTR).

LM_NOSUCHATTR	Specified attribute does not exist.

SEE ALSO

- Section 7.3, “FLEXible API Attributes set by lc_set_attr()”

6.4.21 lc_status()

SYNTAX

status = lc_status(*job*, *feature*)

DESCRIPTION

Returns the status of the requested feature.

A call to this function is made when QUEUEing for a license. Normally QUEUEing is done in the following manner:

```
status = lc_checkout(....LM_CO_NOWAIT,...);
if (status == LM_MAXUSERS || status == LM_USERSQUEUED)
{
    printf("Waiting for license...");
    status = lc_checkout(....LM_CO_WAIT,...);
}
```

However, in the above example, the application must pend on the call to lc_checkout(). If the application needs to continue doing processing, use LM_CO_QUEUE in the call to lc_checkout(). Call lc_status() immediately after the call to lc_checkout() and any other lc_*xxx*() function until the license is granted or denied. This might be coded in the following manner:

```
status = lc_checkout(...,LM_CO_QUEUE,...)
switch (status)
{
       case 0:
              break; /* got the license */
       case LM_MAXUSERS:
       case LM_USERSQUEUED:
       case LM_FEATQUEUE:
              printf("Waiting for license...");
              while (lc_status(job, feature))
              {
                     /* processing */
              }
              break; /* got the license */
       default:
              lc_perror(job, "Checkout for license failed");
}
```

PARAMETERS

`(LM_HANDLE *)` *`job`*	From lc_new_job().
`(char *)` *`feature`*	The feature name.

RETURN

`(int)` *`status`*	Status of this feature (in this process): < 0 — error; 0 — feature is checked out by this process.

ERROR RETURNS

LM_CANTCONNECT	Feature was checked out, but lost connection to the daemon.
LM_FEATQUEUE	This process is in the queue for this feature.
LM_NEVERCHECKOUT	Feature was never checked out by this process, or was checked back in after a checkout.

SEE ALSO

- Section 6.4.3, “lc_checkout()”

6.4.22 lc_userlist()

SYNTAX

```
LM_USERS *users;
users = lc_userlist(job, feature)
```

DESCRIPTION

Provides a list of who is using the feature, including information about the users of the license. This output is used by `lmstat`. See the *FLEXlm End Users Guide* for the behavior of `lmstat`.

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().
`(char *)` *feature*	The feature name.

Note: The call to lc_userlist() is potentially expensive (it may cause a lot of network traffic), depending on the number of users of *feature*. Therefore this function must be used with caution. In particular, it is a good idea to call lc_userlist() when a checkout fails with LM_MAXUSERS/LM_USERSQUEUED error, to inform who is using the feature. However, do *not* call lc_userlist() before every call to lc_checkout(), because this will be guaranteed to cause network load problems when a large number of licenses are checked out.

RETURN

If successful, lc_userlist() returns a pointer to a linked list of structures, one for each user of the license. This data should not be modified by the caller. It will be freed on the next call to lc_userlist().

See `lmclient.h` for a description of the `LM_USERS` struct.

The list of users returned by lc_userlist() includes a special record, indicated by an empty user name (`name[0]==0`), which contains the total number of licenses supported by the daemon for the specified feature (in the `nlic` field), and the daemon's idea of the current time (in the `time` field).

If there is an error, lc_userlist() returns NULL and sets the job error status.

lc_userlist() returns only information about users the server knows about, therefore it will not return any information about users of node-locked uncounted or DEMO licenses, unless the server's license file includes the

node-locked licenses and the client is not reading the license file (via `@host`, `port@host` or USE_SERVER). Queued users and licenses shared due to duplicate grouping are also not returned by lc_userlist().

Reserved licenses are indicated by the lm_isres() macro (defined in `lmclient.h`). In this case, the `name` contains the entity that the reservation is for.

ERROR RETURNS

LM_BADCOMM	Communications error with license server.
LM_CANTMALLOC	The call to malloc() failed.
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOFEATURE	Specified feature cannot be found.

SEE ALSO

- `machind/lmclient.h` for `LM_USERS` structure definition.

6.4.23 lc_vsend()

SYNTAX

```
rcv_str = lc_vsend(job, send_str)
```

DESCRIPTION

Sends a message to the vendor daemon and returns a result string. If the client is not already connected to a server, this function will connect to the first server in the first license file in its list. The string can be up to 140 bytes.

You must set up a processing routine in your vendor daemon to receive the message from lc_vsend() and send the reply. This routine is specified in `lsvendor.c` in the variable `ls_vendor_msg`.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) send_str`	String to be send to your vendor daemon.

RETURN

`(char *) rcv_str`	String returned by ls_vendor_msg() in your vendor daemon; 0 if unsuccessful.

ERROR RETURNS

LM_BADCOMM	Communications problem with the vendor daemon.
LM_CANTREAD	Cannot read data from license server.
LM_NOSERVSUPP	Your vendor daemon does not support this function.

SEE ALSO

- Section 13.2.15, “ls_vendor_msg”

Chapter 7

Controlling Licensing Behavior

Licensing behavior is controlled from within the FLEX*lm* licensed application by setting license *policies* and *attributes*. The Trivial and Simple APIs provide a set of license policies and policy modifiers that are specified in the call to the checkout function. The FLEXible API allows you to control the licensing behavior of your application by setting attributes with the lc_set_attr() function.

The information in this chapter is divided into the following sections:

- Trivial and Simple API License Policies
- Trivial and Simple API Policy Modifiers
- FLEXible API Attributes set by lc_set_attr()

7.1 Trivial and Simple API License Policies

The Trivial and Simple APIs both require that you specify a license policy. A policy can be modified by ORing it with a list of optional modifiers (see Section 7.2, “Trivial and Simple API Policy Modifiers.” The following license policies are available:

- LM_RESTRICTIVE
- LM_QUEUE
- LM_FAILSAFE
- LM_LENIENT

7.1.1 LM_RESTRICTIVE

With this policy, any failure in the license, checkout, or server will be reported to the calling application as an error. The application decides what action to take with this error—it is not necessary that the application fail to run. For example, the application may report the error and continue running, it may exit, or it may run in a limited mode.

7.1.2 LM_QUEUE

This policy is the same as LM_RESTRICTIVE, except that the checkout call will wait for a license if the licenses are all currently in use. To the end user, the application will appear to "hang" until the license is available.

7.1.3 LM_FAILSAFE

With this policy, the application will attempt a checkout, but no checkout failures of any kind will be reported to the calling application. This policy provides "optional" licensing to the user. If the user wants to use licensing, he can, in which case the checkout will succeed. If the user doesn't want to use licensing, or if licensing is for some reason broken, applications will always continue to run.

In the case where all licenses are currently in use, the application will still run. The end user could use SAM*report* to report on historical usage, which will show when licensed use is exceeded. Application users will never be denied usage. Errors that normally make a checkout fail are available as warnings.

7.1.4 LM_LENIENT

In this policy, if all licenses are in use, the checkout will return a failure status showing that all licenses are in use. For any other error, no error is returned. This is another form of "optional" end user licensing, where the user is not penalized if licensing is not set up or if an operational error occurs. Errors that would normally make a checkout fail are available as warnings.

7.2 Trivial and Simple API Policy Modifiers

These modifiers are binary ORed ("|") with the main policies, described in Section 7.1, "Trivial and Simple API License Policies." The following policy modifiers are available:

- LM_MANUAL_HEARTBEAT
- LM_RETRY_RESTRICTIVE
- LM_CHECK_BADDATE
- LM_FLEXLOCK

7.2.1 LM_MANUAL_HEARTBEAT

If this policy modifier is not specified, heartbeats, via lt_heartbeat(), HEARTBEAT(), or lp_heartbeat() are automatically sent every 120 seconds from the application to the server. Default automatic heartbeats are recommended.

If you want to disable automatic heartbeats and call lt_heartbeat(), HEARTBEAT() or lp_heartbeat() directly, use, for example:

```
LM_RESTRICTIVE | LM_MANUAL_HEARTBEAT
```

This indicates that the main policy is LM_RESTRICTIVE, that automatic heartbeats to the license server are disabled, and that the application will call lt_heartbeat(), HEARTBEAT(), or lp_heartbeat() directly.

SEE ALSO

- Section 12.1, “Automatic Heartbeats”

7.2.2 LM_RETRY_RESTRICTIVE

If this policy modifier is set, the application will exit with a short error message after five failed heartbeat messages. This is not normally recommended, but is useful for some simple applications.

7.2.3 LM_CHECK_BADDATE

If set, attempts are made to detect whether the user has set the system date back. This should be used in conjunction with setting `ls_a_check_baddate` to 1 in the `machind/lsvendor.c` file.

SEE ALSO

- Section 9.1.2, “Limited Functionality Demos”
- Section 13.2.2, “ls_a_check_baddate”

7.2.4 LM_FLEXLOCK

If set, FLEX*lock* functionality is enabled.

SEE ALSO

- Information on FLEX*lock* in the *FLEXlm Programmers Guide*.

7.3 FLEXible API Attributes set by lc_set_attr()

FLEXible API attributes allow you control over licensing policy, internal operations of FLEX*lm* (e.g., automatic heartbeat mechanism, etc), and control of the licensing parameters of your process (e.g., define how FLEX*lm* will define “username,” “hostname,” and “display name,” etc. for managed license distribution).

To set FLEXible API attributes, call the lc_set_attr() function, described in Section 6.4.20, “lc_set_attr().”

The essential FLEXible API attribute which should be set by every FLEXible API licensed application is LM_A_LICENSE_DEFAULT. This attribute defines the default license file location.

The following attributes are often useful:

- Vendor-defined Hostid:
  LM_A_VENDOR_ID_DECLARE
- Customized checkout:
  LM_A_BORROW_EXPIRE
- Information useful for error, or informational, reporting:
  LM_A_BORROW_STAT
  LM_A_LF_LIST
  LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO

The attributes are described in the sections the follow; with one attribute per section. The first line of each section is the data type of the attribute. All attribute definitions are in `lm_attr.h`. The attributes are changed with lc_set_attr() and are queried with lc_get_attr().

When using these attributes with lc_set_attr(), the argument must be of the correct type (each attribute below lists its associated type) and must then be cast to `LM_A_VAL_TYPE`. When using them with lc_get_attr(), the pointer argument should point to a value of the correct type (noting that `short` and `int` are different in this case), and must be cast to a `short *`.

SEE ALSO

- Section 8.2, “Advanced FLEXible API Attributes”
- Section D.2, “Obsolete FLEXible API Attributes”

7.3.1 LM_A_APP_DISABLE_CACHE_READ

Type: `(int)`

Default: `0`

By default, after a successful checkout, the license file location used for the feature is cached in the *`VENDOR`*`_LICENSE_FILE` variable in the registry (Windows) or `$HOME/.flexlmrc` (UNIX). All subsequent checkouts for features from this vendor will read the cache, first, to determine the license file location.

To disable reading the cache, set this attribute to `1`. Set this attribute immediately after the call to lc_new_job() and before any other code in the application.

SEE ALSO

- "Registry and $HOME/.flexlmrc"
- Section 7.3.6, "LM_A_CKOUT_INSTALL_LIC"

7.3.2 LM_A_BORROW_EXPIRE

Type: `(char *)`

Default: None

Date and optional time when a borrowed license expires.

If you want your end users to request license borrowing from within your application, issue a license with the BORROW keyword on a feature line and write an interface in which the user specifies the end date (and optionally time) when the borrowed license will be returned. Before the call to lc_checkout(), call:

```
lc_set_attr(job, LM_A_BORROW_EXPIRE, (LM_A_VAL_TYPE)datestring);
```

where *`datestring`* is the date (and optionally time) when the borrowed license expires (provided by the user through the interface). The date in *`datestring`* is in *`dd-mmm-yyyy`*`[:`*`hh`*`:`*`mm`*`]` format and the time is according to a 24-hour clock. For example:

```
13-dec-2005:15:00
```

All subsequent license checkouts in this job will attempt to borrow licenses.

SEE ALSO

- Section 3.5.6, "BORROW"
- `lmborrow` command in *FLEXlm Programmers Guide* or *FLEXlm End Users Guide*

7.3.3 LM_A_BORROW_STAT

Type: `(LM_BORROW_STAT *)`

Default: None

Use this attribute to retrieve information about features borrowed on the client machine, that is, the machine where `lmborrow` was run. This is useful for finding out which features have been borrowed while disconnected from the network. After the call to lc_checkout() that activates borrowing, call:

```
LM_BORROW_STAT *s;
lc_get_attr(job, LM_A_BORROW_STAT, &s);
for (; s; s = s->next)
{
    /* s contains local borrow status */
}
```

The same information is provided by running the FLEX*lm* utility `lmborrow -status`.

SEE ALSO

- `lmborrow` command in *FLEXlm Programmers Guide* or *FLEXlm End Users Guide*

7.3.4 LM_A_CHECK_BADDATE

Type: `(int)`

Default: False

If True, and the license that authorizes the application has an expiration date, a check is made to see if the system date has been set back on the client node. If the checkout fails for this reason, the checkout error is LM_BADSYSDATE.

SEE ALSO

- Section 9.1.2, “Limited Functionality Demos”
- Section 13.2.2, “ls_a_check_baddate”

7.3.5 LM_A_CHECK_INTERVAL

Type: `(int)`

Default: 120 second interval

LM_A_CHECK_INTERVAL controls the rate at which the licensed application sends automatic heartbeat messages to the license server. To disable automatic heartbeat messages, set LM_A_CHECK_INTERVAL to `-1`, (as well as setting LM_A_RETRY_INTERVAL to `-1)`. The minimum value for LM_A_CHECK_INTERVAL is 30 seconds.

The results of possible settings of this variable are:

Variable:	Setting:	Result:
`check_interval`	-1	Automatic heartbeats are disabled.

check_interval	>= 0, < 30	Value is ignored and current interval is unchanged.
check_interval	>= 30	Timer interval.

If automatic heartbeats are disabled, you must call lc_heartbeat() periodically to check the status of the license server.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”
- Section 7.3.16, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL”
- “Heartbeats” chapter in the *FLEXlm Programmers Guide*

7.3.6 LM_A_CKOUT_INSTALL_LIC

Type: `(int)`

By default, a successful checkout automatically updates the registry *VENDOR*`_LICENSE_FILE` setting (where *VENDOR* is your vendor name) to include the license file location that was used for the checkout. This can be disabled by setting this attribute to 0.

Default: None

SEE ALSO

- “Registry and $HOME/.flexlmrc”
- Section 7.3.1, “LM_A_APP_DISABLE_CACHE_READ”

7.3.7 LM_A_FLEXLOCK

Type: `(int)`

Default: Off

Turns on FLEX*lock* capability. This must be enabled to use FLEX*lock*, but application security is poorer. FLEX*lock* is available only on Windows.

See the *FLEXlm Programmers Guide* and Section 11.5, “FLEXlock,” in this manual for additional information on FLEX*lock*.

7.3.8 LM_A_FLEXLOCK_INSTALL_ID

Type: `(short *)`

Default: Unused

For additional security, each time that your application is installed, and the user activates the FLEX*lock* operation, a random id number is generated. This number can be used to identify work done with your application in this mode. If this number is saved in the work and compared when accessing it, you may be able to determine if your application has been re-installed. FLEX*lock* is available only on Windows.

You can obtain this number by calling:

```
long code_id;
lc_get_attr(job, LM_A_FLEXLOCK_INSTALL_ID, (short *)&code_id);
```

After the FLEX*lock* operation is activated, an entry is generated in the registry. It is located at:

```
HKEY_CURRENT_USER\Software\Macrovision Corporation\FLEXlock
```

A subkey for each feature is located inside the FLEX*lock* subkey and is a combination of the vendor name and the feature name. If this subkey is deleted, the program will act as if you had never activated the FLEX*lock* functionality. (Familiarity with the registry editor is necessary for testing FLEX*lock*-enabled features.)

See the *FLEXlm Programmers Guide* and Section 11.5, “FLEXlock,” for additional information on FLEX*lock*.

7.3.9 LM_A_LF_LIST

Type: Pointer to `(char **)`

List of all license files searched for features. Useful for failure messages for debugging. For example:

```
#include "lm_attr.h"
/*...*/
char **cp;
lc_get_attr(job, LM_A_LF_LIST, (short *)&cp);
if (cp)
{
       puts("files searched are: ");
       while (*cp)
             printf("\t%s\n",(short *)*cp++);
}
```

7.3.10 LM_A_LICENSE_DEFAULT

Type: `(char *)`

Note: It is strongly recommended that this attribute be set in all licensed applications.

The expected location of the application's *license file directory*. This is a directory that contains one or more license files with a `.lic` extension. This location can be hardcoded at compile time or determined dynamically at run-time. However, it is recommended to determine this location dynamically at run-time. See the *FLEXlm Programmers Guide* for more information on how this location is used by the licensed application.

If LM_A_LICENSE_DEFAULT is set, FLEX*lm* still honors the *VENDOR*_LICENSE_FILE and LM_LICENSE_FILE environment variables and/or the license finder first.

7.3.11 LM_A_LICENSE_FMT_VER

Type: `(char *)`

Default: LM_BEHAVIOR_V8

Licenses generated by lc_cryptstr() will be compatible with the version specified. Valid arguments are LM_BEHAVIOR_V*x*, where *x* is 2, 3, 4, 5, 5_1, 6, 7, 7_1, 8, 8_1, 8_2, 8_3, or 9_0. If the license compatible with the desired version cannot be generated:

- The error LM_LGEN_VER (-94) will be generated: "Attempt to generate license with incompatible attributes."
- The FEATURE line will be left as is, without replacing the signature with a correct one.

SEE ALSO

- Section 6.4.4, "lc_cryptstr()"

7.3.12 LM_A_LINGER

Type: `(long)`

Default: 0 (no linger)

This option controls the license linger time for your application. Any checkout performed after setting LM_A_LINGER to a non-zero value will cause the license to be held by the vendor daemon for the specified number of seconds after either a checkin or your process exits. The vendor daemon checks for lingering licenses only once per minute, which will limit the granularity of this setting.

SEE ALSO

- Section 11.2, “Lingering Licenses”

7.3.13 LM_A_LONG_ERRMSG

Type: `(int)`

Default: True

The default is long error messages. Error messages can be presented in a long, more descriptive format. The new format contains embedded newline characters, which some applications may not be able to handle, or may need special handling.

Applications will often find it useful to present the short error message first, and then long error message upon user request. This can be done thus:

```
lc_set_attr(job, LM_A_LONG_ERRMSG, (LM_A_VAL_TYPE)0);
....
/*error occurs*/
lc_perror(job);
/* user requests long error message */
lc_set_attr(job, LM_A_LONG_ERRMSG, (LM_A_VAL_TYPE)1);
lc_perror(job);
```

Note that this only works if another FLEX*lm* error doesn’t occur in between, which would change the error condition and message. Not all error conditions have long explanations or context-sensitive information.

Example:

```
Invalid host
  The hostid of this system does not match the hostid
  specified in the license file
Hostid:           12345678
License path:     ./file1.lic:./file2.lic:./file3.lic
FLEXlm error:     -9,9
```

The format is:

```
short-error-description
optional-long-explanation [1-3 lines]
optional-context-information
License path:   path1:...:pathn
FLEXlm error:   major, minor
```

7.3.14 LM_A_PERROR_MSGBOX (Windows Only)

Type: `(int)`

Default: True

If True, lc_perror() presents the error message in an error dialog. Also turned off when `FLEXLM_BATCH` is set.

7.3.15 LM_A_PROMPT_FOR_FILE (Windows Only)

Type: `(int)`

Default: True

When True, the user is prompted for the license file path or server name or IP address, if needed. Also turned off when `FLEXLM_BATCH` is set.

7.3.16 LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL

Type: `(int)`

Default: 5 for LM_A_RETRY_COUNT, 60 for LM_A_RETRY_INTERVAL

Together, LM_A_RETRY_COUNT and LM_A_RETRY_INTERVAL are used for automatic reconnection to a license server. Once license server failure is detected, the automatic heartbeat mechanism attempts to reconnect to the license server. If reconnection fails, then the reconnect will be re-attempted LM_A_RETRY_COUNT times at intervals of LM_A_RETRY_INTERVAL. If the application wants manual control over this activity, enable manual heartbeats by setting LM_A_RETRY_INTERVAL to -1 and LM_A_CHECK_INTERVAL to -1. The minimum value for LM_A_RETRY_INTERVAL is 30 seconds.

If LM_A_RETRY_COUNT is set to -1, the application will attempt retrying forever—for applications desiring a more lenient policy, this is recommended.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”
- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 7.3.18, “LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX”

7.3.17 LM_A_TCP_TIMEOUT

Type: `(int)`

Default: 7200 seconds (2 hours)

Maximum: 15300 seconds (4 hours 15 minutes).

If a TCP client node crashes or the client node is disconnected from the network, the license will be automatically checked back in LM_A_TCP_TIMEOUT seconds later. 0 means no TCP timeout.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”

7.3.18 LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX

Type: Pointer to a function returning `int`. Return value unused.

