Reference Manual

VERSION 7.1

GLOBEtrotter Software, Inc.

September 2000

Printed in the USA.

September 2000

Table of

Contents

Chapter 1

Introduction

1.1 About This Manual

This manual, the *FLEXlm Reference Manual*, provides a comprehensive description of all aspects of FLEX*lm*® from the software developer's perspective, including a complete description of the FLEXible API, the most complete API available for license management.

The *FLEXlm Programmers Guide* provides an introduction to FLEX*lm*, instructions for evaluating FLEX*lm* on UNIX and Windows systems, descriptions of the Trivial and Simple APIs, and guidelines for integration of FLEX*lm* into your application.

The *FLEXlm End Users Guide* contains information relevant to users of products that utilize FLEX*lm* as their licensing 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.

1.2 How to Use This Manual

This manual should be used as a reference to the advanced features of FLEX*lm*. It should also be used if you plan to use the FLEXible API in your application.

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

1.3 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.

1.4 FLEX*lm* Terms and Definitions

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

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 routines)
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 application software (*client*). FLEX*lm* can also restrict software usage to particular nodes or user names.
Client	An application program requesting or receiving a license.
Daemon	A process that “serves” clients. Sometimes referred to as a *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 GLOBEtrotter Software) and contains the vendor’s unique encryption seeds.
Vendor name	Name of the vendor as found in `lm_code.h`. Used as the name of the vendor daemon.

lmgrd	The daemon 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 the vendor daemons.
Server node	A computer system that is running the license server software. The server node will contain all the site-specific information regarding all feature usage. Multiple server nodes used for redundancy can logically be considered the server node.
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	See Appendix F, "Migrating to the Counterfeit Resistant Option."
Signature	A secure 12- to 120-character hexadecimal number which "authenticates" the readable license file text, ensuring that the license text has not been modified.
License server	An `lmgrd` and one or more vendor daemon processes. License server refers to the processes, not the computer on which they run.

1.5 FLEX*lm* APIs

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

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

- Trivial API
- Simple API
- FLEXible API
- Java API

GLOBEtrotter 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.

In the Trivial and Simple APIs, a licensing "policy" is selected as an argument to the license request call. In these APIs a "heartbeat" function is usually called explicitly by the application, and the policy upon server failure must be programmed into the application.

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.

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.
- Vendor-defined hostid.

If your application requires the FLEXible API *only* for a list of users, you can concurrently use the Simple or Trivial API for licensing and the FLEXible API only for a list of users—this is the recommended solution for this problem.

The Simple and Trivial APIs (as well as the Java API) are documented in the *FLEXlm Programmers Guide*, while the FLEXible API is documented in detail in this manual.

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.

Chapter 2

Incorporating FLEX*lm* Into Your Application

To incorporate FLEX*lm* into your application, you will add function calls to your application program, build your application, and build a custom vendor daemon as discussed in the following sections.

2.1 FLEX*lm* Naming Conventions

All FLEX*lm* client routines and variables adhere to certain naming conventions. These conventions are:

- Trivial API functions are all uppercase MACROS defined in `lmpolicy.h`.
- Simple API function names start with `lp_`. The “p” stands for “policy,” since this is policy-based licensing.
- FLEXible API client routine names start with `lc_`.

2.2 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 Trivial API macros and is a good place to start to learn how to integrate FLEX*lm* with your application. A Simple API example program, `lmsimple.c`, is also available. The source to these three 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 Trivial API example program to demonstrate the usage of UDP and other 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 kit 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 GLOBEtrotter Software may not include them in future FLEX*lm* releases.

2.3 Client Heartbeats and License Server Failures

Your application will need to communicate regularly with the license server via "heartbeat" calls to ensure that the license server is still running. Programming how the heartbeats occur and what action takes place when the license server is not running are the most important part of incorporating FLEX*lm* in an application. Heartbeats for Trivial and Simple APIs are discussed in the *FLEXlm Programmers Guide*. The FLEXible API heartbeat is addressed in the following sections:

- Section 3.21, "lc_heartbeat()"
- Section 4.3, "LM_A_CHECK_INTERVAL"
- Section 4.22, "LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL"
- Section 4.25, "LM_A_USER_EXITCALL"
- Section 4.27, "LM_A_USER_RECONNECT"
- Section 4.28, "LM_A_USER_RECONNECT_DONE"

2.4 Lingering Licenses

A lingering license allows you to specify how long a license will remain checked out beyond either an lc_checkin() call 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 lc_checkout() call.

SEE ALSO

- Section 4.16, "LM_A_LICENSE_FMT_VER"
- Section 5.5, "FEATURE or INCREMENT Lines"
- Section 3.6, "lc_checkout()"

2.5 FLEX*lm* and Multi-threaded Applications

FLEX*lm* can be used in multi-threaded applications as long as the same FLEX*lm* job is not referenced simultaneously in more than one thread. FLEX*lm* is safe for multi-threaded applications, but FLEX*lm* calls are not reentrant.

2.6 Multiple Jobs

lc_new_job() function calls enable 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. For example, one job could use TCP and another job UDP, running simultaneously, although this is not necessarily a good reason for multiple jobs.

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 only be done 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 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 3.17, “lc_free_job()”
- Section 3.19, “lc_get_attr()”
- Section 3.26, “lc_new_job()”
- Section 3.15, “lc_first_job(), lc_next_job()”
- Section 3.29, “lc_set_attr()”

2.7 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 checkout call. 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 lc_checkout(), call:

```
CONFIG *conf;
extern int flexlock_flag;
conf = lc_auth_data(job, feature, name);
if (conf->idptr && (conf->idptr->type == HOSTID_FLEXLOCK))
{
          flexlock_flag = 1;
}
```

Before all subsequent lc_checkout() calls, if `flexlock_flag` is true, then do not make the lc_checkout() call:

```
       if (flexlock_flag || (lc_checkout(...feature2...) == 0))
              /* feature2-enabled */
```

2.8 Security and FLEX*lm*

No software is completely secure, and FLEX*lm* is no exception. While GLOBEtrotter Software has made every effort to ensure the integrity of FLEX*lm*, all points of attack can never be anticipated. GLOBEtrotter Software maintains a list of techniques for making your implementation more secure. These techniques are recommended only for companies with the most stringent security requirements, and are not necessary for most companies. Please contact technical support (support@globes.com) for a description of these techniques. (For security reasons, they are only available to supported companies by email.)