Default: No user exit handler (licensed application exits)

DESCRIPTION

The function pointer LM_A_USER_EXITCALL (or the extended version, LM_A_USER_EXITCALL_EX) is set to point to the callback routine that is to receive control if reconnection fails after LM_A_RETRY_COUNT attempts. If no routine is specified, then lc_perror() is called, and the licensed application will exit. The LM_A_USER_EXITCALL routine is called as follows:

```
(*exitcall)(feature);
```

or, with LM_A_USER_EXITCALL_EX:

```
(*exitcallEx) (job, feature, vendor_data);
```

This callback function is an exit handler; it should not return. Behavior is undefined if any FLEXlm client routine is called after this exit handler is called. If you want to do anything more subtle than exiting, do not specify an exit handler function. Instead, set LM_A_RETRY_COUNT to -1 and specify an LM_A_USER_RECONNECT function which sets a global variable; monitor this variable in the mainline code (not in a callback function). Take appropriate action when the global variable reaches your defined limit.

Note: LM_A_USER_EXITCALL_EX provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both LM_A_USER_EXITCALL_EX and LM_A_USER_EXITCALL are set.

CALLBACK PARAMETERS

`(LM_HANDLE *) job`	LM_A_USER_EXITCALL_EX only: A pointer to the current license job.
`(char *) feature`	A pointer to the feature name.
`(void *)user_data`	LM_A_USER_EXITCALL_EX only: A pointer to user-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”
- Section 7.3.16, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL”
- Section 8.2.17, “LM_A_VENDOR_CALLBACK_DATA”

7.3.19 LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX

Type: Pointer to a function returning `int`. Return value unused.

Default: No user reconnection handler

DESCRIPTION

The function pointer LM_A_USER_RECONNECT (or the extended version, LM_A_USER_RECONNECT_EX) is set to point to a reconnection callback routine. This routine is called each time just before a reconnection is attempted, either by the automatic heartbeat mechanism, or as a result of the application program calling lc_heartbeat().

The LM_A_USER_RECONNECT routine is called as follows:

```
(*reconnect)(feature, pass, total_attempts,
                    interval)
```

or, with LM_A_USER_RECONNECT_EX:

```
(*reconnectEx)(job, feature, pass, total_attempts,
                    interval, vendor_data)
```

Note: LM_A_USER_RECONNECT_EX provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both LM_A_USER_RECONNECT_EX and LM_A_USER_RECONNECT are set.

CALLBACK PARAMETERS

`(LM_HANDLE *) job`	LM_A_USER_RECONNECT_EX only: A pointer to the current license job.
`(char *) feature`	Feature name.
`(int) pass`	Current attempt number.
`(int) total_attempts`	Maximum number of passes that will be attempted.
`(int) interval`	Time in seconds between reconnection attempts.
`(void *)user_data`	LM_A_USER_RECONNECT_EX only: A pointer to vendor-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute.

If LM_A_RETRY_COUNT is set to a value <=0, then the reconnect handler will not be called.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”
- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 7.3.18, “LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX”
- Section 8.2.17, “LM_A_VENDOR_CALLBACK_DATA”

7.3.20 LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX

Type: Pointer to a function returning `int`. Return value unused.

Default: None.

DESCRIPTION

The function pointer LM_A_USER_RECONNECT_DONE (or the extended version, LM_A_USER_RECONNECT_DONE_EX) is set to point to a callback routine. This routine is called when reconnection is successfully completed.

The LM_A_USER_RECONNECT_DONE handler is called as follows:

```
(*reconnect_done)(feature, tries, total_attempts,
                  interval)
```

or, for LM_A_USER_RECONNECT_DONE_EX:

```
(*reconnect_doneEx)(job, feature, tries, total_attempts,
                  interval, vendor_data)
```

Note: LM_A_USER_RECONNECT_DONE_EX provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both LM_A_USER_RECONNECT_DONE_EX and LM_A_USER_RECONNECT_DONE are set.

CALLBACK PARAMETERS

`(LM_HANDLE *) job`	LM_A_USER_RECONNECT_DONE_EX only: A pointer to the current license job.
`(char *) feature`	Feature name.
`(int) tries`	Number of attempts that were required to re-connect for this feature.
`(int) total_attempts`	Maximum number of retry attempts that would be made.
`(int) interval`	Interval in seconds between reconnection attempts.

`(void *)user_data`	LM_A_USER_RECONNECT_DONE_EX only: A pointer to vendor-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute.

SEE ALSO

- Section 7.3.19, "LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX"
- Section 8.2.17, "LM_A_VENDOR_CALLBACK_DATA"

7.3.21 LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO

Type: Pointer to `LM_VD_GENERIC_INFO` or pointer to `LM_VD_FEATURE_INFO`

Both attributes get information from your vendor daemon. LM_A_VD_GENERIC_INFO gets information which is not specific to a feature, and which is mostly found in `lsvendor.c`.

LM_A_VD_FEATURE_INFO gets information about a particular feature, and provides an accurate count of licenses used, users queued, etc., and works correctly when a license file has more than one FEATURE or INCREMENT line for the same feature name. This will result in a LM_NOSERVSUPP error if the particular `CONFIG` struct has been merged with another `CONFIG` in the vendor daemon.

These attributes will only work on your vendor daemon. If a request is made for a feature only served by a different vendor daemon, then the LM_NOADMINAPI error results.

A pointer to a struct is given as an argument to lc_get_attr(); upon successful return, this struct is filled with the appropriate information. The following example illustrates the use of both attributes. Though lc_get_config() and lc_next_conf() are described in Section 8.1, "Advanced FLEXible API Functions," one of their legitimate uses is with LM_A_VD_GENERIC_INFO and LM_A_VD_FEATURE_INFO.

Note: If you are reporting on a feature that has been checked out successfully, use lc_auth_data(), instead of lc_next_conf().

```
#include "lmclient.h"
#include "lm_code.h"
#include "lm_attr.h"
/* ... */
/*
 * Print out GENERIC and FEATURE information for every
 * license file line for a given feature name
 */
void
vendor_daemon_info(LM_HANDLE *job, char *feature)
{
  CONFIG *conf, *c;
  LM_VD_GENERIC_INFO gi;
  LM_VD_FEATURE_INFO fi;
  int first = 1;

    c = (CONFIG *)0;

    for (conf = lc_next_conf(job, feature, &c);conf;
                          conf=lc_next_conf(job, feature, &c))
    {
        if (first)

        {
/*
 *           get generic daemon info
 */
            gi.feat = conf;
            if (lc_get_attr(job, LM_A_VD_GENERIC_INFO,
                                 (short *)&gi))
            {
                  lc_perror(job, "LM_A_VD_GENERIC_INFO");
            }
            else
            {
                  printf(" conn-timeout %d\n",
                                       gi.conn_timeout);
                  printf(" normal_hostid %d\n",
                                       gi.normal_hostid);
                  printf(" minimum_user_timeout %d\n",
                                       gi.minimum_user_timeout);
                  printf(" min_lmremove %d\n",
                                       gi.min_lmremove);
                  printf(" use_featset %d\n",
                                       gi.use_featset);
                  printf(" dup_sel 0x%x\n", gi.dup_sel);
                  printf(" use_all_feature_lines %d\n",
                                       gi.use_all_feature_lines);
                  printf(" do_checkroot %d\n",
                                       gi.do_checkroot);
```

```
                    printf(" show_vendor_def %d\n",
                                        gi.show_vendor_def);
                }
                 first = 0;
            }
/*
 *      get specific feature info
 */
            fi.feat = conf;
            if (lc_get_attr(job, LM_A_VD_FEATURE_INFO,
                                                (short *)&fi))
            {
                lc_perror(job, "LM_A_VD_FEATURE_INFO");
            }
            else
            {
                 printf("\nfeature s\n", conf->feature);
                 printf("code %s\n", conf->code);
                 printf("rev %d\n", fi.rev);
                 printf("timeout %d\n", fi.timeout);
                 printf("linger %d\n", fi.linger);
                 printf("res %d\n", fi.res);
                 printf("tot_lic_in_use %d\n",
                                        fi.tot_lic_in_use);
                 printf("float_in_use %d\n",
                                        fi.float_in_use);
                 printf("user_cnt %d\n", fi.user_cnt);
                 printf("num_lic %d\n", fi.num_lic);
                 printf("queue_cnt %d\n", fi.queue_cnt);
                 printf("overdraft %d\n", fi.overdraft
            }
    }
}
```

DETECTING OVERDRAFT FOR SUITES

This is a special case for OVERDRAFT. With suites, when you check out a feature, you also silently check out a token for the suite. Both the suite and feature token may be in the OVERDRAFT state, or only one, or neither. To detect suite overdraft, the code must get the parent/suite feature name, and then check for overdraft for this feature. Use lc_auth_data() to ensure that you get the `CONFIG` struct for the license that has been checked out.

```
conf=lc_auth_data();
if ((conf->package_mask & LM_LICENSE_PKG_COMPONENT)
    && (conf->package_mask & LM_LICENSE_PKG_SUITE))
{
    fi.feat = conf->parent_feat;
    if (lc_get_attr(job, LM_A_VD_FEATURE_INFO, (short *)&fi))
             lc_perror(job, "LM_A_VD_FEATURE_INFO");
```

```
        else
                printf("suite overdraft is %d\n", fi.overdraft);
    }
```

SEE ALSO

- Section 8.1.7, “lc_get_config()”
- Section 8.1.10, “lc_next_conf()”
- Section 6.4.1, “lc_auth_data()”

7.3.22 LM_A_VENDOR_ID_DECLARE

Type: Pointer to `LM_VENDOR_HOSTID` struct.

Default: None

This is for supporting vendor-defined hostid. The struct defines and declares the hostid to FLEX*lm*.

SEE ALSO

- Chapter 15, “Vendor-Defined Hostid Types.”
- `machind/lmclient.h` for `LM_VENDOR_HOSTID` definition
- `examples/vendor_hostid` directory

7.3.23 LM_A_VERSION, LM_A_REVISION

Type: `(short)`

Default: Version and revision of the libraries you have linked with FLEX*lm* version. Cannot be set. Only for use with lc_get_attr().

7.3.24 LM_A_WINDOWS_MODULE_HANDLE (Windows only)

Type: `(long)`

Default: 0

This is only needed for a specific situation on Windows: You are building a DLL, and the FLEX*lm* library (`lmgr.lib`) is linked into your DLL. Or put another way, the FLEX*lm* functions are not in a static binary, but only in a DLL. In this case, the DLL makes calls to the following functions before calling lc_checkout():

```
lc_set_attr(job, LM_A_WINDOWS_MODULE_HANDLE,
         (LM_A_VAL_TYPE)GetModuleHandle(dllname));
```

where *`dllname`* is the name of the DLL. If these calls are not made, Windows dialogs and error messages do not work properly.

Chapter 8

Advanced FLEXible API Features

The FLEXible API functions, attributes, and vendor variables in this chapter provide advanced FLEX*lm* functionality. Basic functionality is covered by the material in Chapter 6, “FLEXible API,” Chapter 7, “Controlling Licensing Behavior,” and Chapter 13, “Vendor Daemon.”

8.1 Advanced FLEXible API Functions

The following FLEXible API functions are advanced and used for functionality not found in the basic set of functions. They should be used with care, and questions are welcomed before their use. Table 8-1 summarized these functions.

Table 8-1: Advanced FLEXible API Functions

Function	Description
l_new_hostid()	Allocates a hostid structure.
lc_borrow_return()	Returns a borrowed license early.
lc_check_key()	Validates a license signature.
lc_cleanup() (Windows only)	Cleans up FLEXlm resources after they are no longer needed.
lc_convert()	Converts a license file from one format to the other. Formats include readable and decimal.
lc_free_hostid()	Frees the memory associated with a hostid structure.

Table 8-1: Advanced FLEXible API Functions (Continued)

Function	Description
lc_get_config()	Returns license file data for a given feature.
lc_get_errno()	Returns the most recent FLEX*lm* error.
lc_init_simple_composite()	Initializes a composite hostid.
lc_next_conf()	Returns the next line in the license file matching the given feature.
lc_remove()	Removes a given feature for the specified user.
lc_shutdown()	Shuts down the FLEX*lm* license servers.
lc_test_conf()	Returns license file data for the most recently tested feature.

8.1.1 l_new_hostid()

SYNTAX

```
hostid = l_new_hostid()
```

DESCRIPTION

Returns a malloc'd and zeroed hostid. Use lc_free_hostid() to free this memory. This may be needed when doing vendor-defined hostids.

PARAMETERS

None.

RETURN

`(HOSTID *)` *`hostid`*	A `HOSTID` struct, or null.

ERROR RETURNS

LM_CANTMALLOC	malloc() call failed.

SEE ALSO

- Section 7.3.22, “LM_A_VENDOR_ID_DECLARE”
- Chapter 15, “Vendor-Defined Hostid Types”

8.1.2 lc_borrow_return()

SYNTAX

```
status = lc_borrow_return(job, feature, display)
```

DESCRIPTION

lc_borrow_return() is used to return a borrowed license to the pool of available licenses before the borrow period expires. The application which calls this routine must be running on the same machine from which the license was originally borrowed.

In order for this routine to have an effect, the license server must be configured to allow early return of borrowed licenses.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(char *) *feature*	The name of the borrowed feature to be returned early.
(char *) *display*	The display from which the borrow was initiated. Required if the current display is different than what was used to initiate the borrow. On Windows, it is the system name or, in the case of a terminal server environment, the terminal server client name. On UNIX, it is in the form `/dev/tty`*xx* or the X-Display name. If NULL, FLEX*lm* uses the current display name, determined via the standard method appropriate to the platform.

RETURN

(int) *status*	The FLEX*lm* error code, or 0 for no error.

ERROR RETURNS

LM_BADPARAM	Problem with *job* or *feature* argument.
LM_BORROW_ERROR	Cannot read the local borrow info.
LM_BORROW_RETURN_SERVER_ERR	The server generated an error during a license return attempt.
LM_CANTCONNECT	Cannot connect to license server to return the license.
LM_CANTWRITE	Cannot send data to license server
LM_NOFEATURE	The feature specified doesn't match any borrowed items on the client side.

SEE ALSO

- Section 8.3.4, “ls_borrow_return_early”

8.1.3 lc_check_key()

SYNTAX

```
status = lc_check_key(job, conf, code)
```

DESCRIPTION

lc_check_key() determines if the signature in the `CONFIG` structure, pointed to by `conf`, is valid. This function is optional and only for use during license installation; it is not used in an application that also calls lc_checkout(). If authenticated license information is needed without checking out a license, use lc_checkout() with the LM_CO_LOCALTEST flag.

To verify a license file upon installation, use code similar to the following example:

```
VENDORCODE code;
lc_new_job(..., &code, ...);
feats = lc_feat_list(...);
while (*feats)
{
    pos = 0;
    while (conf = lc_next_conf(job, *feats, &pos))
    {
         if (lc_check_key(job, conf, &code))
```

```
                /*error*/
        }
        feats++;
    }
```

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(CONFIG *) *conf*	From lc_next_conf(), lc_get_config().
pointer to (VENDORCODE) *code*	From lc_new_job().

RETURN

(int) *status*	The FLEX*lm* error code, or 0 for no error.

ERROR RETURNS

LM_BADCODE	Signature is invalid—license has been typed incorrectly, or altered in some way.
LM_BADPARAM	Problem with *conf* argument.
LM_FUTURE_FILE	License format is invalid and may be from a “future” FLEX*lm* version.

SEE ALSO

- examples/advanced/exinstal.c
- Section 8.1.10, “lc_next_conf()”
- Section 8.1.5, “lc_convert()”
- Section 6.4.8, “lc_feat_list()”
- Section 6.4.3, “lc_checkout()”

8.1.4 lc_cleanup() (Windows only)

SYNTAX

```
(void) lc_cleanup()
```

DESCRIPTION

This function frees all allocated memory, releases all handles, and kills all threads that are created by FLEX*lm* client library routines. lc_cleanup() must be the last FLEX*lm* client library routine that is called by your FLEXenabled application. Do not call any other FLEX*lm* client library routines after calling this routine.

SPECIAL CONSIDERATIONS

This routine is required if:

- Your FLEXenabled application dynamically links to the FLEX*lm* client library

  Call lc_cleanup() before the DLL is unmapped from the application's address space.

- You distribute your product as a DLL

  To ensure all FLEX*lm* resources are properly cleaned up after your end user's application unmaps your FLEXenabled DLL, make sure you call lc_cleanup() in the logical clean up point in your DLL.

8.1.5 lc_convert()

SYNTAX

```
status = lc_convert(job, str, return_str, errors, flag)
```

DESCRIPTION

This is an API for companies that want to provide their own front-end for installing license files. lc_convert() can be used in combination with lc_check_key() to provide a user-friendly front-end.

lc_convert() also changes `this_host` in the SERVER line to the real host name in either decimal or readable licenses. It does this only if lc_convert() is run on the same hostid as appears on the SERVER line and does not do this for hostids of DEMO or ANY.

If readable output is requested, the output will be compatible with the LM_A_LICENSE_FMT_VER setting, which defaults to the current FLEX*lm* version.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job()
(char *) *str*	License file (in readable or decimal format) as a string.
pointer to (char *) *return_str*	*str* converted to desired format. Should be freed by caller; use lc_free_mem() on Windows.
pointer to (char *) *errors*	If return value is non-zero, then this is set to a description of the problem. Should be freed by caller; use lc_free_mem() on Windows.
(int) *flag*	LC_CONVERT_TO_READABLE or LC_CONVERT_TO_DECIMAL, defined in lmclient.h.

RETURN

(int) *status*	0 == success. -1, if syntax error in *str*, and *errors* is set to explanatory message. Otherwise, FLEX*lm* errno.

ERROR RETURNS

LM_BADPARAM	Invalid *flag* argument.

SEE ALSO

- examples/advanced/exinstal.c for an example program
- Section 8.1.10, “lc_next_conf()”
- Section 8.1.5, “lc_convert()”
- Section 6.4.8, “lc_feat_list()”

- Section 6.4.4, “lc_cryptstr(),” because lc_convert() has a similar interface to lc_cryptstr()
- Section 7.3.11, “LM_A_LICENSE_FMT_VER”

8.1.6 lc_free_hostid()

SYNTAX

```
(void) lc_free_hostid(job, hostid)
```

DESCRIPTION

lc_free_hostid() frees the memory associated with a hostid which has been allocated with l_new_hostid() or lc_copy_hostid(). If passed a hostid list, lc_free_hostid() frees the whole list.

lc_free_job() removes any timers associated with *job*.

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().
`(HOSTID *)` *hostid*	From l_new_hostid().

RETURN

None.

ERROR RETURNS

LM_BADPARAM	No such job.

SEE ALSO

- Section 8.1.1, “l_new_hostid()”

8.1.7 lc_get_config()

SYNTAX

```
conf = lc_get_config(job, feature)
```

DESCRIPTION

Gets the license file data for a given feature. FLEX*lm* allows multiple valid FEATURE and INCREMENT lines (of the same feature name) in a license file. lc_get_config() will return the first `CONFIG` struct, and lc_next_conf() retrieves the next (lc_next_conf() can also find the first). lc_get_config() does

not authenticate feature lines. That is, a user can type in a FEATURE line with an invalid signature, and lc_get_config() will still return it. lc_get_config() and lc_next_conf() are usually needed only with LM_A_VD_GENERIC_INFO or LM_A VD_FEATURE_INFO.

To get authenticated information from a FEATURE line, you must first check out the feature, and then use lc_auth_data().

PARAMETERS

`(LM_HANDLE *)` *job*	From lc_new_job().
`(char *)` *feature*	The desired feature.

RETURN

`(CONFIG *)` *conf*	The `CONFIG` struct. If no feature found, then NULL. The `CONFIG` struct is defined in the header file `lmclient.h`.

ERROR RETURNS

LM_NOFEATURE	Specified feature does not exist.
LM_NOCONFFILE	License file does not exist.
LM_BADFILE	License file corrupted.
LM_NOREADLIC	Cannot read license file.
LM_SERVNOREADLIC	Cannot read license data from license server.

SEE ALSO

- Section 6.4.1, “lc_auth_data()”
- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 6.4.3, “lc_checkout()”

- Section 8.1.10, “lc_next_conf()”
- Section 7.3.21, “LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO”

8.1.8 lc_get_errno()

SYNTAX

```
error = lc_get_errno(job)
```

DESCRIPTION

This function returns the most recently set FLEX*lm* error number. This value is available after the return of any FLEX*lm* function.

For more detailed error information, which includes the error number, use lc_err_info() as an alternative to this function.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().

RETURN

`(int) error`	See `lmclient.h`, `lm_lerr.h`, and `lmerrors.h` for a list of possible FLEX*lm* errors and associated English descriptions.

SEE ALSO

- Section 6.4.5, “lc_err_info()”
- Section 6.4.19, “lc_perror()”
- `machind/lmclient.h`

8.1.9 lc_init_simple_composite()

SYNTAX

status = lc_init_simple_composite(*job, hostid_list, num_ids*)

DESCRIPTION

This function initializes a vendor-defined composite hostid. Invoke this function right after the call to lc_new_job(). The composite hostid is valid for the life of the license job and is accessed by specifying the HOSTID_COMPOSITE hostid type to lc_hostid().

A composite hostid is a 12-byte hashed hexidecimal value formed by combining the values of one or more simple hostids types, as specified by *hostid_list*. If an invalid hostid type for the platform appears in *hostid_list*, lc_init_simple_composite() returns the value of the default hostid type for that platform.

EXAMPLE

The following example demonstrates the definition and use of a composite hostid made up from HOSTID_ETHER and HOSTID_DISPLAY.

```
#include lmclient.h
LM_HANDLE *job
char buf[MAX_CONFIG_LINE];
/* Set up the list of hostid types which comprise 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_DISPLAY};
int num_ids = 2;
   /*...*/
lc_new_job(..., &job);
/* Register the composite hostid */
ret = lc_init_simple_composite(job, hostid_list, num_ids);
if (ret != 0)
{
   /* error processing */
}
/* ... */
/* Now, access the composite hostid value */
if (lc_hostid(job, HOSTID_COMPOSITE, buf))
{
   /* error processing */
}
```

PARAMETERS

`(LM_HANDLE *)` *`job`*	From lc_new_job().
`(int *)`*`hostid_list`*	A pointer to an integer array of one or more simple hostid types to be included in the composite hostid. See Section 6.4.13, “lc_hostid(),” for a list of valid hostid types.
`int` *`num_ids`*	The number of hostid types specified in *`hostid_list`*.

RETURN

`(int)` *`status`*	0 = Success. !0 = Failure, FLEX*lm* errno is returned.

ERROR RETURNS

LM_COMPOSITEID_ INIT_ERR	Error initializing the composite hostid.
LM_COMPOSITEID_ ITEM_ERR	An item needed for composite hostid is missing or invalid.

SEE ALSO

- Section 6.4.13, “lc_hostid()”
- `machind/lmclient.h`

8.1.10 lc_next_conf()

SYNTAX

```
CONFIG *pos = 0;
conf = lc_next_conf(job, feature, &pos);
```

DESCRIPTION

Returns the next line in the license file, as a pointer to a CONFIG structure, matching *feature*. The search is started from *pos*, where *pos* = 0 indicates the first line. lc_next_conf() automatically updates *pos*, so it is ready to retrieve the next feature line on a subsequent call.

lc_next_conf() does not authenticate FEATURE lines. That is, a user can type in a FEATURE line with an invalid signature, and lc_next_conf() will still return it. lc_get_config() and lc_next_conf() are usually needed only with LM_A_VD_GENERIC_INFO or LM_A VD_FEATURE_INFO.

To get authenticated information from a FEATURE line, you must first check out the feature, and then use lc_auth_data().

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(char *) *feature*	The desired feature line.

RETURN

(CONFIG *) *conf*	The CONFIG struct. If none found, then NULL.
pointer to (CONFIG *) *pos*	Declare CONFIG *pos = 0; use &pos for argument. Internally updated by lc_next_conf() to next license file entry.

ERROR RETURNS

See error returns for lc_get_config().

EXAMPLE

```
CONFIG *pos = 0, *conf;
while (conf = lc_next_conf(job, "myfeature", &pos))
{
    /* ... */
}
```

SEE ALSO

- Section 6.4.1, “lc_auth_data()”
- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 6.4.3, “lc_checkout()”
- Section 8.1.7, “lc_get_config()”
- Section 7.3.21, “LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO”

8.1.11 lc_remove()

SYNTAX

status = lc_remove(*job*, *feature*, *user*, *host*, *display*)

DESCRIPTION

Removes the specified user’s license for *feature*. This is used by the `lmremove` command, and has the same restrictions regarding the “lmadmin” group. lc_remove() normally is only used when the client’s system has had a hard crash, and the server does not detect the client node failure. If lc_remove() is called on a healthy client, the license will be checked out again by the client with its next heartbeat.

Note: If `lmgrd` is started with the `-x lmremove` flag, then lc_remove() has no effect.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(char *) *feature*	Remove the license for this feature.
(char *) *user*	User name of license to remove.
(char *) *host*	Host name of license to remove.
(char *) *display*	Display name of license to remove.

RETURN

`(int) status`	0—OK,!=0, error status.

ERROR RETURNS

LM_BADCOMM	Communications error.
LM_BADPARAM	No licenses issued to this user.
LM_CANTCONNECT	Cannot connect to license server.
LM_CANTREAD	Cannot read from license server.
LM_CANTWRITE	Cannot write to license server.
LM_NOFEATURE	Feature not found in license file data.
LM_NOTLICADMIN	Failed because user is not in “lmadmin” group.
LM_REMOVETOOSOON	Failed because `ls_min_lmremove` time has not elapsed.

SEE ALSO

- Chapter 10, “The License Manager Daemon”
- Section 13.2.8, “ls_min_lmremove”
- Section 6.4.12, “lc_heartbeat()”

8.1.12 lc_shutdown()

SYNTAX

```
status = lc_shutdown(job, prompt, print)
```

DESCRIPTION

Shuts down the FLEX*lm* servers. This is used by `lmdown`.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(int) prompt`	Unused.
`(int) print`	Unused.

RETURN

`(int) status`	0 — server not shut down; <> 0 — server shut down.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOTLICADMIN	You are not an authorized license administrator.
LM_CANTREAD	Cannot read data from license server.

SEE ALSO

- Chapter 10, “The License Manager Daemon”

8.1.13 lc_test_conf()

SYNTAX

```
conf = lc_test_conf(lm_job)
```

DESCRIPTION

Retrieves the license file line for the most recently tested feature. A feature is tested, but not checked out, when the LM_CO_LOCALTEST flag is set in the call to lc_checkout(). Use lc_test_conf() to retrieve information from a license file for tested features. For checked out features, use lc_auth_data().

The behavior of lc_test_conf() is undefined if called in any other context other than after a tested feature.

The following code excerpt demonstrates using this function. In this example, the application obtains information for the feature just tested in order to display various fields. For this example, consider the following feature line from the license file:

```
FEATURE f1 demo 1.0 permanent 4 NOTICE="ab" \
        SIGN=xxxxxxxxxxx
```

The code appears as follows in the application:

```
VENDORCODE code;
LM_HANDLE *lm_job;
CONFIG *conf;

lc_new_job(0, lc_new_job_arg2, &code, &lm_job)

/* Now, perform the test checkout*/
if (lc_checkout(lm_job, "f1",...,...,LM_CO_LOCALTEST,...))
{
     /* error processing
}
else
{
     /* Retrieve information for feature "f1" */
     conf = lc_test_conf(lm_job);
     /* Display information for feature "f1" */
     printf("feature=%s version=%s daemon=%s \n",
             conf->feature, conf->version, conf->daemon);
     }
}
```

PARAMETERS

(LM_HANDLE *) *lm_job*	From lc_new_job().

RETURN

(CONFIG *) *conf*	The CONFIG struct, or NULL if error. The CONFIG struct is defined in the header file lmclient.h.

ERROR RETURNS

None.

SEE ALSO

- `lmclient.h` for the `CONFIG` struct definition
- Section 6.4.1, “lc_auth_data()”
- Section 6.4.3, “lc_checkout()”
- Section 8.1.7, “lc_get_config()”
- Section 8.1.10, “lc_next_conf()”

8.2 Advanced FLEXible API Attributes

The attributes in this section provide advanced capabilities beyond what is provided by the basic attributes in Chapter 7, “Controlling Licensing Behavior.” They should be used with care, and questions are welcomed before their use. These attributes are summarized in Table 8-2.

Table 8-2: Advanced FLEXible API Attributes

Attribute	Description
LM_A_BEHAVIOR_VER	Sets FLEX*lm* behavior to the given version.
LM_A_CHECKOUT_DATA	Allows labeling of a given checkout request.
LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX	Defines a pointer to a checkout filter function for feature pre-processing.
LM_A_CHECKOUTFILTERLAST_EX	Defines a pointer to a checkout filter function for feature post-processing.
LM_A_CRYPT_CASE_SENSITIVE	Controls case-sensitivity of license file authentication.
LM_A_DIAGS_ENABLED	Controls the output of detailed diagnostic data.

Table 8-2: Advanced FLEXible API Attributes (Continued)

Attribute	Description
LM_A_DISABLE_ENV	Controls the use of the `LM_LICENSE_FILE` environment variable.
LM_A_DISPLAY_OVERRIDE	Overrides the display name.
LM_A_HOST_OVERRIDE	Overrides the host name.
LM_A_LICENSE_CASE_SENSITIVE	Controls license file case-sensitivity.
LM_A_MT_HEARTBEAT (UNIX Only)	Controls the use of multi-threaded heartbeats.
LM_A_PERIODIC_CALL	Defines a pointer to a function to be called after the specified number of heartbeats.
LM_A_PERIODIC_COUNT	Specifies the number of heartbeats between calls to function specified by `LM_A_PERIODIC_CALL`.
LM_A_PLATFORM_OVERRIDE	Overrides the platform name.
LM_A_RETRY_CHECKOUT	Controls checkout retries.
LM_A_USER_OVERRIDE	Overrides the user name.

8.2.1 LM_A_BEHAVIOR_VER

Type: `(char *)`

Default: `LM_BEHAVIOR_V8`

This sets the behavior of the FLEX*lm*-enabled application to the given version of FLEX*lm*.

Valid values are LM_BEHAVIOR_V*x*, where *x* is `2`, `3`, `4`, `5`, `5_1`, `6`, `7`, `7_1`, `8`, `8_1`, `8_2`, or `8_3`.

SEE ALSO

- Section 13.2.1, “ls_a_behavior_ver”

8.2.2 LM_A_CHECKOUT_DATA

Type: `(char *)`

Default: None

The LM_A_CHECKOUT_DATA attribute allows you to set a checkout-data string. It is used to label the next successful checked out feature or to qualify a feature check in request with the checkout-data label.

The checkout-data string is a character string, with a maximum size of MAX_VENDOR_CHECKOUT_DATA bytes (32 bytes). The default value is the NULL string. Each unique value of LM_A_CHECKOUT_DATA represents a unique license group; the NULL string can be one of those unique values. Like all other attributes, set this before the checkout or checkin request; it takes effect for all subsequent calls to lc_checkout() or lc_checkin() until it is changed.

There are two scenarios where a checkout-data string, defined via the LM_A_CHECKOUT_DATA attribute setting, can be used:

- To force the license count to be accumulated incrementally across multiple checkouts in the same license job rather than as an aggregate.
- To create a custom duplicate grouping criteria.

These scenarios are described below.

INCREMENTAL LICENSE COUNT

By default, checkout requests for the same feature and version from the same license job have an aggregate affect on the number of licenses consumed. That is, the FLEX*lm* client library checks out only as many additional licenses for the feature as necessary to reach the count specified in the checkout request. For example:

If an initial checkout request which asks for two licenses for feature “f1”

```
lc_checkout(lm_job, f1, "1.0", 2,...,LM_DUP_NONE);
```

is followed by a second call which asks for five licenses for the same feature,

```
lc_checkout(lm_job, f1, "1.0", 5,...,LM_DUP_NONE);
```

the second call checks out only three additional licenses (5 minus 2). If the second call asks for the same or fewer number of licenses than were already checked out, the second call does not check out additional licenses.

To force identical checkout requests, instead, to have an incremental effect on the number of consumed licenses, include each call to lc_checkout() in its own license job. Alternatively, use the LM_A_CHECKOUT_DATA attribute to set a unique checkout-data string to the job just prior to the call to lc_checkout(). This causes the second or subsequent request to check out exactly the number of licenses specified in the call to lc_checkout(), regardless of the number asked for in previous requests. For example:

```
/* Check out 2 licenses for f1,
   label them "first checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "first checkout");
lc_checkout(lm_job, f1, "1.0", 2,...,LM_DUP_NONE)

/* Check out 5 more licenses for f1,
   label them "second checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "second checkout");
lc_checkout(lm_job, f1, "1.0", 5,...,LM_DUP_NONE)

/* Check in 2 licenses for f1 labeled "first checkout " */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "first checkout");
lc_checkin(lm_job, f1, 0);

/* Check in 5 licenses for f1 labeled "second checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "second checkout");
lc_checkin(lm_job, f1, 0);
```

This makes it possible to have different sets of licenses for a given feature without having to create a separate license job. Each set is labeled with a different checkout-data string, and is tracked separately by the license server. Subsequent check in requests then can qualify the feature with the checkout-data string by setting it with the LM_A_CHECKOUT_DATA attribute before calling lc_checkin(). Each checkout or checkin request uses the value of the checkout-data string from the last call to lc_set_attr().

CUSTOM DUPLICATE GROUPING

Duplicate grouping based on USER, HOST, or DISPLAY may not represent the duplicate grouping criteria you need. The checkout-data string can be used to group duplicates via a custom vendor criteria, in addition to the USER/HOST/DISPLAY duplicate grouping criteria. Set the checkout-data string using the LM_A_CHECKOUT_DATA attribute and then, in the subsequent call to lc_checkout(), set the LM_DUP_VENDOR bit in the duplicate grouping bitmask. For example, if user A's application performs the following checkout request:

```
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "dup1");
lc_checkout(lm_job, f1, "1.0", 1,...,LM_DUP_VENDOR)
```

and user B's application makes a similar request:

```
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "dup1");
lc_checkout(lm_job, f1, "1.0", 1,...,LM_DUP_VENDOR)
```

then the two checkout requests are considered duplicates, based on the vendor duplicate grouping criteria of `dup1`, and only one license is consumed.

Note: Duplicate grouping criteria can also be expressed in the license file via the DUP_GROUP keyword on FEATURE/INCREMENT lines. See Section 3.5.7, "DUP_GROUP," for further details.

If a custom vendor duplicate grouping criteria is specified, via either LM_DUP_VENDOR in the call to lc_checkout() or DUP_GROUP on the FEATURE/INCREMENT line, LM_A_CHECKOUT_DATA must set a checkout-data string.

USER VISIBILITY

You have the option in your vendor daemon of allowing the LM_A_CHECKOUT_DATA string to be visible or not. The daemon variable `ls_show_vendor_def` controls whether the checkout-data string is visible to your end users via `lmstat` (or any utility which calls lc_userlist()).

If you are considering using this attribute, we recommend that you contact Technical Support for guidance.

SEE ALSO

- Section 6.4.3, "lc_checkout()"
- Section 6.4.20, "lc_set_attr()"
- Section 11.4, "Multiple Jobs"
- Section 13.2.11, "ls_show_vendor_def"

8.2.3 LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX

Type: Pointer to a function returning `int`.

Default: None

DESCRIPTION

The function pointer LM_A_CHECKOUTFILTER (or the extended version, LM_A_CHECKOUTFILTER_EX) is set to point to a checkout filter callback function. This filter function is invoked each time lc_checkout() finds a FEATURE/INCREMENT line that is a candidate for fulfilling the license request. Candidates are selected based on the feature name.

This filter provides the application the opportunity to examine the FEATURE/INCREMENT line before lc_checkout() processes it, and either allows lc_checkout() to proceed processing the line or rejects this particular line. lc_checkout() may still reject the line even though the filter function allows processing to proceed.

The LM_A_CHECKOUTFILTER routine is called as follows:

```
status = (*myCheckoutFilter)(config);
```

or, with LM_A_CHECKOUTFILTER_EX:

```
status = (*myCheckoutFilterEx)(job, config, vendor_data);
```

Note: LM_A_CHECKOUTFILTER_EX provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both LM_A_CHECKOUTFILTER_EX and LM_A_CHECKOUTFILTER are set.

CALLBACK PARAMETERS

`(LM_HANDLE *) job`	LM_A_CHECKOUTFILTER_EX only: A pointer to the license job that was passed to the call to lc_checkout().
`(CONFIG *) config`	A pointer to the `CONFIG` struct representing the candidate feature.
`(void *)vendor_data`	LM_A_CHECKOUTFILTER_EX only: A pointer to either vendor-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute or NULL.

RETURN VALUES

The return value from this function effects the outcome of the current checkout request in the following ways

- If `0` is returned —

  lc_checkout() continues to evaluate the FEATURE line in an attempt to satisfy the license request.

- If a non-zero value is returned —

  lc_checkout() does not continue to evaluate the current FEATURE line but, instead, takes one of the following actions:

  - Invokes this filter function, again, with a subsequent candidate FEATURE line for the license request.
  - If there are no more candidate FEATURE lines, lc_checkout() fails and sets the FLEX*lm* error number to LM_LOCALFILTER; the license is not checked out.

SEE ALSO

- Section 6.4.3, “lc_checkout()”
- Section 8.2.4, “LM_A_CHECKOUTFILTERLAST_EX”
- Section 8.2.17, “LM_A_VENDOR_CALLBACK_DATA”

8.2.4 LM_A_CHECKOUTFILTERLAST_EX

Type: Pointer to a function returning `int`.

Default: None

DESCRIPTION

The function pointer LM_A_CHECKOUTFILTERLAST_EX is set to point to a checkout filter callback function. This filter function is invoked after lc_checkout() authenticates a FEATURE/INCREMENT line locally as being valid for fulfilling the license request.

This filter provides the application the opportunity to examine the FEATURE/INCREMENT line after lc_checkout() authenticates it locally, and either allows lc_checkout() to proceed processing the line or rejects this particular line. If the filter function allows processing to proceed, lc_checkout() passes a served license request to the license server where it still may get denied.

The LM_A_CHECKOUTFILTER_EX routine is called as follows:

```
status = (*myCheckoutFilterLastEx)(job, config, vendor_data);
```

CALLBACK PARAMETERS

`(LM_HANDLE *) job`	A pointer to the license job that was passed to the call to lc_checkout().
`(CONFIG *) config`	A pointer to the `CONFIG` struct representing the authenticated feature line.
`(void *)vendor_data`	A pointer to either vendor-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute or NULL.

RETURN VALUES

The return value from this function effects the outcome of the current checkout request in the following ways

- If `0` is returned —

  For served licenses, lc_checkout() passes the license request to the license server for further processing. For unserved licenses, lc_checkout() checks out the license.

- If a non-zero value is returned —

  lc_checkout() does not continue to evaluate the current FEATURE line but, instead, takes one of the following actions:

  - Invokes this filter function, again, with a subsequent authenticated FEATURE line for the license request.
  - If there are no more candidate FEATURE lines, lc_checkout() fails and sets the FLEX*lm* error number to LM_LOCALFILTER; the license is not checked out.

SEE ALSO

- Section 6.4.3, “lc_checkout()”
- Section 8.2.3, “LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX”
- Section 8.2.17, “LM_A_VENDOR_CALLBACK_DATA”

8.2.5 LM_A_CRYPT_CASE_SENSITIVE

Type: (short)

Default: Case-insensitive comparison

If specified as a non-zero integer, LM_A_CRYPT_CASE_SENSITIVE will cause the output of the authentication function to be compared to the code in the license file with a case-sensitive comparison.

8.2.6 LM_A_DIAGS_ENABLED

Type: (short)

Default: On (1)

This option allows FLEX*lm* to produce some diagnostic output for failures of the lc_checkout() call if the environment variable FLEXLM_DIAGNOSTICS is set. If LM_A_DIAGS_ENABLED is set to 0, this diagnostic information is unconditionally disabled.

The FLEXLM_DIAGNOSTICS environment variable can be used by your end users to obtain more information if a checkout fails. If FLEXLM_DIAGNOSTICS is set, an lc_perror() call is made. If FLEXLM_DIAGNOSTICS is set to "2," then in addition to the lc_perror() call, the arguments to lc_checkout() (except for the KEY information) are printed to stderr, also (on Windows, this is logged to flex_err.log).

The diagnostics are enabled by default. Macrovision recommends that this be left enabled. This will allow us to help you debug your end users' problems with error messages more explicit than, "can't get license." In these situations, we are unable to help. We developed and distributed the FLEXLM_DIAGNOSTICS to enable us (and your support people) to help your end users more effectively.

8.2.7 LM_A_DISABLE_ENV

Type: (short)

Default: LM_LICENSE_FILE environment variable enabled

If set to a non-zero value, disable_env will force the FLEX*lm* client functions to disregard the setting of the LM_LICENSE_FILE environment variable. It's rare that there's a legitimate reason to use this, but it does come up with certain utilities that may explicitly need to ignore the LM_LICENSE_FILE

environment variable. It is strongly discouraged that this be used in your applications, as many end user sites are familiar with FLEX*lm*, and need to assume that `LM_LICENSE_FILE` will be effective.

Note: This must be set *before* LM_A_LICENSE_DEFAULT to be effective.

8.2.8 LM_A_DISPLAY_OVERRIDE

Type: `(char *)`

Default: No override of display name

Note: This value cannot be changed for a job after the initial connection to the vendor daemon.

WINDOWS PLATFORMS

This string, if specified, is used to override the display name as derived from the WTSQuerySessionInformation() call from the Platform SDK Terminal Services API. The most common use of this attribute is for setting the display to the remote hostname in a Terminal Server environment.

UNIX PLATFORMS

This string, if specified, is used to override the display name as derived from the UNIX ttyname() system call.

The most common use of this attribute is for setting the display to the X-Display name. Unfortunately, the only reliable way of obtaining the name of the X-Display is via a call to an X-based routine. Therefore, this can only be done by the X-based application, after XOpenDisplay() (or XtAppInitialize()) has been called.

The Display name is available via the X macro DisplayString(*`display`*).