2.8.1 Counterfeit Resistant Option (CRO)

FLEX*lm* offers a Counterfeit Resistant Option (CRO), which is a separately priced add-on. 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*, see the *FLEXlm Programmers Guide* for more information about implementing CRO. If you have shipped applications with pre-v7.1 FLEX*lm* and are considering adopting CRO, see Appendix F, “Migrating to the Counterfeit Resistant Option” in this manual.

2.8.2 Using lmstrip for Additional Security

`lmstrip` and its source, `lmstrip.c`, are included in the FLEX*lm* SDK. `lmstrip` has three related, but different, uses:

- Adds security to UNIX application binaries
- Provides additional security for licensing libraries
- Allows two companies to use two different FLEX*lm* versions in the same binary

The usage for `lmstrip` is:

```
lmstrip filename [-l] [-e | -n] [-r] [-m]
     [-mapfile mapfilename] [strings...]
```

where:

filename	Name of the file to strip.
`-l`	List internal and external names to be stripped.
`-e`	Don’t strip external names.
`-n`	Don’t strip internal and external names.
`-r`	Replace strings with random printable characters.
`-m`	Create or use mapfile. Default mapfile name is `lmstrip.map`. Forces randomized names to be the same across invocations. Required for Windows; optional for UNIX.
mapfilename	Use this file name to override the default mapfile name.
strings	Strip these strings from the executable.

Use `-e` if lc_*xxx*() calls are made from shared library back into your code. Use `-r` if you are linking two versions of FLEX*lm* into the same binary.

By default, `lmstrip` replaces all FLEX*lm* function names with null characters. This adds security to fully linked binaries.

If you're running `lmstrip` on an object file, using the `-r` argument replaces the function names with random characters, each name truncated to no more than six characters.

Adding Security to UNIX Application Binaries

When run on a dynamically linked binary, `lmstrip` adds more security than the normal UNIX `strip` command, because these binaries retain references to the function calls in case they're called from a shared library. `lmstrip` removes any such references.

For this reason `lmstrip` cannot be used as-is on a binary when any lc_*xxx*() call is made from a shared library (which is very rare). Should this apply to you, use `lmstrip -e`, which leaves the lc_*xxx*() calls, but still strips the l_*xxx*() calls. This gives about the same level of security, because the most important functions, from a security viewpoint, are the l_*xxx*() calls.

Because symbol names don't occur in fully linked Windows binaries, this procedure is not needed on Windows.

Providing Additional Security for Licensing Libraries

On UNIX, we recommend the following steps:

1. `ld -r` *`file`*`.o liblmgr.a -o` *`ofile`*`.o`

   `ofile``.o` then includes all necessary FLEX*lm* calls.

2. `lmstrip -r` *`ofile`*`.o`

   This randomizes the names of the FLEX*lm* function calls.

3. You then ship *`ofile`*`.o` to your customers, knowing that they will not see functions called lc_checkout(), etc., in the resulting object file.

On Windows, the usage is:

```
C:> copy lmgr.lib mylmgr.lib
C:> lmstrip -r -m mylmgr.lib
C:> lib mylmgr.lib
C:> lmstrip -r -m myfuncs.lib
C:> lib myfuncs.lib
```

where *`mylmgr`*`.lib` is renamed to be unique for your company.

With `-m`, `lmstrip` creates a mapfile which contains a lookup table of randomized symbol names which is reused later for other object or library files, ensuring the names are mapped identically. For example, “lc_checkout” may be renamed to “xLfH3C.” If this happens in two separate object files, the renaming must be identical.

You can now safely ship *`mylmgr`*`.lib` and *`myfuncs`*`.lib` to your customers. When your customer links their object with *`myfuncs`*`.lib` and *`mylmgr`*`.lib`, everything is resolved and functions correctly. And the temptation to alter the libraries and/or functions is reduced because the function names are not meaningful nor deducible.

LINKING WITH A LIBRARY THAT ALREADY USES FLEX*LM*

On UNIX, follow the steps in “Providing Additional Security for Licensing Libraries” (UNIX, above), using `ld -r` and `lmstrip -r`. The resulting object file can be linked with a library that already calls FLEX*lm*, along with a previous FLEX*lm* library version. Both coexist successfully.

On Windows, follow the steps in “Providing Additional Security for Licensing Libraries” (Windows, above), using `lmstrip -r -m` *`mylmgr`*`.lib` won’t conflict with other companies’ use of FLEX*lm*, because the symbol names are altered.

Chapter 3

FLEXible API

This is the most powerful API available for license management. As such, it contains many options enabling considerable flexibility. Where possible, new applications should use the Simple or Trivial APIs which are documented in the *FLEXlm Programmers Guide*. 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.

3.1 FLEXible API Library Routines

The routines to manage licenses are all contained in the FLEX*lm* client library `liblmgr.a` (`lmgr.lib` for Windows), therefore, you will link your application program with the FLEX*lm* client library. The following are the most commonly used routines—however, the only routines required in your application are lc_new_job() and lc_checkout():

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_err_info()	Useful for translating error messages.
lc_errstring()	Returns an explanatory error string for the most recent error.
lc_free_job()	Frees a job allocated with lc_new_job().
lc_get_attr()	Retrieves a FLEX*lm* client attribute.

lc_get_config()	Gets the first occurrence of the FEATURE line in the cached license file.
lc_heartbeat()	Sends heartbeat from client to server.
lc_hostid()	Gets the system hostid.
lc_idle()	Supports the TIMEOUT option in end-user options file (*vendor*.opt).
lc_init()	Used in place of lc_new_job() in license generators (like lmcrypt and makekey).
lc_new_job()	Initializes FLEX*lm* and creates a license job.
lc_perror()	Prints an error message to stderr.
lc_set_attr()	Sets a FLEX*lm* client attribute.
lc_userlist()	Gets a list of the users of a feature.