In addition, it is essential to note that there are at least three possible aliases for using the monitor attached to the computer in use: `localhost:0`, `unix:0`, and `:0`. If any of these are used, `LM_A_DISPLAY_OVERRIDE` should use the result of gethostname() instead. Finally, it may be safest to use the IP address as a string to avoid the problem of aliases for a particular display host.

8.2.9 LM_A_HOST_OVERRIDE

Type: `(char *)`

Default: No override of host name

This string, if specified, will be used to override the host name as derived from the UNIX gethostname() system call.

Note: This value cannot be changed for a job after the initial connection to the vendor daemon.

8.2.10 LM_A_LICENSE_CASE_SENSITIVE

Type: `(int)`

Default: False

If `True`, the license file is case-sensitive. This should be set to `True` to generate license files compatible with older versions of FLEX*lm*. This attribute is automatically set to `True` if the LM_A_LICENSE_FMT_VER or LM_A_BEHAVIOR_VER attributes are set to LM_BEHAVIOR_V5_1 or less.

SEE ALSO

- Section 13.2.3, “ls_a_license_case_sensitive”

8.2.11 LM_A_MT_HEARTBEAT (UNIX Only)

Type: `(int)`

Default: True

This flag applies to UNIX platforms only. If `True`, the automatic heartbeat mechanism is controlled via a dedicated thread in the FLEX*lm*-enabled application. The platform must support pthreads in order to implement a dedicated thread for heartbeat messages.

If `False`, automatic heartbeats are controlled in the application’s main thread using the SIGALRM signal.

SEE ALSO

- Chapter 12, “Heartbeats”

8.2.12 LM_A_PERIODIC_CALL

Type: Pointer to a function returning `int`. Return value not used.

Default: No periodic call

This function, if specified, will be called each LM_A_PERIODIC_COUNT times that lc_heartbeat() is called. lc_heartbeat() is called directly or automatically depending on the value of LM_A_CHECK_INTERVAL.

SEE ALSO

- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 6.4.12, “lc_heartbeat()”

8.2.13 LM_A_PERIODIC_COUNT

Type: `(int)`

Default: 0 (no PERIODIC_CALL)

This is the count of how many times lc_heartbeat() must be called before the function specified by LM_A_PERIODIC_CALL is called. lc_heartbeat() is called directly or automatically depending on the value of LM_A_CHECK_INTERVAL.

SEE ALSO

- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 6.4.12, “lc_heartbeat()”

8.2.14 LM_A_PLATFORM_OVERRIDE

Type: `(char *)`

Default: No override of the FLEX*lm* platform name

This string, if specified, overrides the current platform name. The name can be either a FLEX*lm* platform name or a vendor-defined name. This attribute is used in conjunction with the PLATFORMS= keyword on FEATURE/INCREMENT lines. The FLEX*lm* Release Notes contain the currently supported platforms and their associated FLEX*lm* platform names.

SEE ALSO

- Section 3.5.16, “PLATFORMS”
- FLEX*lm Release Notes* located in the `machind` directory of the FLEX*lm* SDK.

8.2.15 LM_A_RETRY_CHECKOUT

Type: `(int)`

Default: True

When True, checkouts that fail due to communications errors are automatically retried once. Sometimes this second attempt will succeed on networks with poor communications, but this makes failure take twice as long. This is default behavior in all of the FLEX*lm* APIs.

8.2.16 LM_A_USER_OVERRIDE

Type: `(char *)`

Default: No override of user name

This string, if specified, will be used to override the user name as derived from the UNIX password file. On Windows, the user name is set to the host name, but can be overridden with this attribute.

Note: This value cannot be changed after the initial connection to the vendor daemon.

8.2.17 LM_A_VENDOR_CALLBACK_DATA

Type `(void *)`

Default: `(void *) NULL`

This attribute allows you to set a pointer to vendor-defined data. The pointer is of type `void` regardless of the data type. This pointer is passed, as the last argument, to the following callback functions, thus making this data available to those functions.

- `LM_A_CHECKOUTFILTER_EX`
- `LM_A_USER_EXITCALL_EX`
- `LM_A_USER_RECONNECT_EX`
- `LM_A_USER_RECONNECT_DONE_EX`

8.3 Advanced Vendor Variables

The vendor variables in this section provide advanced capabilities beyond what is provided by the basic variables in Chapter 13, “Vendor Daemon.”

Table 8-3: Advanced Vendor Variables

Variable	Description
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_conn_timeout	Controls vendor daemon connection timeout.
ls_do_checkroot (UNIX Only)	Controls requirement for the vendor daemon’s residence on a real root filesystem.
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_use_all_feature_lines	Controls feature line succession.
ls_user_lockfile	Renames the vendor daemon locking mechanism.
ls_user_lock (Windows only)	Renames the vendor daemon locking mechanism

8.3.1 ls_borrow_in

```
void (*ls_borrow_in) (char ** borrowdata, int datasize) =
                        default_BorrowIn;
```

This callback function removes the record that matches `borrowdata` from the borrow-data cache. The license server 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:

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

  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 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_init` and `ls_borrow_out`.

SEE ALSO

- Section 8.3.2, “ls_borrow_init”
- Section 8.3.3, “ls_borrow_out”

8.3.2 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 license server 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 comprising that record. The first byte of the subsequent record immediately follows the last byte of the previous record.

The license server, at start up 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

- Section 8.3.1, “ls_borrow_in”
- Section 8.3.3, “ls_borrow_out”

8.3.3 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 license server 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

- Section 8.3.1, “ls_borrow_in”
- Section 8.3.2, “ls_borrow_init”

8.3.4 ls_borrow_return_early

```
(int) ls_borrow_return_early = 0;
```

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

This variable has two settings:

0	Don’t allow borrowed licenses to be returned early. This is the default setting.
1	Allow licenses to be returned early.

SEE ALSO

- Section 8.3.1, “ls_borrow_in”
- Section 8.3.2, “ls_borrow_init”
- Section 8.3.3, “ls_borrow_out”

8.3.5 ls_conn_timeout

```
(int) ls_conn_timeout = MASTER_WAIT;
            /* How long to wait for a connection */
```

`ls_conn_timeout` is the amount of time (in seconds) that vendor daemons will wait for connections from vendor daemons on other nodes when using redundant servers. It should normally not be changed.

8.3.6 ls_do_checkroot (UNIX Only)

```
(int) ls_do_checkroot = 0;
        /* Perform check that we are running on the real root */
```

To require that your vendor daemon be running on a file system which has its root directory as the “real” root directory of the disk, set this option. This prevents an end 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 he desired.

Theft by using `chroot` is considered to be an obscure, difficult kind of theft. The user has to have root permission, and setting up a phony `/` directory is a non-trivial task. It requires that the necessary parts of the OS from `/etc`, `/dev`, `/bin`, etc. be copied into this phony `/` directory and is an ongoing administrative hassle.

The check performed by `ls_do_checkroot` will fail on a diskless node. This prevents diskless nodes from acting as license servers. Macrovision does not recommend running license daemons on diskless nodes, but if you choose to support this, you will need to set `ls_do_checkroot` to 0.

For improved security, set `ls_do_checkroot` to 1. For minimization of confusion and support calls when your customers are running on diskless nodes, set `ls_do_checkroot` to 0.

8.3.7 ls_dump_send_data

```
(int) ls_dump_send_data = 0;
                /* Set to non-zero value for debug output */
```

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

8.3.8 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. 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 default setting is `0` (case insensitive).

8.3.9 ls_use_all_feature_lines

```
(int) ls_use_all_feature_lines = 0;
/* Use ALL copies of feature lines that are...
```

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 customer a FEATURE line with a count of 5, then upgraded them with a new line with a count of 7, they would now be able to use 12 licenses.

Also note that 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

- Section 3.5, “FEATURE /INCREMENT Lines”

8.3.10 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 machine 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`*.
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 pre-v8.2, use this mechanism for all daemons in the group.
  - If all vendor daemons involved in the synchronization are v8.2 or later, see Section 8.3.11, “ls_user_lock (Windows only).”
- UNIX Platforms
  - Use this mechanism regardless of vendor daemon version.

8.3.11 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 machine at one time. If you want 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 which 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 machine. 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 machine 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 to so that the set of lock names can be constructed at runtime, 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 via the `ls_user_lock` identifier:

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

The callback function can be included directly into `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 v8.2 or later, use this mechanism for all daemons in the group.

- If one or more vendor daemons involved in the synchronization are pre-v8.2, see Section 8.3.10, “ls_user_lockfile.”.

• UNIX Platforms

- This mechanism is not applicable; see Section 8.3.10, “ls_user_lockfile.”

8.4 Advanced License File Features

8.4.1 CAPACITY

```
CAPACITY
```

Optional keyword. The most common purpose of the CAPACITY keyword is to provide capacity based licensing. Capacity could be based on any measure of a machine’s resources. This technique effectively charges more for a more powerful system.

The CAPACITY keyword enables a checkout multiplier implemented internally in the licensed 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 2 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, check out 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 licensed application machine. 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 machines 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://cedar.intel.com/cgi-bin/ids.dll/content/content.jsp?cntKey=Generic+Editorial::xeon_cpu_counter&cntType=IDS_EDITORIAL

Chapter 9

License Models

9.1 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. Call your FLEX*lm* salesperson for a description of the additional types of licensing models that FLEX*lm* supports.

9.1.1 Limited Time, Uncounted Demos

This is the most popular method. 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 like:

```
FEATURE f1 corp 1.0 1-jan-2005 uncounted HOSTID=DEMO \
        SIGN=AB0CC0C16807
```

This indicates the expiration date and the fact that it's a demo license (node-locked to HOSTID=DEMO). The product is fully usable until January 1, 2005. FEATURE lines like this can be pre-printed 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-2005 uncounted HOSTID=DEMO SIGN=AB1CC0916A06
FEATURE f1 corp 1.0 1-feb-2005 uncounted HOSTID=DEMO SIGN=ABDCC0116A06
FEATURE f1 corp 1.0 1-mar-2005 uncounted HOSTID=DEMO SIGN=BBDCA0D151ED
FEATURE f1 corp 1.0 1-apr-2005 uncounted HOSTID=DEMO SIGN=BBDCB0E155F1
[...]
```

If the current date is February 1, 2005, then the salesperson would give an evaluator the third line, which expires in a month, March 1, 2005. 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-jan-2005 uncounted HOSTID=DEMO \
        SIGN=AB1CC0916A06
```

The FEATURE line must appear after the PACKAGE line to work correctly.

9.1.2 Limited Functionality Demos

FLEX*lm* does do some security checks to prevent users from setting system dates back. Though date-setback detection can be circumvented, most "honest users" (customers 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` struct that authorized the checkout. Put the expiration date (`CONFIG->date`) in a prominent place in the GUI so that the date-setback detection is more public.

PROVIDE AN INSISTENT REMINDER

If it is an expiring evaluation version, periodically do something annoying—perhaps a popup that appears every few minutes which encourages the user to purchase the product.

DISABLE SOME FUNCTIONALITY

A classic example 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

- Section 3.5, "FEATURE /INCREMENT Lines"
- Section 3.7, "PACKAGE Lines"
- Section 6.4.1, "lc_auth_data()"
- Section 7.2.3, "LM_CHECK_BADDATE"
- Section 13.2.2, "ls_a_check_baddate"

9.2 Lenient Licensing: Report Log and OVERDRAFT

More and more companies prefer licensing that does not deny usage, but bills customers for their usage.

9.2.1 FLEX*lm* Report Log File

A FLEX*lm* report log file (which is enabled with `lmswitchr` and/or an end-user options file REPORTLOG entry) provides a relatively secure method of tracking end-user usage. See the *FLEXlm End Users Guide* for more information about starting and managing a report log file. The report log file can be used for billing customers for their usage. A common method for doing this is to provide a FEATURE line with an OVERDRAFT. OVERDRAFT usage is logged to the REPORTLOG file, which is then read by FLEX*bill*, from which an invoice can be generated. FLEX*bill* is a separate product available from Macrovision.

ADVANTAGES

The advantages of this system include:

- The end user is not denied usage during peak usage periods (within limits).
- The vendor can gain additional revenue over traditional floating usage schemes.

A customer 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 SAM*report*. However, this does not mean that no tampering is possible. There are three conditions that must be considered:

- First, the customer may simply lose a file (either by accident or on purpose). Files are “ended” when a license server stops and starts or when an `lmreread` is performed. These sections can be lost without detecting a file modification, although the fact that a time period is missing *can* be detected.
- Second, 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.”
- Finally, a similar policy will be needed for files that have been altered.

9.2.2 OVERDRAFT Detection

Applications may want to inform users when they’re 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

- Section 3.5, “FEATURE /INCREMENT Lines”
- Section 7.3.21, “LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO”

9.3 Mobile Licensing

See the *FLEXlm Programmers Guide* for examples of licensing methods that will allow end users to use FLEX*lm*-licensed applications on computers not connected to a license server.

Chapter 10

The License Manager Daemon

The purpose of the license manager daemon, `lmgrd`, is to:

- Start and maintain all the vendor daemons listed in the VENDOR lines of the license file.
- Refer application checkout (or other) requests to the correct vendor daemon.

`lmgrd` is a standard component of FLEX*lm* that neither requires nor allows for vendor customization. The license manager daemon does allow the license file location and the server-to-server connection timeout interval to be set by the end user. These options are set by command-line arguments when starting `lmgrd`.

10.1 lmgrd Command-Line Syntax

`lmgrd` is the main daemon program for FLEX*lm*. When you invoke `lmgrd`, it looks for a license file which contains information about vendors and features.

The command-line syntax for `lmgrd` is:

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

where:

`-c` *`license_file_list`*	Use the specified license file(s).
`-l [+]`*`debug_log_path`*	Write 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.

`-2 -p`	Restricts usage of `lmdown`, `lmreread`, and `lmremove` to a FLEX*lm* administrator who is by default root. If there 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. If `-2 -p` is used when starting `lmgrd`, no user on Windows can shut down the license server with `lmdown`.
`-local`	Restricts the `lmdown` command to be run only from the same machine where `lmgrd` is running.
`-x lmdown`	Disable the `lmdown` command (no user can run `lmdown`). If `lmdown` is disabled, you will need to stop `lmgrd` via `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. (v4.0+ `lmgrd`)
`-x lmremove`	Disable the `lmremove` command (no user can run `lmremove`). (v4.0+ `lmgrd`)
`-z`	Run 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 `lmgrd` version number and copyright and exits.

`-help`	Displays usage information and exits.

Note: The license-file list 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`.

10.1.1 Starting lmgrd on UNIX Platforms

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

10.1.2 Starting lmgrd on Windows Platforms

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

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

The problem with running a server this way is that it occupies a window on the screen, and may be difficult to start and stop. On Windows, `lmgrd` can be installed as a service to allow it to be started and stopped through a user interface and run in the background.

To get `lmgrd` to run as a service, you need to "install" it. Two methods are available:

- Using the GUI application, LMTOOLS
- Using the command line utility, `installs.exe`

Both are located in the *`platform`* directory of your SDK installation. Using LMTOOLS to install `lmgrd` as a service is the recommended technique (see the *FLEXlm Programmers Guide*). If you prefer to use the `installs` utility, see Section 18.9, "Installing lmgrd as a Service Using installs."

10.2 License Server Configuration

FLEX*lm* supports:

- Single license server nodes
- Redundancy via a license-file list
- Three-server redundancy

If all the end user's data is on a single file server, then there is no need for redundant servers, and Macrovision recommends the use of a single server node for the FLEX*lm* daemons. If the end user's data is split among two or more server nodes and work is still possible when one of these nodes goes down or off the network, then multiple server nodes can be employed.

In all cases, an effort should be made to select stable systems as server nodes; in other words, do not pick systems that are frequently rebooted or shut down for one reason or another. Multiple server nodes can be any supported server nodes—it is not required that they be the same architecture or operating system.

FLEX*lm* supports two methods of redundancy: redundancy via a license-file list in the `LM_LICENSE_FILE` environment variable and a set of three redundant license servers.

See the *FLEXlm Programmers Guide* for more information on license servers and recommendations about configuring license server machines.

Chapter 11

Advanced FLEX*lm* Topics

Basic instructions for building your FLEX*lm* SDK, adding calls to the Simple or Trivial API into your application, and building your application can be found in the *FLEXlm Programmers Guide*. This chapter contains information about some advanced FLEX*lm* features that can be implemented using the FLEXible API and about FLEX*lm* security.

11.1 FLEX*lm* Example Applications

On both UNIX and Windows, the FLEX*lm* SDK contains the source for an example FLEXible API client application program called `lmflex.c`. `lmclient.c` is a small standalone program using the macro-based Trivial API and is a good place to start to learn how to integrate FLEX*lm* with your application. A function-based Trivial API example is provided in `ltclient.c`. A Simple API example program, `lmsimple.c`, is also available. The source to these four example programs is in the `machind` directory.

For Windows systems, the `machind` directory contains `lmwin.c`, the source for an example GUI application. `lmwin` uses Microsoft Visual C++ to build a slightly more complicated macro-based Trivial API example program to demonstrate more advanced options.

The `lmcrypt` and `makekey` programs can be used to generate licenses for your customers, or they can be used as examples of license generation programs. Source to the `lmcrypt` and `makekey` programs is in the `machind` directory. The `lmcrypt` and `makekey` programs generate the same signatures on all FLEX*lm*-supported platforms for all FLEX*lm* versions, thus allowing you to create licenses for any supported platform on any other supported platform.

FLEX*lm* SDKs also contain an `examples` directory at the top level of the SDK hierarchy. The `examples` directory contains example programs, which have been put in the SDK to illustrate how to perform various operations with FLEX*lm*. These programs are *not supported*, and Macrovision may not include them in future FLEX*lm* releases.

11.2 Lingering Licenses

A lingering license allows you to specify how long a license will remain checked out beyond either the call to lc_checkin() or program exit (whichever comes first). To use this feature, call lc_set_attr() before checking out the feature that should linger:

```
lc_set_attr(job, LM_A_LINGER, (LM_A_VAL_TYPE)x)
```

where *x* is the number of seconds to make the license linger.

In addition, the end user can specify a longer linger interval in the vendor daemon's options file, as such:

```
LINGER f1 100
```

The longer of the developer-specified and user-specified times will be used. The actual time of checkin will vary somewhat since the vendor daemon checks all lingering licenses once per minute. If, however, a new license request is made that would otherwise be denied, a check of the lingering licenses is made immediately to attempt to satisfy the new request. Linger is useful for programs that normally take under a minute to complete. Linger is generally useful only if `DUP_GROUP` is set in the license file or if *`dup_group`* is set in the call to lc_checkout().

SEE ALSO

- Section 7.3.11, "LM_A_LICENSE_FMT_VER"
- Section 3.5, "FEATURE /INCREMENT Lines"
- Section 6.4.3, "lc_checkout()"

11.3 FLEX*lm* and Multithreaded Applications

FLEX*lm* can be used in multithreaded applications as long as the same FLEX*lm* job is not referenced simultaneously in more than one thread. FLEX*lm* is safe for multithreaded applications, but FLEX*lm* functions are not re-entrant.

11.4 Multiple Jobs

In the FLEXible API, lc_new_job() enables applications to support more than one FLEX*lm* job in a single binary. Each job has a separate connection to a license server, as well as a independent set of job attributes. When a new job is created with lc_new_job(), all the FLEX*lm* attributes are set to defaults, and attributes can be set completely independently for this new job.

Multiple jobs may be desirable for the following reasons:

- If LM_LICENSE_FILE is a license-file list (colon-separated on UNIX and semi-colon separated on Windows) with more than one license server supporting features for the client, and if the application needs to check out more than one feature, it may be necessary to communicate with two license servers to check out the necessary licenses. This can be done only with multiple jobs, because a separate connection is required for each server.
- It may be convenient to have a single process manage licenses for other processes. It is usually convenient to manage each process's license as a separate job.
- lc_checkin() checks in all licenses for a given feature name. If the application needs to check in only some of the licenses, this can be done with multiple jobs, where groups of checkouts are done in separate jobs, and checked in separately from each job.

The first item can be important as an alternative way of supporting license server redundancy. Following is a program excerpt that illustrates how to support this:

```
LM_HANDLE *job1 = 0, *job2 = 0;
VENDORCODE code;
if (lc_new_job((LM_HANDLE *)0, lc_new_job_arg2, &code, &job1))
       /* error processing */ ;
set_all_my_attr(job1); /* do all necessary lc_set_attr() calls */
if (lc_checkout(job1, "f1", "1.0", 1, LM_CO_NOWAIT, &code,
                                          LM_DUP_NONE))
       /* error processing */ ;
/*     We checkout out one feature successfully, so we're
 *     connected to a server already. In order to connect to
 *     another server, we would need another job
 */
if (lc_checkout(job1, "f2", "1.0", 1, LM_CO_NOWAIT, &code,
     LM_DUP_NONE))
{
       if (lc_new_job(job1, lc_new_job_arg2, &code, &job2))
       {
```

```
            /* error processing */
            job2 = 0;
        }
        else
        {
            set_all_my_attr(job2);/* Reset attributes */
            if (lc_checkout(job2, "f2", "1.0", 1,
                 LM_CO_NOWAIT, &code, LM_DUP_NONE))
            {
                 /* error processing */ ;
            }
        }
}
/* application code here */
lc_checkin(job1, LM_CI_ALL_FEATURES, 0);
lc_free_job(job1);
if (job2 && job2 != job1)
{
        lc_checkin(job2, LM_CI_ALL_FEATURES, 0);
        lc_free_job(job2);
}
```

If the application is managing many jobs, you may want to free jobs with lc_free_job() to save memory. When doing so, make sure that you do not delete a job which still has a license checked out—this can result in a core dump.