These and additional routines are documented in this chapter, and other routines are documented in Appendix E, “Rarely Used Functions and Attributes.” It is rare that an application will require the functions in Appendix E, and care should be used when calling them.

3.2 Building Your Application

The include file lmclient.h contains all the symbolic definitions required for most calls. lm_attr.h contains the definitions used in the lc_set_attr() and lc_get_attr() calls.

If you use any of the FLEX*lm* symbolic definitions, macros, or data structures, you must include lmclient.h in your C module. lc_set_attr() calls require you to include lm_attr.h.

In order to build your application:

1. Insert the FLEXible API calls that you require into your code.
2. Link your code with the FLEX*lm* client library and the FLEX*lm* object file. For static linking:

	FLEX*lm* Object File	**FLEX*lm* Client Libraries**
Standard	`lm_new.o`	`liblmgr.a`
Add for CRO		`libcrvs.a` `libsb.a`

	FLEX*lm* Object File	**FLEX*lm* Client Libraries**	**MSVC++ Libraries**

Standard	`lm_new.obj`	`lmgr.lib`	`libcmt.lib` (/MT)
			`oldnames.lib`
			`kernel32.lib`
			`user32.lib`
			`netapi32.lib`
			`advapi32.lib`
			`gdi32.lib`
			`comdlg32.lib`
			`comctl32.lib`
			`wsock32.lib`

Add for CRO		`libcrvs.lib` `libsb.lib`	
Add for FLEX*lock*		`flock.lib`	

If you want to link with the FLEX*lm* client, FLEX*lock*, and/or Microsoft Visual C++ libraries dynamically, see the *FLEXlm Programmers Guide*.

On UNIX, if you have installed the FLEX*lm* libraries into `/usr/xxx/`, link with a command of the following form:

```
% cc -o program program.o $(OBJS)
              -L/usr/xxx/flexlm/v7.1/platform -llmgr
```

where `$(OBJS)` is the list of the objects for your program and *`platform`* is, for example, `sun4_u5` for Solaris. You can put `-llmgr` anywhere after your objects, and before `-lsocket` and `-lintl`, if needed on your system. For a correct example, see how `lmclient` is linked in the makefile in the *`platform`* directory.

On UNIX, it is strongly recommended that your application be linked with dynamic OS libraries. That is, avoid `-BSTATIC` or linking directly with `libc.a` or other system libraries. Here's why:

- On many UNIX systems, NIS and DNS will fail unless applications are linked with shared system libraries.
- Many important system fixes are implemented by shipping new shared libraries to end users. By linking with static libraries, users often don't obtain essential fixes to applications unless the application is re-linked.

3.3 lc_auth_data()

SYNTAX

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

DESCRIPTION

Gets the license file line for a feature that has been checked out. Since lc_auth_data() only returns features which have been successfully checked out, the data returned is authenticated.

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 call lc_checkout() with the LM_CO_LOCALTEST flag, use the alternate function lc_test_conf() to retrieve the license file line for the tested feature. This can only be done after the most recent call to lc_checkout(). lc_test_conf() takes a job handle parameter and returns a `CONFIG *` struct.

SEE ALSO

- Section 3.20, “lc_get_config()”
- `lmclient.h` for the `CONFIG` struct definition

3.4 lc_check_key()

SYNTAX

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

DESCRIPTION

lc_check_key() determines if the signature in the `CONFIG` is valid. To verify a license file upon installation, you could 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 3.27, “lc_next_conf()”
- Section 3.8, “lc_convert()”
- Section 3.14, “lc_feat_list()”

3.5 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 call is used if the application has need of a feature for a period of time, then no longer needs it. For UDP, this call is essential to free a license, otherwise, the server has to timeout the license. 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. Unused for UDP.

RETURN

None.

3.6 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 lc_checkout() call.

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.

- It is strongly recommended that the application first indicate the expected license file location, with:

  lc_set_attr(*job*, LM_A_LICENSE_DEFAULT, \
  (LM_A_VAL_TYPE) *lic_path*/license.dat)

  The *lic_path* should be a location in your installation hierarchy. Since this is rarely known at compile time, the most common method is to use the registry on Windows, or getenv() on UNIX to find out where the application was installed. This makes license installation and product use easier and more reliable.
- 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	Perform local tests, but do not check out a license (return status). The status from this call will detect all checkout errors that can be determined from the license file *only.* In particular, LM_MAXUSERS/LM_USERQUEUED is not detected.

`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. 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).

The first client that checks out the feature specifies the duplicate grouping for the feature. Any subsequent client that attempts to check out the feature with a different duplicate grouping mask will be rejected and an error reflecting this will appear in the `lmgrd` debug log file. The duplicate grouping value is reset whenever all licenses are checked back in.

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 lc_checkout() call (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_BADFEATPARAM	"Duplicate selection mismatch for this feature" The checkout request for this feature has specified a duplicates mask (`LM_DUP_`*`xxx`*) that does not match the mask specified by an earlier checkout. This is probably the result of using different versions of your client software, or from having an uninitialized variable in the *`dup_group`* field for lc_checkout().
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_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_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.

SEE ALSO

- `machind/lmflex.c`
- Section 4.4, “LM_A_CHECKOUT_DATA”
- Section 4.15, “LM_A_LICENSE_DEFAULT”
- Section 4.10, “LM_A_HOST_OVERRIDE”
- Section 4.7, “LM_A_DISPLAY_OVERRIDE”
- Section 4.26, “LM_A_USER_OVERRIDE”
- Section 3.5, “lc_checkin()”
- Section 5.5, “FEATURE or INCREMENT Lines”
- Section 3.31, “lc_status()”
- Section 2.6, “Multiple Jobs”
- Section 4.6, “LM_A_CKOUT_INSTALL_LIC”

3.7 lc_chk_conf()

SYNTAX

errors = lc_chk_conf(*job*, *conf*, *check_name*)

DESCRIPTION

Given a pointer to a `CONFIG` struct, lc_chk_conf() returns a string describing errors in the struct, or NULL if no problems are found.

- Normally lc_ck_conf() should only be used by a license generation program that calls lc_cryptstr(), such as `lmcrypt`, because warnings are issued on valid license feature lines.
- *`conf`* must be a valid `CONFIG` pointer — otherwise it will core dump.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(CONFIG *) conf`	The feature `(CONFIG *)` to be checked.
`(int) check_name`	If non-zero, error messages will be reported if the feature name is invalid.

RETURN

`(char *) errors`	A descriptive error string, or 0 if no errors found.

3.8 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 3.27, “lc_next_conf()”
- Section 3.8, “lc_convert()”
- Section 3.14, “lc_feat_list()”
- Section 3.9, “lc_cryptstr(),” because lc_convert() has a similar interface to lc_cryptstr()
- Section 4.16, “LM_A_LICENSE_FMT_VER”

3.9 lc_cryptstr()

SYNTAX

```
[global include and variable info:]
#include “lm_code.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]
[...]
   LM_CODE_GEN_INIT(&code);
   if (lc_init(0, VENDOR_NAME, &code, &lm_job))
   {
   /* present error */
   }
   [...]
   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 */
   }
```

DESCRIPTION

Generates license file as a string with signatures filled in. This new function is used by the `lmcrypt` command, and for some vendors will be an easier interface than lc_crypt() for generating licenses. You pass a string, which is a whole, valid license file, with one exception: each signature must be replaced with `SIGN=0` (zero). If you want to generate both a signature and a license key on a FEATURE line, see Appendix F, "Migrating to the Counterfeit Resistant Option."

If *`flag`* has LM_CRYPT_ONLY set, then the function returns the signature for the first FEATURE, INCREMENT, PACKAGE, or UPGRADE line in the file. If the LM_CRYPT_ONLY bit is clear in the *`flag`* argument (`!(`*`flag`*` & LM_CRYPT_ONLY)`), then the whole file is returned as a string, with valid signatures. If *`flag`* has LM_CRYPT_FORCE set, then every line will have the signature recomputed, even if the key is not set to `SIGN=0`. If LM_CRYPT_FORCE is set, and if a line already has a signature, the start date will be taken from the current signature.

Comment lines are retained in the *`return_str`* output.

`return_str` and *`errors`* are malloc'd by lc_cryptstr() and not reused by FLEX*lm*, so it is the responsibility of the caller to free the space returned if needed. (lc_free_mem() should be used on Windows and can be used everywhere, to free this memory).

The default start date is today's date. If you want to specify a start date other than today, then in place of `SIGN=0` in the license key, use the following syntax:

```
start:dd-mmm-yyyy
```

Example:

```
start:1-jan-2005
```

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 *) lm_job`	From lc_new_job().
`(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. (With v7.1, *do not* XOR code.data[0] and code.data[1] with *VENDOR_KEY5*.)
(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 it's 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.

Only 7-bit ASCII characters are supported on FEATURE lines, so lc_cryptstr() reports a warning if an 8-bit character is encountered. To turn off these warnings, OR in the LM_CRYPT_IGNORE_FEATNAME_ERRS *flag* option.

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 3.4, "lc_check_key()"
- Section 3.8, "lc_convert()"
- Section 3.18, "lc_free_mem()
- Section 3.24, "lc_init()"
- Section 4.16, "LM_A_LICENSE_FMT_VER"
- `machind/lmcrypt.c`
- `machind/makekey.c`
- Appendix F, "Migrating to the Counterfeit Resistant Option"

3.10 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 three items that need to be translated are context and long and short error messages, which all depend on the

err_info.maj_errno. *err_info.context* is the English context message, which is also available in machind/lcontext.h. The English error message for *err_info.maj_errno* is in machind/lmerrors.h (short) and machind/lm_lerr.h (long). Given an *err_info.maj_errno* and a language, there should be a unique context string and unique long and short error messages.

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 3.28, “lc_perror()”
- Section 3.12, “lc_errtext()”

3.11 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 FLEX*lm* calls. Check that the previous FLEX*lm* call 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 3.28, “lc_perror()”
- Section 3.12, “lc_errtext()”
- Section 3.10, “lc_err_info()”

3.12 lc_errtext()

SYNTAX

```
string = lc_errtext(job, lm_errno)
```

DESCRIPTION

lc_errtext() returns the English text string corresponding to the FLEX*lm* *lm_errno*. Do not attempt to free memory for this string—it’s managed by FLEX*lm*. It’s value changes when another FLEX*lm* error occurs.

Normally, lc_errstring() or lc_perror() are preferred and recommended, since they contain more information, including the FLEX*lm* minor error number (used by GLOBEtrotter Software for support when needed) and any system error information, if applicable.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(int) lm_errno`	FLEX*lm* error number.

RETURN

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

SEE ALSO

- Section 3.28, “lc_perror()”
- Section 3.10, “lc_err_info()”

3.13 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. Use lc_next_conf(), lc_get_config(), or lc_auth_data() to obtain 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.

3.14 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.

The dupaction() callback routine is called with two parameters:

```
(*dupaction)(feature, vendor)
```

where:

`(char *) feature`	Feature name.

(char *) *vendor*	Vendor daemon for *feature*.

ERROR RETURNS

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

3.15 lc_first_job(), lc_next_job()

SYNTAX

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

DESCRIPTION

lc_first_job() and lc_next_job() are 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 3.17, “lc_free_job()”
- Section 3.26, “lc_new_job()”
- Section 2.6, “Multiple Jobs”

3.16 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.

Note: Do not use this function on the return data from lc_gethostid() or lc_getid_type(), because they free their own memory.

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 E.1.1, “l_new_hostid()”

3.17 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(). On Windows, this call is mandatory and must be matched to the corresponding lc_new_job() call. On UNIX, this call is needed only by an application that uses a large number of jobs over its lifetime.

PARAMETERS

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

RETURN

None.

ERROR RETURNS

LM_BADPARAM	No such job.