Jobs can be found and managed using lc_first_job() and lc_next_job(), which are used to walk the list of jobs. Attributes for jobs are set and retrieved with lc_set_attr() and lc_get_attr().

SEE ALSO

- Section 6.4.10, “lc_free_job()”
- Section 6.4.11, “lc_get_attr()”
- Section 6.4.17, “lc_new_job()”
- Section 6.4.9, “lc_first_job()”
- Section 6.4.20, “lc_set_attr()”

11.5 FLEX*lock*

FLEX*lock* is available only on Windows.

If you are using FLEX*lock* and the FLEXible API and want to check out a feature, you must call the LM_USE_FLEXLOCK() macro before the call to the lc_checkout(). LM_USE_FLEXLOCK() can be used with the Trivial, Simple, or FLEXible API.

```
LM_USE_FLEXLOCK();
status = lc_checkout(job, feature, version, num_lic, flag,
                     code, dup_group)
```

CHECKING OUT MORE THAN ONE FEATURE WITH FLEX*LOCK* ENABLED

If you are using FLEX*lock* and your application checks out more than one feature, use the FLEXible API.

You will need a flag indicating that the first checkout was authorized by FLEX*lock*. In the following example code, the flag is called `flexlock_flag` and is initialized to 0.

After the first successful call to lc_checkout() returns, call:

```
CONFIG *conf;
extern int flexlock_flag;
conf = lc_auth_data(job, feature);
if (conf->idptr && (conf->idptr->type == HOSTID_FLEXLOCK))
{
    flexlock_flag = 1;
}
```

Before all subsequent calls to lc_checkout(), check the value of `flexlock_flag`. If `flexlock_flag` is set to `false,` proceed with the call:

```
if (flexlock_flag || (lc_checkout(...feature2...) == 0))
{
    /* feature2-enabled */
}
```

11.6 Security and FLEX*lm*

11.6.1 Counterfeit Resistant Option (CRO)

FLEX*lm* offers the Counterfeit Resistant Option (CRO). Without CRO, FLEX*lm* utilizes the standard FLEX*lm* license key which uses a proprietary, non-public-key digital signature method. CRO offers a standard public-key system which is recognized by the security community and recommended for US government work (with US government export approval). The system comes from Certicom (http://www.certicom.com) and uses elliptical curve cryptography. With CRO, the possibility of counterfeiting licenses becomes more remote.

If you are new to FLEX*lm* or considering adopting CRO, see the *FLEXlm Programmers Guide* for more information about implementing CRO.

11.6.2 Using lmstrip for Additional Security

The `lmstrip` utility is used to obfuscate FLEX*lm* external symbol names, FLEX*lm* internal symbol names, and vendor-specified symbol names in FLEX*lm*-enabled libraries and executables. Obfuscating these symbol names provides a level of security against reverse engineering your product. The `lmstrip` utility provides the following benefits.

- Adds security to UNIX FLEX*lm*-enabled application binaries.
- Provides additional security for FLEX*lm*-enabled licensing libraries.
- Allows multiple FLEX*lm*-enabled libraries, possibly employing different versions of FLEX*lm*, to be referenced from the same binary without symbol conflict.

This utility is located in the *`platform`* directory of the FLEX*lm* Software Development Kit. The source (shipped with UNIX platforms only) is located at `machind/lmstrip.c`.

UNIX USAGE

```
lmstrip filename [-f filename] [-e | -n] [-r] [-m]
        [-mapfile mapfilename] [strings...]
```

To display a list of all FLEX*lm* internal and external symbol names to be obfuscated:

```
lmstrip -l
```

where:

`filename`	Name of the file to process.
`-f` *`filename`*	Additional files to process.
`-e`	Do not obfuscate FLEX*lm* external symbol names.
`-n`	Do not obfuscate either FLEX*lm* internal or external symbol names.
`-r`	Obfuscates by replacing each symbol name with a string of random printable characters. Each new string is no more than six bytes. This is the default action.

`-m`	Creates a mapfile if it does not exist or use an existing mapfile. Default mapfile name is `lmstrip.map`. Forces randomized names to be the same across invocations.
`-mapfile` *`mapfile`*	Uses *`mapfile`* to override the default mapfile name.
`strings`	A list of vendor-specified symbol names in *`filename`* to be obfuscated.
`-l`	Displays a list of all FLEX*lm* internal and external symbol names to be obfuscated

WINDOWS USAGE

```
lmstrip [-e] [-i] [-v | -q]
        [-m[rw] [mapfile]] [-o string_file] filename
```

To display a list of all FLEX*lm* internal and external symbol names to be obfuscated:

```
lmstrip -l
```

To display a list of non-FLEX*lm* symbol names in *`filename:`*

```
lmstrip -p filename
```

where:

`-e`	Do not obfuscate FLEX*lm* external symbol names.
`-i`	Do not obfuscate FLEX*lm* internal symbol names.
`-v`	Verbose mode. In addition to standard status information, displays all symbol names, along with their new names, that have been obfuscated.
`-q`	Quiet mode. All output is suppressed
`-mr [`*`mapfile`*`]`	Reads mapfile *`mapfile`*. Forces randomized names to be the same across invocations. Default mapfile name is `lmstrip.map`.

`-mw [mapfile]`	Writes mapfile *`mapfile`*. This file contains a lookup table of obfuscated symbol names that can be used in later invocations. Default mapfile name is `lmstrip.map`.
`-mrw [mapfile]`	Reads and updates mapfile *`mapfile`*. Default mapfile name is `lmstrip.map`.
`-o string_file`	A file containing one or more vendor-specified symbol names to obfuscate in *`filename`*. One symbol name per line of the form: • *`+symbol`* Adds the symbol to the list of vendor-specified symbols to be obfuscated. • *`-symbol`* Removes the symbol as an obfuscation candidate. *`symbol`* can be either a FLEX*lm* symbol name or a vendor-specified symbol name.
`filename`	Name of the file to process.
`-p`	Displays the list of non-FLEX*lm* symbol names in *`filename`*. Used in conjunction with `-mr` *`mapfile`*, symbols that have already been obfuscated are suppressed from the list.
`-l`	Displays a list of all FLEX*lm* internal and external symbol names to be obfuscated

ADDING SECURITY TO UNIX APPLICATION BINARIES

When the UNIX `strip` command processes a dynamically linked binary, it leaves external symbol names intact in case they are referenced from a shared library. Using `lmstrip` on a dynamically linked binary adds more security because it obfuscates all FLEX*lm* symbol names that UNIX `strip` leaves intact.

If any external FLEX*lm* symbol is referenced from a shared library (which is very rare), use `lmstrip -e`, which leaves the FLEX*lm* external symbols names intact, but still obfuscates the internal FLEX*lm* symbol names.

Because symbol names do not appear in fully linked Windows binaries, this procedure is not needed on Windows platforms.

PROVIDING ADDITIONAL SECURITY FOR LICENSING LIBRARIES

UNIX scenario:

1. `ld -r my_routines.o liblmgr.a -o ofile.o`

   `ofile.o` now includes both the necessary FLEX*lm* functions referenced from *`my_routines`*`.o` and the contents of *`my_routines`*`.o`.
2. `lmstrip ofile.o` *`my_symbol_1 my_symbol_2 my_symbol_3`*

   This obfuscates the FLEX*lm* symbol names as well as the additional vendor-specified symbols.
3. You then ship `ofile.o` to your customers, knowing that the FLEX*lm* symbol names and the specified vendor symbol names are not detectable in the resulting object file.

Windows scenario:

1. `lib my_routines.lib lmgr.lib /out:ofile.lib`

   `ofile.lib` now includes the contents from both *`my_routines`*`.lib` and `lmgr.lib`.
2. `lmstrip -o` *`mystrings`* `ofile.lib`

   Where *`mystrings`* contains lines similar to the following:

   ```
   +my_symbol_1
   +my_symbol_2
   +my_symbol_3
   ```

   This obfuscates the FLEX*lm* symbol names as well as the additional symbol names specified in `mystrings`.
3. `lib ofile.lib`

   This re-sorts `ofile.lib` so that subsequent linking can resolve the newly obfuscated symbols.
4. You then ship `ofile.lib` to your customers, knowing that the FLEX*lm* symbol names as well as the vendor-specified symbol names are not detectable in the resulting library file.

LINKING WITH A LIBRARY THAT ALREADY USES FLEX*LM*

You can protect against symbol name conflict among multiple FLEX*lm*-enabled licensing libraries by using `lmstrip` on each individual licensing library. This technique obfuscates the FLEX*lm* symbol names locally to each library, eliminating FLEX*lm* symbol name conflict when two or more such libraries are linked into an application binary.

Follow the scenarios presented in “Providing Additional Security for Licensing Libraries”, above. Because the FLEX*lm* symbols are altered, the resulting object file or library can be linked with a library that already has references to FLEX*lm* symbol names, regardless of the FLEX*lm* version. Both coexist successfully.

Chapter 12

Heartbeats

The FLEX*lm* license server uses a TCP/IP port to communicate to the licensed application; it can detect when a connection to the licensed 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 via *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 alive and healthy. By default, heartbeats are sent and acknowledgments are received automatically by the FLEX*lm*-licensed application at a pre-determined 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.

Every time the license server receives a heartbeat, it immediately sends an acknowledgement message back to the application. This message doesn't get picked up by the application until it is time for the next heartbeat; this latency is, by default, 120 seconds. This means the FLEX*lm*-licensed application does not detect that the license server is down until the time has passed for two heartbeats to sent. If the license server is shut down, the next heartbeat succeeds, because the acknowledgement 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.

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

Heartbeats can be automatic or manual. The distinction being how the timing of heartbeat messages is managed and how reconnection attempts are recognized and handled. Table 12-1 summarizes these distinctions.

Table 12-1:Heartbeat Summary

	Automatic Heartbeats	**Manual Heartbeats**
Heartbeat Timing	Control of heartbeat timing is transparent via a dedicated thread created by FLEX*lm* 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 multi-threaded. • A dedicated heartbeat thread is automatically interrupted rather than the application.	• No multi-threaded requirement. • The application has total control of the timing of heartbeat messages.

12.1 Automatic Heartbeats

By default, regular heartbeats are automatically initiated once a license is checked out. Heartbeats are managed via a timer running as a separate thread created on behalf of the application. There are four aspects to automatic heartbeat behavior:

- **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 tried every 60 seconds for up to 5 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 retries, the application, by default, exits with the error message, "Lost License, cannot re-connect" (UNIX — stderr, WINDOWS—dialog box).

12.1.1 Controlling Automatic Heartbeat Behavior

Various aspects of automatic heartbeat default behavior can be overridden by setting the appropriate FLEXible API attributes via lc_set_attr(). Table 12-2 lists the attributes and the effect they have on automatic heartbeat behavior.

Table 12-2: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	

Table 12-2:Automatic Heartbeat Control

Attribute	Description
LM_A_CHECK_INTERVAL	Used to set the interval at which heartbeats are exchanged. Default is 120 seconds.
Reconnection Strategy	
LM_A_RETRY_COUNT	Used to adjust the number of reconnect attempts. If -1, the application retries forever. Default is 5 attempts.
LM_A_RETRY_INTERVAL	Used to adjust 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 vendor-defined exit handler. Default is for FLEX*lm* to call exit() system call on behalf of the application.

SEE ALSO

- Section 7.3.5, “LM_A_CHECK_INTERVAL”
- Section 7.3.16, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL”
- Section 7.3.18, “LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX”
- Section 7.3.19, “LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX”

- Section 7.3.20, “LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX”
- Section 8.2.11, “LM_A_MT_HEARTBEAT (UNIX Only)”

12.2 Manual Heartbeats

Manual heartbeats are used instead of automatic heartbeats if the application:

- Wants to be in control of the precise timing of heartbeats.
- Cannot tolerate the extra thread introduced by automatic heartbeats.
- Cannot be multithreaded.
- 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 sections.

To set up manual heartbeats, the application needs to

1. Turn off the automatic heartbeat timer before a license is checked out.

   For the FLEXible API:

   Set both the LM_A_CHECK_INTERVAL and LM_A_RETRY_INTERVAL attribute to -1, as follows:

   ```
   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);
   ```

   For Trivial or Simple:

   Use the LM_MANUAL_HEARTBEAT policy modifier in the checkout call.

2. Invoke lc_heartbeat() (for Trivial or Simple API use lt_heartbeat(), HEARTBEAT() or lp_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 (5 calls to the heartbeat routine), it causes the application to exit (via the standard exit() system call) with the error message, “Lost License, cannot re-connect” (UNIX — stderr, WINDOWS—dialog box).

SEE ALSO

- Section 4.2.4, “lt_heartbeat(), HEARTBEAT()”
- Section 5.2.4, “lp_heartbeat()”
- Section 6.4.12, “lc_heartbeat()”
- Section 7.2.1, “LM_MANUAL_HEARTBEAT”

12.2.1 Controlling Manual Heartbeat Behavior

Various aspects of manual heartbeat default behavior can be overridden by setting the appropriate FLEXible API attributes via lc_set_attr(). Table 12-3 lists the attributes and the effect they have on automatic heartbeat behavior.

Table 12-3: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 5 attempts.
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 vendor-defined exit handler. Default is for application exit.

SEE ALSO

- Section 7.3.16, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL”
- Section 7.3.18, “LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX”
- Section 7.3.19, “LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX”
- Section 7.3.20, “LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX”

12.2.2 Manual Heartbeat Example

The code sequence below demonstrates one way to implement manual heartbeats; it uses 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 by
   FLEXlm after five 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);
}
```

12.3 License Server-side Considerations

The license server waits 2 hours (default) to hear from the FLEX*lm* application before checking in its licenses.

- The `LM_A_TCP_TIMEOUT` attribute is used to adjust default timeout value for the TCP/IP port. 0 means no TCP/IP port timeout.
- `ls_minimum_user_timeout` in `lsvendor.c` adjusts general timeout (not just TCP/IP related). The end user can adjust this timeout value further with the TIMEOUT and TIMEOUTALL options file keywords, see the *FLEXlm End Users Guide* for more details.

Chapter 13

Vendor Daemon

The FLEX*lm* installation program on UNIX builds a vendor daemon (either a `demo` vendor daemon or your own, depending on your instructions). A `demo` vendor daemon is provided in your Windows installation, but you have to rebuild your Windows FLEX*lm* SDK to build your own vendor daemon, *`vendor`* or *`vendor.`*`exe`. Your vendor daemon can be customized via variables in `machind/lsvendor.c`, but changes to this file are normally neither suggested nor required.

13.1 Configuring Your Vendor Daemon

To configure your vendor daemon:

1. Edit `lm_code.h` to change the VENDOR_NAME field to your vendor daemon name.
2. Customize `lsvendor.c`, if necessary (not normally needed).
3. Build the FLEX*lm* SDK using `make` (UNIX) or `nmake` (Windows).

13.2 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.

SEE ALSO

- Section 8.3, “Advanced Vendor Variables”
- Section D.3, “Obsolete Vendor Daemon Variables”

13.2.1 ls_a_behavior_ver

```
(char *) ls_a_behavior_ver = 0; /* like LM_A_BEHAVIOR_VER */
```

This can be set to LM_BEHAVIOR_V*x*, where *x* is 2, 3, 4, 5, 5_1, 6, 7, 7_1, 8, 8_1, 8_2, 8_3, or 9_0. The default, 0, is LM_BEHAVIOR_CURRENT, which is V9_0 in v9.5.

SEE ALSO

- Section 8.2.1, “LM_A_BEHAVIOR_VER”

13.2.2 ls_a_check_baddate

```
(int) ls_a_check_baddate = 0; /* like LM_A_CHECK_BADDATE */
```

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

- Section 7.2.3, “LM_CHECK_BADDATE”
- Section 9.1.2, “Limited Functionality Demos”

13.2.3 ls_a_license_case_sensitive

```
(int) ls_a_license_case_sensitive = 0;
/* like LM_A_LICENSE_CASE_SENSITIVE */
```

If set to 1, licenses are case-sensitive. Default is 0, not case-sensitive.

SEE ALSO

- Section 8.2.10, “LM_A_LICENSE_CASE_SENSITIVE”

13.2.4 ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade

```
(int) ls_compare_vendor_on_increment = 0;
                    /* Compare vendor-defined */
(int) ls_compare_vendor_on_upgrade = 0;
                    /* Compare vendor-def fields */
```

If VENDOR_STRING is used in your license files, then the value of these two variables may need to be modified. If one is set, 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 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 0, then it is not included.

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

- The feature names match
- The 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

- Section 3.5.24, “VENDOR_STRING”
- Section 3.6, “UPGRADE Lines”

13.2.5 ls_daemon_periodic

```
(void) (*ls_daemon_periodic)() = 0;
/* Vendor-defined periodic call in daemon */
```

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 `.o` file for this routine is linked into your vendor daemon.

13.2.6 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 Section 13.2.10, “ls_outfilter.”

13.2.7 ls_infilter

```
extern LM_HANDLE * lm_job;
(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, via lc_set_errno(`lm_job`, *`errno`*), appropriately.

To obtain the parameters of the current checkin call, use the ls_get_attr() call described in Section 13.2.10, “ls_outfilter.”

13.2.8 ls_min_lmremove

```
(int) ls_min_lmremove = 120;
           /* Minimum amount of time (seconds) that a...
```

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

13.2.9 ls_minimum_user_timeout

```
(int) ls_minimum_user_timeout = 900;
           /* Minimum user inactivity timeout (seconds)
```

This is the minimum value (in seconds) that an end user can set the feature’s TIMEOUT value. An attempt to set a timeout less than `ls_minimum_timeout` will result in the minimum value being set. If `ls_minimum_user_timeout` is set to 0, then the user TIMEOUT option is disabled.

13.2.10 ls_outfilter

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

Note: Please contact Technical Support before using `ls_outfilter`. Callbacks in this area are rarely needed, and we’re happy to provide assistance when they are.

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 code appropriately with lc_set_errno().

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

```
ls_get_attr(attr, &value)
```

where:

attr	An attribute specified in `ls_attr.h`.
`(char *)` *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, etc. for use in your filtering function.

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

`(char *)` *feature*	Feature desired.
`(char *)` *num_lic*	Number of licenses.
`(char *)` *wait*	“Wait until available” flag. If set to `1`, the request is queued if a license is not available.
`(CLIENT_DATA *)` *who*	The user.
`(char *)` *version*	Version number of feature.
`(char *)` *dup_sel*	Duplicate license selection criteria.

`(char *) linger`	How long the license is to linger.
`(char *) sign`	Signature from FEATURE line.

RETURN

0 — OK, license checked out

<> 0 — Error

Note: ls_get_attr() can be used to retrieve all the parameters that ls_checkout() requires.

13.2.11 ls_show_vendor_def

```
(int) ls_show_vendor_def = 0;
                /* If non-zero, the vendor daemon will send...
```

Your client 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 wish for your users to be able to view it with `lmstat`, then set `ls_show_vendor_def` to 1.

13.2.12 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.

13.2.13 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.

13.2.14 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.

13.2.15 ls_vendor_msg

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

To add support for sending messages from your client code to the daemon (with lc_vsend()), initialize `ls_vendor_msg` with a pointer to your routine which processes the message and create the reply for the client. `ls_vendor_msg` is called with a single parameter—the character string sent by the client. 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.

SEE ALSO

- Section 6.4.23, “lc_vsend()”

Chapter 14

Composite Hostids

14.1 Overview

A composite hostid combines FLEX*lm* hostids together to provide a more secure hostid. A composite hostid is a hashed 12-character hexidecimal value formed by combining the values of one or more simple hostids types. It can be used anywhere a simple hostid is used: on a SERVER line or FEATURE line of a license file.

Incorporating a composite hostid into your license policy involves the following actions:

- Decide which hostid types are included in your composite hostid.
- Include code in your FLEX*lm*-enabled application to initialize the composite hostid.
- For served licenses, modify your vendor daemon to initialize the composite hostid.
- Provide a way for your end user to derive the composite hostid from their machine on which your product is installed.
- Issue licenses that specify the COMPOSITE= hostid type.

The subsequent sections below explain each of these actions in detail.

SEE ALSO

- Section 2.2, “Hostids for FLEXlm-Supported Machines”
- Section 3.2, “SERVER Lines”
- Section 3.5.10, “HOSTID”

14.2 Defining a Composite Hostid

A composite hostid is defined by a list of simple hostid types. As few or as many types can be incorporated in the list as is appropriate for your license policy. When FLEX*lm* validates the machine'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 hexidecimal value.

At license authentication time, FLEX*lm* compares the calculated value to that in the FEATURE or SERVER line, an exact match indicating a legitimate machine 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 machine as a legitimate host.

Consult Section 6.4.13, "lc_hostid()," 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.

14.3 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 comprise 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, anytime FLEXlm refers to HOSTID_COMPOSITE,
       during this job, it will be calculated based on both
       HOSTID_ETHER and HOSTID_USER*/
    /*...*/
}
```

Invoke lc_init_simple_composite() right 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 machine on which the application is installed. That means both composite hostid constituents, HOSTID_ETHER and HOSTID_USER, have to be defined and match that to which make up the hostid specified by the COMPOSITE= keyword.

A complete sample application incorporating a composite hostid is located in `examples\composite\lmcomplex.c`.

SEE ALSO

- Section 6.4.13, “lc_hostid()”
- Section 8.1.9, “lc_init_simple_composite()”

14.4 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 machine on which the license server is running. If a composite hostid is specified on the SERVER line, then the vendor daemon must validate it. Follow these steps to modify your vendor daemon with the code necessary for this validation.

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 comprise 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, anytime FLEXlm refers to HOSTID_COMPOSITE,
      during this job, it will be calculated based on both
      HOSTID_ETHER and HOSTID_USER*/
   /*...*/
}
```

   For your convenience, this routine is provided in the SDK distribution as `examples\composite\init_composite.c`.

6. Open *`platform`*`\makefile` in a text editor. This example uses a Windows makefile.
   - Add the vendor-defined hostid code to the list of `EXECS`. For this example, add `init_composite.obj`.

     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
```

   - Add `init_composite.obj` to the link line for `$(DAEMON)`.
7. Rebuild your vendor daemon and distribute it to your end user.

14.5 Creating a Composite Hostid Utility

Because the FLEX*lm* SDK utilities, `LMTOOLS`, `lmutil`, and `lmhostid` are not able to determine the composite hostid value, you need to provide a means for your end 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.

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 comprise 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 FLEXlm Composite HostID of this machine 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 end user.

14.6 Providing a Composite Hostid License

To take advantage of your composite hostid, you need to issue licenses to your end user that include the COMPOSITE= hostid type. This hostid type can be used anywhere a hostid is specified.

- For FEATURE lines in your license template:

  `HOSTID=COMPOSITE=`*`composite_hostid`*

- For SERVER lines in your license template:

  `COMPOSITE=`*`composite_hostid`*

where *`composite_hostid`* is a value your end user provides to you via the `lmcomposite` utility described in Section 14.5, “Creating a Composite Hostid Utility.”

SEE ALSO

- Section 3.2, “SERVER Lines”
- Section 3.5.10, “HOSTID”

Chapter 15

Vendor-Defined Hostid Types

15.1 Overview

You can use the FLEXible API to define your own hostid type. If you would like to discuss whether or not vendor-defined hostids are feasible for your application, you can contact Technical Support.

The FLEX*lm* SDK provides 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.

15.2 Editing Source Files

You must define your hostid type (for this example, we are using `vendor_hostid.c`), then make sure that the vendor daemon, FLEX*lm* license generators, and your client application can recognize and use your hostid type. Only `lmcrypt` and `makekey` can generate licenses with vendor-defined hostids.

1. Make a copy of your FLEX*lm* production SDK. Follow these instructions using the files in the duplicate SDK.
2. Copy `examples\vendor_hostid\vendor_hostid.c` to the `machind` directory.

3. View the file and find the `#define` statements. See `lmclient.h` for `HOSTID` and `LM_VENDOR_HOSTID` definitions.

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

extern LM_HANDLE *lm_job; /* This must be the current job! */

/* This example returns only 1 hostid */
#define VENDEF_ID_TYPE HOSTID_VENDOR+1
#define VENDEF_ID_LABEL "VDH"
#define VENDEF_ID "12345678"


/*
 * x_flexlm_gethostid() - Callback to get vendor-defined hostid.
 *(Sorry about all the windows types for this function...)
 */

HOSTID *
#ifdef PC
LM_CALLBACK_TYPE
#endif /* PC */
/*
 * IMPORTANT NOTE:  This function MUST call l_new_hostid() for
 *                  a hostid struct on each call.
 *                  If more than one hostid of a type is
 *                  found, then call l_new_hostid for each
 *                  and make into a list using the `next' field.
 */

x_flexlm_gethostid(idtype)
short idtype;
{
      HOSTID *h = l_new_hostid();

      memset(h, 0, sizeof(HOSTID));
      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);
      }
      return((HOSTID *) NULL);
}
```

```
void
x_flexlm_newid(id)
{
       HOSTID *id;
       LM_VENDOR_HOSTID h;

       memset(&h, 0, sizeof (h));
       h.label = VENDEF_ID_LABEL;
       h.hostid_num = VENDEF_ID_TYPE;
       h.case_sensitive = 0;
       h.get_vendor_id = x_flexlm_gethostid;
       if (lc_set_attr(lm_job, LM_A_VENDOR_ID_DECLARE,
              (LM_A_VAL_TYPE) &h))
       {
              lc_perror(lm_job, "LM_A_VENDOR_ID_DECLARE FAILED");
       }
}
```

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 `machind\makekey.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_HANDLE **) &lm_job) )
   {
         lc_perror(lm_job, "lc_init failed");
         exit(1);
   }

   x_flexlm_newid();
   ```

7. Open your client application source file in a text editor. In this example, we are using `machind\lmflex.c`.
   - Make the `lm_job` variable global by moving it before main().

     ```
     VENDORCODE code;
     LM_HANDLE *lm_job;

     void
     main()
     ```

   - 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();
     ```

8. Open *`platform`*`\makefile` in a text editor. This example uses a Windows makefile.
   - Add the vendor-defined hostid code to the list of EXECS. For this example, add `vendor_hostid.obj`.
   - 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
     ```

   - Add `vendor_hostid.obj` to the link line for `$(DAEMON)`, `makekey`, `lmcrypt`, and `lmflex`. For example, for `lmflex.exe`:

     ```
     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
```

9. Rebuild your duplicate FLEX*lm* SDK.

15.3 Test the Vendor-Defined Hostid

You will use the vendor daemon, license generator, and client application you just built 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. Start your license server pointing to this license file.
4. Run `lmflex`. You should be able to check out “f1.”
5. Exit `lmflex` and stop the license server.

15.4 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);
}
```

Chapter 16

Debugging Hints

16.1 Debugging Your Application Code

There are several issues to be aware of when debugging your FLEX*lm* integrated application. Some of these are described in this chapter.

- If you are experiencing problems on only one platform (or if you run on only a single platform), please check the appropriate platform-specific notes in Chapter 17, “UNIX Platform-Specific Notes,” or Chapter 18, “Windows Platform-Specific Notes.”
- On UNIX, the sleep(3), pclose(3), and system(3) calls often do not work with FLEX*lm*’s default use of SIGALRM. If you must use these calls, disable FLEX*lm* automatic heartbeats by setting LM_A_CHECK_INTERVAL and LM_A_RETRY_INTERVAL to -1 with lc_set_attr() and call lc_heartbeat() periodically.
- On UNIX, FLEX*lm* installs a handler for SIGPIPE and SIGALRM. If your application uses FLEX*lm* automatic heartbeats and forks/execs another process, these signals must be restored to the default before the fork/exec, and then re-restored in the parent process. See signal(3) for details. If you fail to do this, the child process will fail with a segmentation violation, since the signal handler will not exist in the child process. This is due to the fact that the child inherits the signal handler setting of the timer, but it does not inherit the signal handler code.
- If the daemon log file is missing, be sure that you are using Bourne shell syntax in the startup file. In particular, do not use `csh`-style redirection `>&` in one of the `rc` startup files.

If the FLEX*lm* timers are used to perform checking and/or reconnection, non-reentrant routines can possibly be called in the C run-time library. We have verified that the routines called by the timers are free of malloc/free re-entrancy problems, since these are detectable by Purify, but there may be other,

especially I/O or system routines which are not reentrant, but called by FLEX*lm*. The only way to be certain to avoid this problem would be to disable the automatic heartbeat mechanism and call lc_heartbeat() directly.

SEE ALSO

- Section 6.4.12, “lc_heartbeat()”

16.2 Solving Problems In The Field

The most important thing is to use lc_errstring(), lp_errstring(), or ERRSTRING() to present the correct error message to your user for diagnosis. Here are two common problems that occur in the field:

“License server does not support this feature”

This indicates that the client and servers are reading two different copies of the license file. This can be remedied by inserting a USE_SERVER line after the SERVER line in the license file (v5 or later).

“Encryption code in license file is inconsistent”

FLEX*lm* will report the (LM_BADCODE, -8) error when:

- The license file has been mis-typed when entered or changed since it was created.
- The seeds in your application, vendor daemon, and license generation program differ.

If you are beginning to integrate your application with FLEX*lm*, this error is usually the result of not building all the software components with the same encryption seeds. Check `lmcrypt.c`, `makekey.c`, `lsvendor.c`, and your application code carefully to ensure that they are all built with the same encryption seeds. If this is the case, you simply need to make sure that your application, `lmcrypt`, `makekey`, and your vendor daemon have all been rebuilt since the last time that you changed `lm_code.h`, and that there is only one `lm_code.h` file.

16.3 Multiple Vendors Using FLEX*lm* at a Single End-User Site

In the case where multiple software vendors install FLEX*lm*-based products at a single end user site, the potential for license file location conflicts arises. This section summarizes strategies that allow for a minimum of end user inconvenience.

There are basically two cases involved at an end user site when more than one software vendor installs products.

CASE 1: ALL PRODUCTS USE THE SAME LICENSE SERVER NODE(S)

In this case, there are three possible solutions:

- The end user can keep both license files separate, running one `lmgrd` with a license-file list containing both files.
- The end user can keep the license files separate, running two `lmgrd`s, one for each license file. There are no drawbacks to this approach, because the `lmgrd` processes require few system resources.

  When using two separate license files, make sure the port numbers are different, or leave them blank for FLEX*lm* to automatically find an open port.
- You can combine license files by taking the set of SERVER lines from any one license file, and add all the other lines (VENDOR, FEATURE, INCREMENT, PACKAGE, and UPGRADE lines) from all the license files. The combined license file can be located in the default location (`/usr/local/flexlm/licenses/license.dat` on UNIX platforms and `C:\flexlm\license.dat` on Windows) or in any convenient location (with the end user using the `LM_LICENSE_FILE` environment variable or license finder), or multiple copies can be located at fixed locations as required by the various software vendors. The user should leave a symbolic link to the original license file in the locations where each software package expects to find its license file.

In practice, sites that have experienced system administrators often prefer to combine license files. However, sites with relatively inexperienced users and no system administrator usually do better leaving the files separate.

CASE 2: PRODUCTS USE DIFFERENT LICENSE SERVER NODE(S)

In this case, separate license files will be required, one for each distinct set of license servers. The license files can then be installed in convenient locations, and the user's `LM_LICENSE_FILE` environment variable would be set as follows.

```
setenv LM_LICENSE_FILE lic_path1:lic_path2:....:lic_pathn
```

When products from different vendors use different versions of FLEX*lm*, always use the latest versions of `lmgrd` and the `lmutil` utilities.

The latest version of `lmgrd` will always support any FLEX*lm* license. The end user has to find out which `lmgrd` at their site is the latest version. This can be done using `lmgrd -v` to get the version. If an earlier version of `lmgrd` is used than the vendor daemon, then various errors may occur, especially "Vendor daemon can't talk to `lmgrd` (invalid returned data from license server)."

16.4 FLEX*lm* Version Compatibility

When an end user has licensed products that incorporate various versions of FLEX*lm*, care must be taken to insure that the correct versions of `lmgrd` and the FLEX*lm* utilities are used. The rules about FLEX*lm* version compatibility are summarized as:

Version of `lmutil`/LMTOOLS must be >=

Version of `lmgrd`, which must be >=

Version of vendor daemon, which must be >=

Version of FLEX*lm*-licensed application, which must be >=

Version of license file format

Except for the license file, use `lmver` to discover the version of all these components. For the vendor daemon, `lmgrd`, and `lmutil`, you can also use the `-v` argument to print the version.

Chapter 17

UNIX Platform-Specific Notes

On UNIX, automatic heartbeats are performed via multithreading by default on most UNIX platforms. Applications on platforms that support pthreads must link in the pthreads library. On all platforms that support pthreads, this is done by appending `-lpthread` (`-lpthreads` on AIX) to the link line. If an application cannot use the pthreads library, call one of the functions that implements manual heartbeats and link in `lm_nomt.o` *before* `liblmgr.a`, thereby disabling automatic multithreaded heartbeats.

17.1 Macintosh OS X

The FLEX*lm* SDK on Macintosh OS X is accessed from the command line.

17.2 Solaris

The default Solaris TCP port delay of four minutes may cause problems when restarting a license server, especially a license server that is using a hard-coded port number (for example, each of the three-server redundant servers) rather than selecting from a set of default port numbers. If you have a problem restarting license servers on Solaris, a root user can reset the port delay to 2.4 seconds (2400 milliseconds). As a long-term solution, we recommend that one of the following lines be put in a boot script because the port delay is reset every time the machine reboots:

Solaris 2.6-:

```
/usr/sbin/ndd -set /dev/tcp tcp_close_wait_interval 2400
```

Solaris 2.7+:

```
/usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 2400
```

17.3 Hewlett Packard

The `/dev/lan0` device must be readable to obtain an ethernet hostid. The `uname -i` hostid is preferable for this reason, and because ethernet is not always present.

In v2.4, `/dev/lan0` must have read and write permissions for everyone. Ethernet and FDDI are known to be supported devices, although earlier versions of HP-UX had a bug with FDDI as hostid.

17.4 IBM

On RS/6000, `lmgrd` cannot be started in `/etc/rc` because on that OS the TCP/IP networking is started after `/etc/rc` is run. IBM has recommended that this be performed in the `/etc/inittab` file. Add a line like the following to `/etc/inittab` after the lines which start networking:

```
rclocal:2:wait:/etc/rc.local > /dev/console 2>&1
```

IBM changed the system call that returns the node id (`uname`) several times; most recently, in AIX 3.1, the low-order decimal digit of the machine serial number was left off. The AIX 3003 version has a corrected system call which returns the entire serial number. This means that the hostid of your customer's RS/6000 system *can change* when they upgrade OS revisions. We know of no workaround other than to re-issue licenses.

We believe that this condition stabilized in AIX v3.1.

On UNIX, automatic heartbeats are performed via multithreading by default on most UNIX platforms. Applications on platforms that support pthreads must link in the pthreads library. On AIX, this is done by appending `-lpthreads` to the link line.

17.5 Linux

If you are having difficulties building the FLEX*lm* SDK on your Linux platform, make sure that you have installed the correct platform-specific FLEX*lm* file (we provide three):

Red Hat Linux 5.2 or higher	`i86_g2.tar`
Red Hat Linux 6.2 or higher	`i86_r6.tar` `alpha_r6.tar`
Red Hat Linux 7.0 or higher	`it64_lr2.tar`

Macrovision has seen the following three types of problems with the Linux platform:

- Incompatible executables

  Executables built and linked on Redhat v4 are not fully forward compatible with Redhat 5 or 6. They may start to run but may later crash with indecipherable causes.

- Incompatible object files

  Object files (`*.o`) created on Redhat v4 or v5 and moved to a Redhat v6 system will not link correctly on Redhat v6. If all the object files are fully linked on Redhat v5, the v5 executable will run fine on Redhat v6.

- Unexplained problems

  We have customers that have reportedly not been able to build or run the FLEX*lm* kit for Redhat v6 despite `uname -a` indicating that these customers are using the same Redhat v6 that Macrovision uses to build FLEX*lm*. In these rare cases, we have not been able to resolve the situation and are not aware of what causes the problem.

17.6 SGI

SGI has a variety of CPUs, operating systems, and compiler switches that are mutually incompatible. To explain, it's useful to first understand the different CPUs, operating systems, and switches:

OPERATING SYSTEMS

IRIX 6.5+	64-bit OS, supports 64- and 32-bit applications.

MIPS CHIPS

MIPS1	First MIPS chip. The chip itself is no longer supported by SGI, but it's possible to generate binaries that run on this chip. R1000 systems(?).
MIPS2	Not much is known about MIPS2, and it's not relevant anyway.
MIPS3	32- and 64-bit binaries. R4000 and R6000 systems.
MIPS4	Improved 64-bit support. R8000 and R10000 systems.

COMPILER SWITCHES ON IRIX 6.5+

`-o32`	It is the “old 32-bit object” format.
`-n32`	Native to IRIX 6.5+, it is the “new” 32-bit format.
`-64`	Native to IRIX 6.5+; 64-bit.

OTHER COMPILER SWITCHES

`-xgot`	If your application exceeds 64,000 global variables, you must compile and link with objects that have this flag. if you need this, use the libraries with the `_xgot` suffix.

We provide two SGI directories:

`sgi32_u6`	32-bit IRIX 6.5+. `liblmgr.a` (`-n32`) or `liblmgr_o32.a` (`-o32`). MIPS3 libraries run on MIPS4 systems. Requires FLEX*lm* sgi vendor key.
`sgi64_u6`	64-bit (`-64`) IRIX 6.5+. `liblmgr.a`. MIPS3 libraries run on MIPS4 systems. Requires FLEX*lm* sgi64 vendor key.

FLEX*LM* VENDOR KEYS FOR SGI

sgi	All SGI 32-bit applications, including `sgi32_u6`.
sgi64	All SGI 64-bit applications, including `sgi64_u6`.

SGI “ORIGIN” SYSTEMS

These “modular” systems can have more than one hostid. `lmhostid` will report all the hostids for these systems. *A license should be generated for only one of these hostids.*

17.7 SCO

Part of the `install_flexlm.ftp` install scripts may fail on SCO systems. It is not difficult to install without the scripts. Edit the `machind/lm_code.h` file to put in the correct `VENDOR_KEY`s (obtained from Macrovision) and `LM_SEED`s (32-bit numbers you make up that make license files unique) and `VENDOR_NAME`. Then, if it's not an evaluation copy, in the `sco_u3` directory, edit the makefile from `DAEMON = demo`, replacing `demo` with your vendor daemon name. Then type `make` in the `sco_u3` directory.

Chapter 18

Windows Platform-Specific Notes

FLEX*lm* supports the Windows platforms using two sets of libraries (`lmgr.lib` and `lmgr9a.dll`).

18.1 Supported C Compilers

The FLEX*lm* client library on Windows is implemented as a static library or a DLL. The DLL can interface with almost any compiler, but is less secure. However, to build the FLEX*lm* SDK, including your vendor daemon, on a Windows system, FLEX*lm* supports only the Microsoft Visual C++ compiler 5.0 or greater.

18.2 Using Languages Other Than C

FLEX*lm* provides a function-based API that is designed for non-C languages such as Visual Basic . See Chapter 4, “Trivial API,” for more information.

18.3 Linking to your Program

FLEX*lm* can be linked into your application in three ways:

- Linking statically with a FLEX*lm* library that was built with the static C Runtime Library (recommended)

  The static FLEX*lm* library is compiled with Microsoft Visual C++ 5.0, 32-bit, with multithreading enabled, static C Runtime Library (/MT). The static library is named `lmgr.lib`.

- Linking statically with a FLEX*lm* library that uses the C Runtime Library as a DLL.

  Macrovision also provides a library compiled with /MD, multithreaded, using the C Runtime Library as a DLL, called `lmgr_md.lib`.

- Linking dynamically with the FLEX*lm* DLL (less secure)

  The DLL version is called `lmgr9a.dll` with its associated import library `lmgr9a.lib`. The `lmgr9a.dll` library is built using the multithreaded statically linked C Runtime Library. Use the FLEXible API for enhanced DLL security.

  If it is necessary to use the FLEX*lm* DLL, please send email to support@macrovision.com, to get suggested enhancements to improve the security of your application.

If your application is a DLL and the FLEX*lm* library is linked into this DLL, then you need to set one special attribute to allow the Windows context to be properly set. See Section 7.3.24, “LM_A_WINDOWS_MODULE_HANDLE (Windows only).”

18.4 Windows Terminal Server Support

FLEX*lm* detects when a node-locked uncounted license is running under Windows Terminal Server. To run on Terminal Server client machines, the license is node-locked to the Terminal Server machine and the TS_OK keyword is added to the license. For example:

```
FEATURE f1 xyzd 1.00 1-jan-2005 uncounted HOSTID=ts_hostid \
        TS_OK SIGN=sign
```

Without the TS_OK keyword on a feature line, a user running on a Terminal Server client machine will be denied use of a license. In each of the following cases, application A is installed on a Terminal Server machine with its license locked to Terminal Server machine’s hostid.

Case 1: Users can run application A on the terminal server machine by sitting at the terminal server machine, whether or not the feature line contains the TS_OK keyword. The license is checked out as though it was a regular node-locked license.

Case 2: If the feature line does not contain the TS_OK keyword and a user tries to run application A from a Terminal Server Client window, an error is generated (License fails to checkout, -103).

Case 3: If the feature line contains the TS_OK keyword and one or more users try to run application A from a Terminal Server Client window, the license is checked out and the application can run. If the feature line contains the TS_OK keyword, FLEX*lm* does not provide a way to limit the number of clients that can use this node-locked license.

Counted licenses behave on a Terminal Server Client as they do on any other machines on a network.

18.5 FLEX*lm* Callback Routines

The FLEX*lm* API supports application callbacks on various events such as lost of license and hostid acquisition. Like all Windows SDK standard callback routines, FLEX*lm* application callback routines need special attention depending upon the environment that you are using. The following code segments from the sample program demonstrates how this should be done:

```
void LM_CALLBACK_TYPE Quit(char * feature)
```

18.6 FLEX*lm* exit() Callback

The default operation of FLEX*lm* when the connection to the server is lost is to try five times and then exit the program.

18.7 FLEX*id* Hardware Hostids (Dongles)

A FLEX*id* is a parallel or USB dongle that is used to provide a hardware-based, moveable hostid. FLEX*id*s are manufactured for Macrovision and are purchased directly from Macrovision by vendors to sell to their customers. A FLEX*id* hostid has the form `FLEXID=id_string` where *`id_string`* indicates the FLEX*id* serial number.

Macrovision provides an installer that installs the appropriate OS drivers for all currently available FLEX*id*s, available in the *`platform`* folder of the FLEX*lm* SDK and are also available to vendors from http://www.globes.com, the Macrovision support web site. Macrovision recommends that you provide the installer to your customer and with instructions to install all of the FLEX*id* drivers. Consult `FLEXid_README.pdf` in the `machind` directory for specific installation instructions.

SEE ALSO

- Table 6-11 for complete FLEX*id* hostid support

18.8 FLEX*lm* Environment Variables

Many aspects of FLEX*lm* can be controlled using environment variables. Environment variables can be set either in the traditional way or as an entry in the registry. The *FLEXlm End Users Guide* contains a listing of all FLEX*lm* environment variables.

A FLEX*lm* component looks for an environment variable, it first looks to the program's environment variables. If it does not find it, it then looks into the registry in `HKEY_LOCAL_MACHINE\SOFTWARE\FLEXlm License Manager\`*`env_var`* where *`env_var`* is the environment variable. This is especially useful for setting the license file (if not using the default) using `LM_LICENSE_FILE`.

The FLEX*lm* license manager (`lmgrd.exe` and the vendor daemon) can also use registry values when they are started as a SERVICE. The values they use are in a sub-key in the above FLEX*lm* License Manager key. The following code snippet taken from `installs.c` shows how to create registry entries for the server programs.