SEE ALSO

- Section 3.24, “lc_init()”
- Section 3.26, “lc_new_job()”
- Section 3.29, “lc_set_attr()”
- Section 2.6, “Multiple Jobs”

3.18 lc_free_mem()

SYNTAX

```
(void) lc_free_mem(job, char_pointer)
```

DESCRIPTION

lc_free_mem() frees memory allocated by another FLEX*lm* function. lc_free_mem() can be used everywhere, but is currently only needed on Windows.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) char_pointer`	Memory allocated by lc_cryptstr() or lc_convert().

RETURN

None.

SEE ALSO

- Section 3.8, “lc_convert()”
- Section 3.9, “lc_cryptstr()”

3.19 lc_get_attr()

SYNTAX

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

DESCRIPTION

Retrieves a FLEX*lm* attribute. The key describes which attribute to retrieve, and the value is a pointer to the value for the attribute. See `lm_attr.h` for key constants and value types.

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 3.29, "lc_set_attr()"
- Chapter 4, "Controlling Licensing Behavior with lc_set_attr()"

3.20 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. For an authenticated 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 3.3, “lc_auth_data()”
- Section 3.27, “lc_next_conf()”

3.21 lc_heartbeat()

SYNTAX

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

DESCRIPTION

lc_heartbeat() exchanges heartbeat messages with the license server. By default, heartbeats are sent automatically, using lc_timer(). To use lc_heartbeat(), you must call lc_set_attr(*job*, LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE)-1) to turn off the automatic lc_timer(). Heartbeat messages are strongly recommended for security—for the client to ensure that it will re-checkout its licenses from a restarted server, thereby reducing over usage. Heartbeats are not needed for the server to retain a client's license (unless UDP communications is used)—the server retains the license until the client exits. If lc_heartbeat() is called, the client will automatically reconnect and re-checkout from a 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 server is down, and how many reconnect attempts have been made. This can be used in many ways, to inform the user the server is down, and possibly to deny use after a specified number of failures.

The arguments *num_reconnects* and *num_minutes* are optional. Their use is recommended where security is particularly important—otherwise they can be safely set to 0, and they will be ignored. If utilized, they can indicate that a server has been stopped and started many times in a few minutes, 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.

3.21.1 How lc_heartbeat() Works

lc_heartbeat() sends a heartbeat to the server. It then reads the response from the previously sent heartbeat. The first heartbeat is sent when the application first connects to the server, usually in lc_checkout(). In this manner, there is normally no delay in lc_heartbeat().

If lc_heartbeat() is unable to read a response from the server, it attempts to reconnect to the server. If the application has set an LM_A_USER_RECONNECT function, this function will also get called, which is useful if lc_heartbeat() is registered as a callback (the default). If this reconnect fails, then an internal flag is set and subsequent calls to lc_heartbeat() will attempt reconnection. These attempts are made for LM_A_RETRY_COUNT times on UNIX (on Windows, the attempt is made forever). If a reconnection occurs before LM_A_RETRY_COUNT attempts, the LM_A_USER_RECONNECT_DONE routine, if specified, will be called. If a reconnection fails to occur after LM_A_RETRY_COUNT attempts, the LM_A_USER_EXITCALL routine, if specified, will be called. If LM_A_USER_EXITCALL is not specified, the application will exit with the error message, "Lost license, cannot reconnect" to stderr.

3.21.2 lc_heartbeat(), User TIMEOUT Option, and UDP Timeout

If lc_heartbeat() is not called for an extended period, then the application may lose its license. This can happen for two reasons: LM_A_TCP_TIMEOUT has expired or the end user has set a TIMEOUT for this feature in the end-user

options file. In both cases, the 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.

3.22 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. This function is recommended in the future; avoid functions that deal with `HOSTID *` struct information, because this struct may change from version to version.

Note that lc_hostid() may return a space-separated list of hostids, if appropriate.

PARAMETERS

`(LM_HANDLE *)` *`job`*	From lc_new_job().
`(int)` *`id_type`*	Hostid types (HOSTID_*xxx*) are specified and described in `lmclient.h`.

RETURN

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

`(char *) buf`	A pointer to a `char` array of length MAX_CONFIG_LINE. If successful, the hostid string is returned here.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this *`id_type`*.

SEE ALSO

- `lmclient.h` for definition of `HOSTID` struct

3.23 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 */
... get 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 3.21, “lc_heartbeat()”
- Section 4.24, “LM_A_TCP_TIMEOUT
- Section 9.2.9, “ls_minimum_user_timeout”

3.24 lc_init()

SYNTAX

See lc_cryptstr().

DESCRIPTION

lc_init() should only be used with license generators and should not normally be used in applications shipped to clients. Use lc_new_job() instead, because it offers enhanced security.

SEE ALSO

- Section 3.9, “lc_cryptstr()”
- Section 3.26, “lc_new_job()”

3.25 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 the string is LM_LOG_MAX_LEN.

RETURN

None.

ERROR RETURNS

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

SEE ALSO

- Chapter 7, “Distributing and Locating the License File”

3.26 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() should not be used with license generators (like `lmcrypt` and `makekey`). Use lc_init() instead.

All applications that call lc_new_job() must link `lm_new.o` (`lm_new.obj` on Windows) into their application. If the application fails to link with an error about `l_n36_buf`, it means that you need to link in `lm_new.o` (`lm_new.obj`).

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.

Note: lc_new_job() MUST be the first FLEX*lm* call you make in your application. Do NOT call lc_set_attr() or lc_get_attr() before calling lc_new_job().

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. This parameter is also safe for non-DLL code.

RETURN

`pointer to (VENDORCODE)` *`code`*	Pointer to `VENDORCODE` struct. Initialized in this call 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	malloc() call 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 3.24, “lc_init()”
- Section 3.17, “lc_free_job()”
- Section 2.6, “Multiple Jobs”

3.27 lc_next_conf()

SYNTAX

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

DESCRIPTION

Returns the next line in the license file matching *feature*. The search is started from *pos*. 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. For an authenticated feature line, you must first checkout 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. Updated 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 3.3, “lc_auth_data()”

3.28 lc_perror()

SYNTAX

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

DESCRIPTION

Prints a FLEX*lm* error message, in the same format as the UNIX routine 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 message box 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 3.10, “lc_err_info()”
- Section 3.11, “lc_errstring()

3.29 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. The *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

- Chapter 4, “Controlling Licensing Behavior with lc_set_attr()”

3.30 lc_set_registry()

SYNTAX

```
(void) lc_set_registry(job, env_var, value)
```

DESCRIPTION

Used on Windows to facilitate setting an environment variable. This call allows you to write into the registry (assuming your program has the appropriate security attributes).