```
/* next write registry entries */
/* Update the registry */
/* Try creating/opening the registry key */
if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE", 0, KEY_WRITE,
         &hcpl) == ERROR_SUCCESS)
{
    HKEY happ;
    DWORD dwDisp;
    char new_name[120];
    sprintf(new_name,"FLEXlm License Manager\\%s",
            Service_Name);
    if (RegCreateKeyEx(hcpl, new_name, 0, "",
                       REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL,
                       &happ, &dwDisp) == ERROR_SUCCESS)
    {
         RegSetValueEx(happ, "Lmgrd", 0, REG_SZ, Lmgrd_Path,
                       strlen(Lmgrd_Path));
         RegSetValueEx(happ, "LMGRD_LOG_FILE", 0, REG_SZ,
                       Log_File_Path, strlen(Log_File_Path));
         RegSetValueEx(happ, "License", 0, REG_SZ,
                       License_Path, strlen(License_Path));
         RegSetValueEx(happ, "Service", 0, REG_SZ,
                       Service_Name, strlen(Service_Name));
         /* Finished with keys */
         RegCloseKey(happ);
    }
    RegCloseKey(hcpl);
}
```

18.9 Installing lmgrd as a Service Using installs

To install `lmgrd` as a service, use the facility provided in LMTOOLS or use the `installs.exe` utility provided by FLEX*lm* in the *`platform`* directory. `installs.exe` can be used as a command line utility to manually install the service, or can be embedded in your end-user installation procedure to automatically install the service on behalf of the user at installation time.

Usage is:

```
installs  -c license_file_path \
          -e lmgrd_location \
          -l log_file_path \
          -n service_name \
          [-k lmgrd parameters]
```

To remove lmgrd as a service:

```
installs -r -n service_name
```

where:

`-e` *`lmgrd_location`*	Location of lmgrd.exe on your system. Path names with embedded spaces need to be enclosed in double quotes.
`-c` *`license_file_path`*	Location of the license file. Path names with embedded spaces need to be enclosed in double quotes.
`-l` *`log_file_path`*	Location for the debug log file. Path names with embedded spaces need to be enclosed in double quotes.
`-n` *`service_name`*	The service name. If not specified, `FLEXlm License Manager` is used by default. Service names with embedded spaces need to be enclosed in double quotes.

`-k` *`lmgrd parameters`*

Used to pass additional command line arguments to `lmgrd` each time it is started as a service. A string of parameters with embedded spaces needs to be enclosed in double quotes. One or more of the following can be specified:

- `-local`
  Restricts the `lmdown` command to be run only from the same machine where `lmgrd` is running.
- `-options` *`options_file_path`*
  Specifies the full path to the `lmgrd` options file. This file is used to configure the Microsoft Windows event-logging service for `lmgrd` events. If this parameter is omitted, `lmgrd`, by default, looks for a file called `lmgrd.opt` located in the same directory as the `lmgrd` binary.
- `-x lmdown`
  Disables the `lmdown` command (no user can run `lmdown`).
- `-x lmremove`
  Disables the `lmremove` command (no user can run `lmremove`).

See Section 10.1, “lmgrd Command-Line Syntax,” for more information on these three arguments. Other `lmgrd` parameters are not supported here.

`-r`

Remove *service_name* service.

After `installs.exe` is run successfully, `lmgrd` is installed as a Windows service and will be started automatically each time your system is booted. If you wish to customize the `installs` program, the source code is included in the file `machind\installs.c`.

EXAMPLES

- To install lmgrd as a service:

```
installs -e "C:\Program Files\flexlm\i86_n3\lmgrd" \
         -c C:\mylicenses\sample.lic \
         -l C:\mylogfiles\sample.dl \
         -n "Myvendor License Manager"
```

  where:

  - `"C:\Program Files\flexlm\i86_n3\lmgrd"` is the location of the lmgrd executable. (Double quotes are required around the name since there is an embedded space in one of the folder names.)
  - `C:\mylicenses\sample.lic` is the full path name for the license file.
  - `C:\mylogfiles\sample.dl` is the full path name for the debug log file.
  - `"Myvendor License Manager"` is the service name. (Double quotes are required since the service name is composed of multiple words.)

- To remove the service named “Myvendor License Manager” installed in the previous example:

```
installs -r -n "Myvendor License Manager"
```

- To install lmgrd as a service and to configure the lmgrd startup parameter, `-x lmremove`:

```
installs -e "C:\Program Files\flexlm\i86_n3\lmgrd" \
         -c C:\mylicenses\sample.lic \
         -l C:\mylogfiles\sample.dl \
         -n "Myvendor License Manager" \
         -k "-x lmremove"
```

  Now, each time lmgrd starts up, it will be invoked with the `-x lmremove` parameter.

Appendix A

Industry-Standard Licensing APIs

FLEX*lm* offers the most widely used licensing API available—the FLEX*lm* API, which is used by over 1500 software vendors worldwide. However, there has been much effort expended in the search for a “standard” licensing API.

FLEX*lm* offers the vendor the choice of six standard APIs:

- FLEX*lm* Trivial API
- FLEX*lm* Simple API
- FLEX*lm* FLEXible API
- FLEX*lm* Java API
- FLEX*lm* Visual Basic API
- LSAPI (a proposed standard)

FLEX*lm* is the only licensing system available which supports all six APIs.

A.1 The FLEX*lm* Trivial and Simple APIs

These APIs are suitable for most applications, and are robust and easy to implement. See Chapter 4, “Trivial API,” and Chapter 5, “Simple API,” for complete information on these two APIs.

A.2 The FLEX*lm* FLEXible API

The FLEXible API has evolved since 1988, with the input of most of the major software vendors in the UNIX software industry. The goal of the FLEXible API is to give you your choice of licensing models in an easy to implement, robust package. The FLEXible API is documented in Chapter 6, “FLEXible API.”

A.3 LSAPI v1.1

The LSAPI interface, a licensing API first proposed in May, 1992, was designed by a consortium of software vendors with participation from several licensing system vendors. The main "claim to fame" of this interface is that it attempts to provide a solution whereby the end user can choose the license server product from the licensing system vendor of their own choice. While the LSAPI seems to be a simple API, it hides the fact that your code will increase in complexity in order to solve the problem of the replaceable license server, (since both the license server and the licensing system library are, in theory, replaceable by the end user, any security *must* be built into your code, *independent* of the license server). The complexity is exposed to you in the "challenge mechanism," which is a standard authentication technique known as "handshaking."

Note: If you are considering using LSAPI in your product, you should read U.S. patent #5,375,206 issued to HP, and understand its implications.

LSAPI has several significant drawbacks compared to the FLEX*lm* APIs. In addition, Macrovision believes that the stated goal of license server independence cannot be met by the current version of the LSAPI spec (see last point below). Some of the drawbacks of LSAPI compared to the native FLEX*lm* APIs are:

- Unreasonable error reporting (only a total of 14 error codes.)
- No ability for the vendor to support license queueing.
- No vendor-specific checkout filtering.
- New hostid types are not definable by the software vendor.
- No provision to pass messages between the client and license server.
- No way to get license status without doing I/O to the license server.
- No way to support a node-locked license without a license server.
- No way to retrieve information about the licensing policy.
- No way to ship a vendor-neutral license. This means that, in order to accomplish the stated goal of allowing your end user to select the licensing system from the vendor of their choice, you would have to provide licenses in the format required by *each and every* license system which your customer might want to choose. In practice, what this means is that you would need to build and test with every possible licensing system.

Note: You cannot mix LSAPI calls with the native FLEX*lm* API calls.

A.3.1 Data Types for All Calls

(LS_ULONG)	(unsigned long)
(LS_STATUS_CODE)	(unsigned long)
(LS_STR)	(char)
(LS_CHALLENGE)	(structure)
(LS_CHALLENGE_FLEXLM)	(structure)
(LS_HANDLE)	(unsigned long)
(LS_VOID)	(void)

A.4 LSAPI General Calls

(LS_STATUS_CODE) LSEnumProviders((LS_ULONG) *Index*, (LS_STR) **Buffer*)	List providers of licensing service.
(LS_STATUS_CODE) LSGetMessage((LS_HANDLE) *Handle* (LS_STATUS_CODE) *Value*, (LS_STR) **Buffer*, (LS_ULONG) *BufferSize*)	Get message text from licensing system.
(LS_STATUS_CODE) LSQuery((LS_HANDLE) *Handle*, (LS_ULONG) *Information*, (LS_VOID) **InfoBuffer*, (LS_ULONG) *BufferSize*, (LS_ULONG) **ActualBufferSize*)	Query license information.
(LS_STATUS_CODE) LSRelease((LS_HANDLE) *Handle*, (LS_ULONG) *TotUnitsConsumed*, (LS_STR) **LogComment*)	Release license.

`(LS_STATUS_CODE) LSUpdate((LS_HANDLE) Handle, (LS_ULONG) TotUnitsConsumed, (LS_ULONG) TotUnitsReserved, (LS_STR) *LogComment, (LS_CHALLENGE) *lpChallenge, (LS_ULONG) *TotUnitsGranted)`	Update license status.
`(LS_STATUS_CODE) LSRequest((LS_STR) *LicenseSystem, (LS_STR) *PublisherName, (LS_STR) *ProductName, (LS_STR) *Version, (LS_ULONG) TotUnitsReserved, (LS_STR) *LogComment, (LS_CHALLENGE) *Challenge, (LS_ULONG) *TotUnitsGranted, (LS_HANDLE) *Handle)`	Request license.

Note: The challenge in your first LSRequest() call must be of type LS_CHALLENGE_FLEXLM, which is a FLEX*lm* vendor-specific challenge mechanism. Challenge should be setup as in the following code example before calling LSRequest():