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) env_var`	The environment variable name.
`(char *) value`	Value of the environment variable.

RETURN

None.

ERROR RETURNS

LM_NOADMINAPI	Request was made to other company’s vendor daemon.

3.31 lc_status()

SYNTAX

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

DESCRIPTION

Returns the status of the requested feature.

This call is used only 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 wait in the lc_checkout() call. If the application needs to continue doing processing, use LM_CO_QUEUE in an lc_checkout() call and call lc_status() immediately after lc_checkout() and any other lc_*xxx*() calls 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("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 3.6, “lc_checkout()”

3.32 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: lc_userlist() is a potentially expensive call (it may cause a lot of network traffic), depending on the number of users of *`feature`*. Therefore this call 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 checkout call, 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	malloc() call failed.
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOFEATURE	Specified feature cannot be found.

SEE ALSO

- `lmclient.h` for `LM_USER` structure definition.

3.33 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 9.2.16, “ls_vendor_msg”

Chapter 4

Controlling Licensing Behavior with lc_set_attr()

The FLEXible API allows you to control the licensing behavior of your application with a set of *attributes*. FLEX*lm* attributes allow you control over licensing policy, internal operations of FLEX*lm* (e.g., use of timers, 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 FLEX*lm* attributes, use the lc_set_attr() call, described in Section 3.29, “lc_set_attr().”

Essential FLEXible API attributes which should be set by every FLEXible API application, are:

- License file location:
  LM_A_LICENSE_DEFAULT
- Heartbeat security policy:
  LM_A_CHECK_INTERVAL
  LM_A_RETRY_INTERVAL
  LM_A_USER_RECONNECT
  LM_A_USER_RECONNECT_DONE
  LM_A_USER_EXITCALL
- Performance:
  LM_A_RETRY_CHECKOUT

The following attributes are often useful:

- Vendor-defined Hostid:
  LM_A_VENDOR_ID_DECLARE
  LM_A_VENDOR_GETHOSTID

- Customized checkout:
  LM_A_CHECKOUTFILTER
  LM_A_CHECKOUT_DATA
- Information useful for error, or informational, reporting:
  LM_A_LF_LIST
  LM_A_VD_GENERIC_INFO
  LM_A_VD_FEATURE_INFO
- Disabling SIGALRM, for applications such as applications that use FORTRAN and XView, that cannot tolerate any use of SIGALRM:
  LM_A_SETITIMER
  LM_A_SIGNAL

The other attributes are rarely needed, and are listed in Appendix E, “Rarely Used Functions and Attributes.”

The attributes are changed with lc_set_attr() and queried with lc_get_attr(). The section heading is the attribute name. The first line of each section is the data type of the attribute. All attribute definitions are in `lm_attr.h`.

When using these attributes with lc_set_attr(), the argument must be of the correct type (each item 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 *`.

4.1 LM_A_BEHAVIOR_VER

Type: `(char *)`

Default: LM_BEHAVIOR_V7_1

The overall behavior for all FLEX*lm* components can easily be set in LM_VER_BEHAVIOR in `lm_code.h`.

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

For the vendor daemon, in `lsvendor.c`, set:

```
char *ls_a_behavior_ver = LM_BEHAVIOR_Vx;
```

4.2 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 6.1.2, “Limited Functionality Demos”
- Section 9.2.2, “ls_a_check_baddate”

4.3 LM_A_CHECK_INTERVAL

Type: `(int)`

Default: 120 second interval

LM_A_CHECK_INTERVAL controls the client’s detection of daemon failures. FLEX*lm* client routines will install a SIGALRM handler or no handler at all, based on LM_A_CHECK_INTERVAL. 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`	< 0	No SIGALRM timer installed.
`check_interval`	>= 0, < 30	Old interval unchanged.
`check_interval`	>= 30	Timer interval.

If you do not enable FLEX*lm*’s timer, you must call lc_heartbeat() (or lc_timer()) periodically to check the status of the license. You cannot set *`check_interval`* to less than 30 seconds with lc_set_attr().

The timer handler remembers any other handler that was installed, and calls the previously installed handler when it has checked the socket. If it is unacceptable to have handlers installed for either of these signals (or to have the intervals changed), then set *`check_interval`* < 0. If you set

check_interval < 0, then no checking of the daemon will be done unless you call lc_timer() periodically. You could, of course, do this from your own timer signal handler.

Starting with FLEX*lm* v6.0, the default timers included in FLEX*lm* can be used for any Windows application, whether it is a 32-bit Console or WIN32 application. You may still use your own timers by disabling the internal ones.

SEE ALSO

- Section 3.21, “lc_heartbeat()”
- Appendix B, “UDP Communications”
- Section 4.23, “LM_A_SETITIMER, LM_A_SIGNAL (UNIX Only)”

4.4 LM_A_CHECKOUT_DATA

Type: `(char *)`

Default: None.

LM_A_CHECKOUT_DATA allows you to send some vendor-specific data to the vendor daemon in addition to the normal USER/HOST/DISPLAY data. This checkout data can be used to group duplicates in addition to the USER/HOST/DISPLAY by setting the LM_DUP_VENDOR bit in the duplicate grouping bitmask passed to lc_checkout(). If LM_DUP_VENDOR is used, LM_A_CHECKOUT_DATA must be set. The checkout data can be modified before each individual lc_checkout() or lc_checkin() call. This makes it possible for a process to check out several different independent licenses (if LM_DUP_VENDOR is in the duplicate mask) and to check in the licenses independently by setting the vendor-defined field each time before calling lc_checkin(). The vendor-defined data is a character string, with a maximum size of MAX_VENDOR_CHECKOUT_DATA bytes (32 bytes).

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

Each checkout or checkin request uses the value of the vendor-defined data from the last lc_set_attr() call. Checkins will only be performed for features on which the vendor-defined field matches.

4.5 LM_A_CHECKOUTFILTER

Type: Pointer to a function returning `int`.

Default: None.

The checkout filter allows you to examine the FEATURE line which is going to be used in an lc_checkout() request, and either allow the checkout to proceed or reject this particular FEATURE line. This filter function will be called with a pointer to the `CONFIG` struct which is about to be checked out. If this function returns 0, then checkout proceeds; otherwise, if this function returns a non-zero value, then the checkout proceeds to the next available FEATURE line. If this function returns a non-zero value and sets the error obtainable from lc_get_errno(), then this value will be the return of lc_checkout(), otherwise, if lc_get_errno() is set to 0 by this function, the result of lc_checkout() would be LM_LOCALFILTER (assuming the checkout was not attempted on further FEATURE lines or that another FEATURE line did not produce a LM_MAXUSERS/LM_USERSQUEUED result).

Note: Using LM_A_CHECKOUTFILTER when the client is not reading the license file (via `@`*`host`*, *`port`*`@`*`host`* or USE_SERVER) requires the license server to pass each license to the client for verification. For this reason, LM_A_CHECKOUTFILTER should be used with discretion.

4.6 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"

4.7 LM_A_DISPLAY_OVERRIDE

Type: (char *)

Default: No override of display name.

This string, if specified, will be used to override the display name as derived from the UNIX ttyname() system call.

Note: This value cannot be changed for a job after the initial connection to the vendor daemon.

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 an X call. 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.

The following example code can be used for this purpose:

```
/*
 * assume XOpenDisplay or XtAppInitialize has already been called
 */
#include <netdb.h>
char display_name[50], *cp, display_ip[9];
struct hostent *he;
/*...*/
strncpy(display_name, DisplayString(display), 49);
if (!(strncmp(display_name, ":0", 2)) ||
                !(strncmp(display_name, "unix:0", 6)) ||
                !(strncmp(display_name, "localhost:0", 12)))
```

```
{
  static char d[50];
     gethostname(d, 47);
     if (*d)
     {
             strcat(d, ":0");
             display_name = d;
     }
}
he = gethostbyname(display_name)
sprintf(display_ip, "%x", *((int *)he->h_addr));
lc_set_attr(LM_A_DISPLAY_OVERRIDE, display_ip);
```

4.8 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 2.7, “FLEXlock,” for additional information on FLEX*lock*.

4.9 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_LOCAL_MACHINE->SOFTWARE->GLOBEtrotter Software Inc.->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* for additional information on FLEX*lock*.

4.10 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.

4.11 LM_A_LCM

Type: `(int)`

Default: True

Used to turn off the License Certificate Manager. The LCM is available only on Windows.

4.12 LM_A_LCM_URL

Type: `(char *)`

Default: www.globetrotter.com/*vendor*, where *vendor* is your vendor daemon name.

Used to override the License Certificate Manager URL default:

```
lc_set_attr(job, LM_A_LCM_URL,
     (LM_A_VAL_TYPE)"www.mycompany.com/licenses");
```

See the *FLEXlm Programmers Guide* for additional information on LCM. The LCM is available only on Windows.

4.13 LM_A_LF_LIST

Type: Pointer to `(char **)`

List of all license files searched for features. Useful for failure messages for debugging. Note that lc_lic_where() prints only one file, the one last searched. For example:

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

4.14 LM_A_LICENSE_CASE_SENSITIVE

Type: `(int)`

Default: False

If True, the license file is case-sensitive. Before v6, license files were largely case-sensitive. The default is strongly recommended, and makes end-user usage much easier. This should be set to True to generate license files compatible with older versions of FLEX*lm*. This attribute is automatically turned on by setting LM_VER_BEHAVIOR in `lm_code.h` to LM_BEHAVIOR_V5_1 or less.

4.15 LM_A_LICENSE_DEFAULT

Type: `(char *)`

The default license file location. We recommend that this be set to the default location in your distribution hierarchy. If LM_A_LICENSE_DEFAULT is set, FLEX*lm* still honors the *VENDOR*`_LICENSE_FILE` and `LM_LICENSE_FILE` environment variables first.

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

4.16 LM_A_LICENSE_FMT_VER

Type: (char *)

Default: LM_BEHAVIOR_V7_1

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, or 7_1. Note that this is not automatically set by LM_VER_BEHAVIOR in lm_code.h. 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 3.9, “lc_cryptstr()”

4.17 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 2.4, “Lingering Licenses””

4.18 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
```

This attribute is automatically turned off by setting LM_VER_BEHAVIOR in `lm_code.h` to LM_BEHAVIOR_V5_1 or less.

4.19 LM_A_PERROR_MSGBOX (Windows Only)

Type: `(int)`

Default: True

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

4.20 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.

4.21 LM_A_RETRY_CHECKOUT

Type: `(int)`

Default: False (for backward compatibility, but we recommend setting to True).

When True, checkouts that fail due to communications errors are automatically retried once. Often this second attempt will succeed on networks with poor communications. This is turned on by default in both the Simple and Trivial API, and the default is off in the FLEXible API. Use lc_set_attr(*job*, `LM_A_RETRY_CHECKOUT, (LM_A_VAL_TYPE)1`) to turn this attribute on for the FLEXible API (recommended). It's turned off by default in the FLEXible API so that previous default behavior is preserved.

4.22 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 daemon. Once daemon failure is detected, the client library routines will attempt to reconnect to a daemon. If reconnection fails, then the reconnect will be re-attempted LM_A_RETRY_COUNT times at intervals of LM_A_RETRY_INTERVAL. This timing will be done with the same timer that detects the daemon's failure. If no FLEX*lm* timers (SIGALRM) are desired, set LM_A_RETRY_INTERVAL to a negative value. 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. In addition, on Windows, it is not legal to set LM_A_RETRY_COUNT to anything other than -1 without also setting LM_A_USER_EXITCALL, because there is no default behavior for exiting a Windows application.

SEE ALSO

- Section 3.21, “lc_heartbeat()”
- Section 4.3, “LM_A_CHECK_INTERVAL”

4.23 LM_A_SETITIMER, LM_A_SIGNAL (UNIX Only)

Type: Pointer to a function returning `void`.

Default: setitimer() and signal()

This option allows you to replace setitimer() with a routine of your choice. This might be done, for example, if your application is written in FORTRAN on UNIX, where use of SIGALRM is not allowed.

To disable SIGALRM, create a function that does nothing and use a pointer to this function as the setting for both these attributes.