```
LS_CHALLENGE_FLEXLM *Challenge;
LM_CODE(vendor_code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
        VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
...
Challenge->Protocol = LS_FLEXLM_PROTOCOL;
strcpy( Challenge->ChallengeData.VendorName, VENDOR_NAME);
Challenge->ChallengeData.VendorCode = vendor_code;
Challenge->Size = sizeof(*Challenge);
...
LSRequest( ...., (LS_CHALLENGE *)Challenge, ...);
```

For more details on the LSAPI interface, see the “License Service Application Programming Interface, API Specification v1.1,” or contact Microsoft via e-mail at lsapi@microsoft.com, or Dave Berry, Microsoft Developer Relations, 1 Microsoft Way, 4/2, Redmond, WA 98052-6399.

Remember, you cannot mix LSAPI and native FLEX*lm* calls in a single application. The license servers can support a mix of applications which use either native FLEX*lm* or LSAPI, but a single executable must use either native FLEX*lm* or LSAPI.

Appendix B

FLEXlm Parameter Limits

Limitations such as string lengths are listed here. Items that are unlimited are also listed for clarification.

B.1 FLEX*lm* Parameter General Information

B.1.1 Internationalization Support

FLEX*lm* 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

- The chapter entitled “Internationalization Support in FLEX*lm*” in the *FLEXlm Programmers Guide*.

B.1.2 Path Name Formats

FLEX*lm* 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 FLEX*lm* component is limited to those formats supported by the particular operating system on which the FLEX*lm* component resides.

B.2 License File Limits

The limits on names for the major parameters employed in the FLEX*lm* license file are:

Max. number of VENDOR lines	Unlimited
Vendor daemon name length	10 bytes
Host name length	64 bytes
Max. number of FEATURE/INCREMENT/ UPGRADE lines	Unlimited
FEATURE/INCREMENT/ UPGRADE/PACKAGE line length	2048 bytes
FEATURE name length	30 bytes
Version (including date-based version)	10 characters, in floating point format, e.g., `123.4567`, `2.10`, `2005.12`
Latest expiration date	31-dec-9999 (but we recommend using “permanent” instead)
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 2,147,483,646
USER_BASED=*n*	1 through 2,147,483,647
MINIMUM=*n*	1 through 32767
BORROW=*n*	1 through 2,147,483,647
VENDOR_STRING="*string*"	64 bytes
Other optional FEATURE attributes (e.g., NOTICE=)	Limited only by the total length of the FEATURE line.

B.3 Decimal Format License Limits

Max readable length that can be converted to decimal	Approximately 100 bytes. Because ASCII text becomes much larger in decimal format, a FEATURE line of 100 characters is unreadable and more prone to data entry errors in decimal format.
Types of licenses that can be converted to decimal	Everything but PACKAGE and FEATURESET lines.
Types of FEATURE names that can be converted to decimal	Only officially supported FEATURE names. In particular, “-” (hyphen) cannot be converted.

B.4 End-User Options File Limits

The line length limit is the same as the FEATURE line length (2048 characters). There are no other string size limitations on anything in this file. Note that GROUPs can be made arbitrarily large by listing the GROUP more than once—FLEX*lm* concatenates such entries.

B.5 lc_set_attr() limits

LM_A_DISPLAY_OVERRIDE	32 bytes
LM_A_HOSTNAME_OVERRIDE	64 bytes
LM_A_USERNAME	20 bytes
LM_A_CHECKOUT_DATA	32 bytes
LM_A_CHECK_INTERVAL	>20 seconds

B.6 Other API Limits

Maximum length for entire specification of vendor-defined hostid: *`LABEL=hostid`*	41 bytes
Maximum number of licenses in one lc_checkout() request	9999
Maximum long error message length	1024 bytes (length of string returned from lc_errstring())

B.7 Vendor Daemon Limits

NUMBER OF APPLICATIONS PER VENDOR DAEMON

When using TCP/IP ports, each FLEX*lm*-enabled application connected to a vendor daemon uses one or more sockets. The number of sockets any one FLEX*lm*-enabled application requires is dependant on FLEX*lm* 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 (`lmgrd` and the vendor daemon together) is slightly larger than the total number needed by the FLEX*lm*-enabled applications. When using UDP, there is no limit to the number of applications per vendor daemon process, because they can share a single socket in the vendor daemon.

In practice, we encourage end users to put servers on systems configured with enough file descriptors per process to support the number of FLEX*lm*-enabled applications connecting to the vendor daemon, which may require reconfiguring the kernel to increase the number of file descriptors per process.

Note that multiple daemons can be run on a single network, making the number of TCP clients 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.

B.8 lmgrd Limits

`lmgrd` processes per node	Unlimited
Default port number range	27000-27009
License files per `lmgrd` process	Unlimited

B.9 Subnet, Domain, Wide-Area Network Limits

FLEX*lm* has no limitations regarding subnets (because FLEX*lm* 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.

B.10 LM_LICENSE_FILE, *VENDOR*_LICENSE_FILE

Number of licenses in path	Unlimited

Appendix C

FLEX*lm* Status Return Values

C.1 Error Number Table

These are all the possible errors returned from lc_*xxx*() functions:

Error Number:	Symbolic Name and Description:
-1 LM_NOCONFFILE	"cannot find license file" The license file cannot be opened.
-2 LM_BADFILE	"invalid license file syntax" Feature name is > MAX_FEATURE_LEN, or daemon name is > MAX_DAEMON_LEN, or server name is > MAX_SERVER_NAME, or a feature specifies no hostid and # of licenses is <= 0.
-3 LM_NOSERVER	"cannot connect to a license server" The daemon name specified in the license file FEATURE line does not match the vendor daemon name.
-4 LM_MAXUSERS	"licensed number of users already reached" The licensed number of users has been reached.
-5 LM_NOFEATURE	"no such feature exists" The feature could not be found in the license file.

-6 LM_NOSERVICE	"no TCP "license" service exists" This happens if a SERVER line does not specify a TCP port number, and the TCP license service does not exist in /etc/services.
-7 LM_NOSOCKET	"no socket connection to license manager server" lc_disconn() was called after the process had been disconnected from the socket. This error can also occur if an internal error happens within l_sndmsg() or l_rcvmsg().
-8 LM_BADCODE	"encryption code in license file is inconsistent" The code in a license file line does not match the other data in the license file. This is usually the result of not building all the software components with the same encryption seeds. Check makekey.c, lsvendor.c, and your application code carefully to insure that they are all built with the same encryption seeds.
-9 LM_NOTTHISHOST	"invalid host" The hostid specified in the license file does not match the node on which the software is running.
-10 LM_LONGGONE	"feature has expired" The feature has expired, i.e., today's date is after the expiration date in the license file.
-11 LM_BADDATE	"invalid date format in license file" The start or expiration date in the license file is invalid.

-12 LM_BADCOMM	"invalid returned data from license server" The port number returned from lmgrd is invalid. An attempted connection to a vendor daemon did not result in a correct acknowledgment from the daemon. The daemon did not send back a message within the timeout interval. A message from the daemon had an invalid checksum. An lc_userlist() request did not receive the correct data.
-13 LM_NO_SERVER_IN_FILE	"no SERVER lines in license file" There is no SERVER line in the license file. All non-zero license count features need at least one SERVER line.
-14 LM_BADHOST	"cannot find SERVER hostname in network database" The gethostbyname() system call failed for the SERVER name in the license file.
-15 LM_CANTCONNECT	"cannot connect to license server" The connect() system call failed, while attempting to connect to the daemon. The attempt to connect to the vendor daemon on all SERVER nodes was unsuccessful. lc_status() returns LM_CANTCONNECT if the feature had been checked out but the program is in the process of reconnecting. If reconnection fails, the final status return is LM_CANTCONNECT.

-16 LM_CANTREAD	"cannot read data from license server" The process cannot read data from the daemon within the timeout interval. The connection was reset by the daemon (usually because the daemon exited) before the process attempted to read data.
-17 LM_CANTWRITE	"cannot write data to license server" The process could not write data to the daemon after the connection was established.
-18 LM_NOSERVSUPP	"license server does not support this feature" The feature has expired (on the server), or has not yet started, or the version is greater than the highest supported version.
-19 LM_SELECTERR	"error in select system call" The select() system call failed.
-20 LM_SERVBUSY	"license server busy (no majority)", The license server is busy establishing a quorum of server nodes so that licensing can start. This error is very rare, and checkout should be retried if this occurs.
-21 LM_OLDVER	"license file does not support this version" The version requested is greater than the highest version supported in the license file FEATURE line.
-22 LM_CHECKINBAD	"feature checkin failure detected at license server" The checkin request did not receive a good reply from the vendor daemon (the license might still be considered in use).

-23 LM_BUSYNEWSERV	"license server temporarily busy (new server connecting)" The vendor daemon is in the process of establishing a quorum condition. New requests from clients are deferred during this period. This request should be retried.
-24 LM_USERSQUEUED	"users are queued for this feature" This error is similar to MAXUSERS, but supplies the additional information that there are other users in the queue for this feature.
-25 LM_SERVLONGGONE	"license server does not support this version of this feature" The version specified in the checkout request is greater than the highest version number the daemon supports.
-26 LM_TOOMANY	"request for more licenses than this feature supports" A checkout request was made for more licenses than are available. This request will never succeed.
-29 LM_CANTFINDETHER	"cannot find ethernet device" The ethernet device could not be located on this system.
-30 LM_NOREADLIC	"cannot read license file" The license file cannot be read (errno == EPERM or EACCES).
-31 LM_TOOEARLY	"feature not yet available" The feature is not enabled yet (current date is before the feature start date).
-32 LM_NOSUCHATTR	"No such attribute" A call to lc_get_attr() or lc_set_attr() specified an unknown attribute code.

-33 LM_BADHANDSHAKE	"Bad encryption handshake with daemon" The client performs an encryption handshake operation with the daemon prior to any licensing operations. This handshake operation failed.
-34 LM_CLOCKBAD	"Clock difference too large between client and server" The date on the client system does not agree closely enough with the date on the server (daemon) system. The amount of difference allowed is set by the software vendor with lc_set_attr(LM_A_MAX_TIMEDIFF,...).
-35 LM_FEATQUEUE	"In the queue for this feature" This checkout request has resulted in the process being placed in the queue for this feature. Subsequent calls to lc_status() will yield the status of this queued request.
-36 LM_FEATCORRUPT	"Feature database corrupted in daemon" The daemon's run-time feature data structures have become corrupted. This is an internal daemon error.
-38 LM_FEATEXCLUDE	"User/host on EXCLUDE list for feature" The user/host/display has been excluded from this feature by an end user's vendor daemon options file.
-39 LM_FEATNOTINCLUDE	"User/host not on INCLUDE list for feature" The user/host/display has NOT been included in this feature by an end user's vendor daemon options file.

-40 LM_CANTMALLOC	"Cannot allocate dynamic memory" The malloc() call failed to return sufficient memory.
-41 LM_NEVERCHECKOUT	"Feature was never checked out" This code is returned by lc_status() if the feature requested has never been checked out.
-42 LM_BADPARAM	"Invalid parameter" A call to lc_set_attr() specified an invalid value for its attribute. lc_get_attr(LM_A_MASTER,...) called without connection already established to server.
-43 LM_NOKEYDATA	"No FLEXlm key data supplied in lc_new_job() call" No FLEX*lm* key data was supplied to the lc_new_job() call. Some FLEX*lm* functions will be disabled.
-44 LM_BADKEYDATA	"Invalid FLEXlm key data supplied" Invalid FLEX*lm* key data was supplied to the lc_new_job() call. Some FLEX*lm* functions will be disabled.
-45 LM_FUNCNOTAVAIL	"FLEXlm function not available in this version" This FLEX*lm* function is not available. This could be a result of a BADKEYDATA, NOKEYDATA, or DEMOKIT return from lc_new_job().
-47 LM_NOCLOCKCHECK	"Clock setting check not available in daemon" lc_checkout() returns this code when the CLOCK SETTING check between client and daemon is not supported in this daemon. To disable the clock check lc_set_attr(LM_A_MAX_TIMEDIFF, (LM_A_VAL_TYPE)-1)

-48 LM_BADPLATFORM	"FLEXlm platform not enabled" The software is running on a platform which is not supported by the vendor keys you have purchased. To purchase keys for additional platforms, contact Macrovision.
-49 LM_DATE_TOOBIG	"Date too late for binary format" The start date format in FLEX*lm* licenses are good until the year 2027. This is probably a bad date.
-50 LM_EXPIREDKEYS	"FLEXlm key data has expired" The FLEX*lm* demo vendor keys have expired. Contact Macrovision for new demo keys.
-51 LM_NOFLEXLMINIT	"FLEXlm not initialized" A FLEX*lm* function was called before lc_new_job() was called. Always call lc_new_job() first.
-52 LM_NOSERVRESP	"Server did not respond to message" UDP communications failure. UDP communications are not guaranteed. FLEX*lm* makes a best effort to recover from lost and garbled messages, but this indicates a failure.
-53 LM_CHECKOUTFILTERED	"Request rejected by vendor-defined filter" lc_checkout() failed because of the vendor defined routine which is set in `lsvendor.c`: `ls_outfilter`.
-54 LM_NOFEATSET	"No FEATURESET line present in license file" lc_ck_feats() called, but no FEATURESET line in license file.

-55 LM_BADFEATSET	"Incorrect FEATURESET line in license file" Error return from lc_ck_feats().
-56 LM_CANTCOMPUTEFEATSET	"Cannot compute FEATURESET line" Error return from lc_ck_feats(), which occurs because lc_feat_set() can not compute the FEATURESET line. This can happen because there are no FEATUREs in the file.
-57 LM_SOCKETFAIL	"socket() call failed" This can occur when the UNIX OS runs out of system resources.
-58 LM_SETSOCKFAIL	"setsockopt() failed" The setsockopt() call has failed. This is likely due to an OS error.
-59 LM_BADCHECKSUM	"message checksum failure" Communications error—messages between client and server are encrypted and checksummed for security and integrity. The checksum will usually fail because of poor networking communications.
-61 LM_SERVNOREADLIC	"Cannot read license file from server" This occurs when the license file, via LM_LICENSE_FILE, or lc_set_attr(LM_A_LICENSE_FILE, (LM_AL_VAL_TYPE)*path*), is incorrectly defined. This only occurs in lmutil when LM_LICENSE_FILE is set to *port*@*host* or @*host*.

-62 LM_NONETWORK	"Network software (tcp/ip) not available" This is reported on systems where this is detectable. Some systems may have this problem, but the error will not be reported as LM_NONETWORK—system calls will simply fail.
-63 LM_NOTLICADMIN	"Not a license administrator" Various functions, such as lc_remove() and lc_shutdown(), require that the user be an license administrator, depending on how lmgrd was started.
-64 LM_REMOVETOOSOON	"lmremove request too soon" An lc_remove() request occurred, but ls_min_lmremove (defined in lsvendor.c) seconds have not elapsed since the license was checked out. See ls_vendor().
-65 LM_BADVENDORDATA	"Bad VENDORCODE struct passed to lc_new_job()" LM_CODE() macro was not used to define the VENDORCODE argument for lc_new_job(). See lmclient.h and lmflex.c for an example of how to use the LM_CODE() macro.
-66 LM_LIBRARYMISMATCH	"FLEXlm include file/library mismatch" An attempt was made to create a licensed binary with mismatching source/header files and liblmgr.a. The source code version must match the linking libraries.
-67 LM_NONETOBORROW	"No licenses to borrow." All licenses that were available to borrow have already been borrowed.

-68 LM_NOBORROWSUPP	"License BORROW support not enabled." The license or the application doesn't support license borrowing, but borrowing was requested by a user.
-69 LM_NOTONSERVER	"FLOAT_OK can't run standalone on SERVER." If the license server machine hostid is specified after FLOAT_OK, only the floating license can be used to run the application on the license server machine.
-71 LM_BAD_TZ	"Invalid TZ environment variable" On some operating systems, the end user can significantly change the date using the TZ environment variable. This error detects this type of theft.
-72 LM_OLDVENDORDATA	"Old-style vendor keys (3-word)" lm_init() detected that an old LM_CODE() macro was used.
-73 LM_LOCALFILTER	"Local checkout filter requested request" Request was denied by filter specified in lc_set_attr(LM_A_CHECKOUTFILTER (LM_A_VAL_TYPE)*filter*).
-74 LM_ENDPATH	"Attempt to read beyond the end of LF path" An error occurred with the list of license files.
-75 LM_VMS_SETIMR_FAILED	"SYS$SETIMR call failed" SYS$SETIMR is used on VMS to time out certain FLEX*lm* system calls.
-76 LM_INTERNAL_ERROR	"Internal FLEXlm Error - Please report to Macrovision Software"

-77 LM_BAD_VERSION	"Bad version number - must be floating point number, with no letters" A line in the license file has an invalid version number. lc_checkout() was called with an invalid *version* character string.
-78 LM_NOADMINAPI	"FLEXadmin API functions not available" An attempt to get information from another company's vendor daemon was made via lc_get_attr(LM_A_VD_*, ...). This function call is only allowed for the vendor's own vendor daemon.
-82 LM_BADPKG	"Invalid PACKAGE line in license file" PACKAGE line missing or invalid COMPONENTS. A COMPONENT has number of licenses set, with OPTIONS=SUITE. A COMPONENT has number of licenses==0.
-83 LM_SERVOLDVER	"Server FLEXlm version older than client's" Vendor daemon FLEX*lm* version is older than the client's FLEX*lm* version. This is only supported with a v5.0+ client.

-84 LM_USER_BASED	"Incorrect number of USERS/HOSTS INCLUDED in options file -- see server log" When a feature has the USER_BASED attribute, this error occurs when there no INCLUDE line in the end-user options file for this feature, or the number of users included exceeds the number authorized. See Section 3.5, “FEATURE /INCREMENT Lines,” especially USER_BASED.
-85 LM_NOSERVCAP	"Server doesn’t support this request" This occurs when a vendor daemon with a FLEX*lm* version older than the client is being used. The daemon didn’t understand and respond to the request made by the application.
-87 LM_MAXLIMIT	"Checkout exceeds MAX specified in options file" End-user option MAX has been specified for this feature.
-88 LM_BADSYSDATE	"System clock has been set back" Returned from checkout call.
-89 LM_PLATNOTLIC	"This platform not authorized by license" Returned from checkout call where FEATURE line specifies PLATFORMS=”...”.
-90 LM_FUTURE_FILE	"Future license file format or misspelling in license file" Returned from checkout call when license file attribute was introduced in a later FLEX*lm* version than the client.

-91 LM_DEFAULT_SEEDS	"LM_SEEDs are non-unique" Returned from lc_new_job() or lp_checkout() when vendor name is not demo, but encryption seeds are default encryption seeds.
-92 LM_SERVER_REMOVED	"Feature removed during lmreread or wrong SERVER line hostid" Checkout failure due to two possible causes. 1) The feature is removed during lmreread, but the client is reading an old copy of the license file which still has removed feature. 2) The hostid on the SERVER line is for a different host, so all features in this license file were removed.
-93 LM_POOL	"This feature is available in a different license pool" This is a possible response to LM_A_VD_FEATURE_INFO request, indicating that this INCREMENT line can be ignored, as it has been pooled with another line.
-94 LM_LGEN_VER	"Attempt to generate license with incompatible attributes" Occurs with -verfmt arguments to lmcrypt or makekey, or for lminstall -overfmt. Also set by lc_cryptstr() and lc_chk_conf().
-95 LM_NOT_THIS_HOST	"Network connect to THIS_HOST failed" Returned by checkout(). When this_host is used as a host name. Replace this_host with a real host name to resolve this error.

-96 LM_HOSTDOWN	"Server node is down or not responding" Returned by checkout(); indicates the whole license server system is not up, not just the lmgrd process.
-97 LM_VENDOR_DOWN	"The desired vendor daemon is down" Returned by checkout; indicates lmgrd is running, but not the vendor daemon.
-98 LM_CANT_DECIMAL	"The FEATURE line can't be converted to decimal format" Returned by lc_cryptstr(), or lmcrypt/makekey/lminstall. See Section 3.8, "Decimal Format Licenses," for information on what can't be converted to decimal format.
-99 LM_BADDECFILE	"The decimal format license is typed incorrectly" The internal checksum on the decimal line has indicated the line has been typed incorrectly.
-100 LM_REMOVE_LINGER	"Cannot remove a lingering license" Returned to lmremove command. User has already exited, but license is lingering. lmremove doesn't remove the linger time.
-101 LM_RESVFOROTHERS	"All licenses are reserved for others" Checkout return value when a checkout will never succeed, because the end-user options file has all licenses reserved for others.
-102 LM_BORROW_ERROR	"A FLEXid borrow error occurred"

-103 LM_TSOK_ERR	"Terminal Server remote client not allowed" The feature line does not contain TS_OK to allow Terminal Server client usage.
-104 LM_BORROW_TOOLONG	"Cannot borrow that long" The user has specified a borrow period longer than the license allows.
-106 LM_SERVER_MAXED_OUT	"License server out of network connections" The vendor daemon can't handle any more users. See the `lmgrd` debug log for further information.
-110 LM_NODONGLE	"Dongle not attached, or can't read dongle" In order to read the dongle hostid, the correct driver must be installed.
-112 LM_NODONGLEDRIVER	"Missing Dongle Driver" In order to read the dongle hostid, the correct driver must be installed.
-113 LM_FLEXLOCK2CKOUT	"FLEXlock checkouts attempted" Only one checkout is allowed with FLEX*lock*-enabled applications. Subsequent checkout attempts will fail. They should be disabled if first checkout succeeded in FLEX*lock* mode.
-114 LM_SIGN_REQ	"SIGN= attribute required" This is probably because the license is older than the application. You need to obtain a SIGN= version of this license from your vendor.
-115 LM_PUBKEY_ERROR	"Error in Public Key package." Rare error.

-116 LM_NOCROSUPPORT	"CRO not supported for this platform." You are trying to use CRO, but have not installed CRO keys.
-117 LM_BORROW_LINGER_ERR	"BORROW failed." Borrowing information is invalid.
-118 LM_BORROW_EXPIRED	"BORROW period has expired." Borrowed license can no longer be checked out because it has expired.
-119 LM_MUST_BE_LOCAL	"lmdown and lmreread must be run on license server node" When licenses are borrowed, `lmdown` and `lmreread` must be run on the same machine where the license server is running.
-120 LM_BORROW_DOWN	"Cannot lmdown the server when licenses are borrowed" When licenses are borrowed, you must shut down a license server with the `-force` option.
-121 LM_FLOATOK_ONEHOSTID	"FLOAT_OK license must have exactly one dongle hostid" The hostid to which the FLOAT_OK feature is node-locked must be a single dongle hostid, not a list of dongle hostids.
-122 LM_BORROW_DELETE_ERR	"Unable to delete local borrow info"
-123 LM_BORROW_RETURN_EARLY_ERR	"Support for returning a borrowed license early is not enabled" This support is enabled in the vendor daemon via the `ls_borrow_early_return` variable.
-124 LM_BORROW_RETURN_SERVER_ERR	"Error returning borrowed license on server"

-125 LM_CANT_CHECKOUT_ JUST_PACKAGE	"Error when trying to checkout just a PACKAGE(BUNDLE)"
-126 LM_COMPOSITEID_ INIT_ERR	"Composite Hostid not initialized"
-127 LM_COMPOSITEID_ ITEM_ERR	"An item needed for Composite Hostid missing or invalid"
-128 LM_BORROW_MATCH_ERR	"Error, borrowed license doesn't match any known server license."

Appendix D

Obsolete FLEXible API Features

The functions, attributes, variables, and features listed in this section are obsolete but are still provided in the SDK for backward compatibility. Their functionality has been replaced with more current features; where possible, that current feature is noted in the tables below.

If you are implementing FLEX*lm* for the first time into your application, refer to reference material elsewhere in this manual:

- Chapter 6, “FLEXible API”
- Chapter 7, “Controlling Licensing Behavior”
- Chapter 8, “Advanced FLEXible API Features”
- Chapter 13, “Vendor Daemon”

D.1 Obsolete FLEXible API Functions

The following functions are obsolete, and exist only for compatibility with earlier FLEX*lm* versions. They should be used with care, and questions are welcomed before their use.

Function	Description
lc_alarm()	Sets a timer. See: • Section 6.4.12, “lc_heartbeat()” • Section 7.3.5, “LM_A_CHECK_INTERVAL”
lc_baddate()	Detects if system date has been set back. See: • Section 7.3.4, “LM_A_CHECK_BADDATE” • Section 13.2.2, “ls_a_check_baddate”

Function	Description
lc_chk_conf()	Validates a config structure. See: • Section 6.4.1, “lc_auth_data()”
lc_ck_feats()	Checks the FEATURESET line for a given vendor. FEATURESET is no longer used.
lc_copy_hostid()	Returns a copy of the hostid list. See: • Section 6.4.13, “lc_hostid()”
lc_crypt()	Computes the license key for a FEATURE line. See: • Section 6.4.4, “lc_cryptstr()”
lc_disalarm()	Turns off a timer set with lc_alarm(). See: • Section 6.4.12, “lc_heartbeat()” • Section 7.3.5, “LM_A_CHECK_INTERVAL”
lc_disconn()	Drops the connection to the server. This functionality is no longer provided by FLEX*lm*; use functionality provided by the native operating system.
lc_display()	Returns environment information about the current display. This functionality is no longer provided by FLEX*lm*; use functionality provided by the native operating system.
lc_errtext()	Returns the English text string corresponding to the FLEX*lm* errno. See: • Section 6.4.19, “lc_perror()” • Section 6.4.5, “lc_err_info()”
lc_feat_set()	Computes the FEATURESET code. FEATURESET is no longer used.

Function	Description
lc_get_feats()	Gets a license key from the FEATURESET line. FEATURESET is no longer used.
lc_gethostid()	Returns the hostid for the local host. See: • Section 6.4.13, “lc_hostid()”
lc_getid_type()	Returns the HOSTID of the specified type for the local host. See: • Section 6.4.13, “lc_hostid()” • Chapter 15, “Vendor-Defined Hostid Types”
lc_hostname()	Returns environment information about the current hostname. This functionality is no longer provided by FLEX*lm*; use functionality provided by the native operating system.
lc_isadmin()	Verifies the specified user is a license administrator. This functionality is no longer provided by FLEX*lm*; use functionality provided by the native operating system.
lc_set_errno()	Sets the FLEX*lm* errno. See: • Section 4.4.5, “lc_err_info()” • Section 4.4.19, “lc_perror()” • Section 6.1.6, “lc_get_errno()”
lc_timer()	Exchanges heartbeat messages with the license server. See: • Section 6.4.12, “lc_heartbeat()”
lc_username()	Returns environment information about the current username. This functionality is no longer provided by FLEX*lm*; use functionality provided by the native operating system.

D.2 Obsolete FLEXible API Attributes

The following attributes are obsolete, and exist only for compatibility with earlier FLEX*lm* versions. They should be used with care, and questions are welcomed before their use.

Attribute	Description
LM_A_CONN_TIMEOUT	Sets the timeout for connection to a vendor daemon. See: • Section 7.3.16, "LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL"
LM_A_LKEY_LONG	Enables 64-bit license keys. License keys are obsolete. See: • Section 3.5.17, "SIGN"
LM_A_LKEY_START_DATE	Enables embedded start dates in license keys. License keys are obsolete. See: • Section 3.5.19, "START"
LM_A_MAX_TIMEDIFF	Tests difference in clock settings between the client and the server. Now, automatically performed when needed.
LM_A_SETITIMER, LM_A_SIGNAL (UNIX Only)	Controls which function is used in place of settimer().
LM_A_USE_START_DATE	Enforces start date to be used in check outs. See: • Section 3.5.19, "START"

D.3 Obsolete Vendor Daemon Variables

The following vendor daemon variables are obsolete, and exist only for compatibility with earlier FLEX*lm* versions. They should be used with care, and questions are welcomed before their use.

Variable	Description
`ls_a_lkey_long`	Enables 64-bit license keys. License keys are obsolete. See: • Section 3.5.17, “SIGN”
`ls_a_lkey_start_date`	Enables embedded start dates in license keys. License keys are obsolete. See: • Section 3.5.19, “START”
`ls_enforce_startdate`	Enforces start date to be used in check outs. See: • Section 3.5.19, “START”
`ls_tell_startdate`	Compares start date with system date. See: • Section 13.2.2, “ls_a_check_baddate”
`ls_use_featset`	Enables requirement of FEATURESET line. FEATURESET is obsolete.

D.4 Obsolete License File Features

D.4.1 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 hex 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 v6, 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's 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` and `makekey.c` sources in the `machind` directory of your FLEX*lm* SDK.
- 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 license line template.

  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-yyy
  ```

  for the license key field in the license line template.

  Example:

  ```
  FEATURE f2 demo 1.0 permanent uncounted start:03-jan-2003
              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

- V6 applications (even those accepting 12-character license keys) will accept licenses with long, 20-character license keys.
- Pre-v6 applications will not accept licenses with 12-character license keys.
- License generators (`lmcrypt`, `makekey`) 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 FLEX*lm* v6+, licenses with short keys can be shipped.
- New customers can receive licenses with short keys.

D.4.2 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:

`vendor`	Name of the vendor daemon used to serve at least some feature(s) 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 customer's license file. This is used, for example, to insure that a customer uses a complete update as supplied, without adding old FEATURE lines from the vendor.

SEE ALSO

- Chapter 2, "The License File: Overview"

D.4.3 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 hex characters) are required to achieve a "nearly" unique hostid. It is likely, however, that using the last 16 or 8 hex characters are very nearly unique. Therefore, we recommend 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's put in the license file. So if you want to use 96-bit CPUID, then that's what should go in the license.

CONVERTING FROM 96-BIT TO 32-BIT

The 32-bit hostid is simply the last 9 characters from the 96-bit version. Similarly, the 64-bit is the last 19 characters:

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:

`-cpu`	32-bit hostid
`-cpu32`	32-bit hostid
`-cpu64`	64-bit hostid
`-cpu96`	96-bit hostid

SECURITY ISSUES

Where available, the CPUID is the preferred hostid, because it is likely to be the most secure hostid. We have taken extra precautions in the applications and vendor daemons to make this hostid extra secure.

We do not believe that the CPUID length is important to security. We have every reason to believe that a duplicate 32-bit or 64-bit hostid will be so rare as to be insignificant, although only time will tell.

Index

M