```
null_func() {}
/* ... */
lc_set_attr(job, LM_A_SETITIMER, (LM_A_VAL_TYPE)null_func);
lc_set_attr(job, LM_A_SIGNAL, (LM_A_VAL_TYPE)null_func);
```

4.24 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 3.21, “lc_heartbeat()”
- Appendix B, “UDP Communications”

4.25 LM_A_USER_EXITCALL

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

Default: No user exit handler (program exits).

The function pointer LM_A_USER_EXITCALL can be set to point to the 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 program will exit. If the LM_A_USER_EXITCALL function returns control to its caller, program operation will continue as if no license had been checked out. The LM_A_USER_EXITCALL routine is called as follows:

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

The exitcall() function will be called for *each feature* that the program had checked out, if that feature's license is lost. If the exitcall() function returns, it will be called again for the next feature. After it has been called for all features, control will return to the program at the point where detection of loss of licenses occurred.

SEE ALSO

- Section 3.21, "lc_heartbeat()"

4.26 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.

4.27 LM_A_USER_RECONNECT

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

Default: No user reconnection handler.

This reconnection routine is called each time just before a reconnection is attempted, either automatically as a result of the timer set by LM_A_CHECK_INTERVAL, or as a result of the application program calling lc_timer().

The reconnection routine is called as follows:

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

where:

`(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.

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

SEE ALSO

- Section 3.21, “lc_heartbeat()”
- Section 4.3, “LM_A_CHECK_INTERVAL”

4.28 LM_A_USER_RECONNECT_DONE

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

Default: No user `reconnect_done` handler.

This function will be called when reconnection is successfully completed.

The reconnection done handler is called as follows:
`(*reconnect_done)(feature, tries, total_attempts, interval)`

where:

`(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.

4.29 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(), and upon successful return, this struct is filled with the appropriate information. The following example illustrates the use of both attributes:

```
#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:

```
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);
}
```

4.30 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

- Section 5.13.3, "Vendor-Defined Hostids"
- `lmclient.h` for `LM_VENDOR_HOSTID` definition
- `examples/vendor_hostid` directory

4.31 LM_A_VERSION and 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().

4.32 LM_A_WINDOWS_MODULE_HANDLE

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`) gets linked into your DLL. Or put another way, the FLEX*lm* calls are not in a static binary, but only in a DLL. In this case, the DLL should make the following call 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 this call is not made, Windows dialogs and error messages do not work properly.

Chapter 5

The License File

Please refer first to the license file description in the *FLEXlm Programmers Guide*—especially the license file examples—to get an overview of the license file. The following is a detailed description of every license file attribute. Most companies need only use a small portion of the capabilities of the license file.

5.1 Format of the License File

A license file consists of the following sections:

SERVER/VENDOR lines

These lines appear in the license file if a license server is used (that is, if any features are *counted*). The SERVER line(s) contain information about the node(s) where `lmgrd` is running. The vendor-specific VENDOR line(s) contain information about the vendor daemon(s) that run 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. GLOBEtrotter recommends the use of a USE_SERVER line, particularly where performance is important.

FEATURE lines

This section consists of any combination of FEATURE, INCREMENT, UPGRADE, or PACKAGE lines. This section is required in the license file read by `lmgrd`. This section is also required in the license file read by a client application, unless a USE_SERVER line is used.

Comment lines

Comment lines should begin with a "#" character. However, in practice, all lines not beginning with a FLEX*lm* reserved keyword are considered comments.

Long lines can be broken up. It is customary to use a "\" line continuation character, but in v7+ this is not required, particularly because newlines are often added by emailers.

Note: See the *FLEXlm Programmers Guide* for information on `lmcrypt` and `makekey`, the license generation utilities. Also see Section 3.9, "lc_cryptstr()," for generating licenses with a C function call.

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

End users often need to edit a few fields in the license file. Nearly all of the fields in a license file are authenticated; if the authenticated portions are edited by the license administrator, an LM_BADCODE error will result.

The only data items in the license file that are editable by the end user are:

- Host names on SERVER lines
- (Optional) port numbers on the SERVER or VENDOR lines
- (Optional) path names on VENDOR lines
- (Optional) options file path names on VENDOR lines
- (Optional) 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.

Note: The SERVER hostid(s) 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.

5.2 SERVER Lines

The SERVER line specifies the node name and hostid of the license server machine and the port number of the `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.

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.

`SERVER` *`host hostid`* [*`port`*]

`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). Note that a hostid list on the SERVER line is not supported. Alternate special hostids can also be specified here, including ANY, HOSTNAME=*host*, etc. See Section 5.13, “Hostids for FLEXlm-Supported Machines,” for information about expected, special, and 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, or if the vendor daemon or clients are older than FLEX*lm* v6. Using a port number in the range 27000-27009 is recommended when specifying a port number, because v6 utilities and clients can then use `@`*host* to find the server.

SEE ALSO

- Section 5.13, “Hostids for FLEXlm-Supported Machines”
- Section 5.13.2, “Special FLEXlm Hostids”

5.3 VENDOR Line

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

Note: Prior to FLEX*lm* v6, the VENDOR line was called a DAEMON line. DAEMON is still recognized, and DAEMON is required for `lmgrd`s and vendor daemons older than v6.

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

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, the PATH environment variable, plus the current directory, is used by `lmgrd` to find the daemon process to start. Prior to v6, this path was required.
options_file_path	Path to the end-user options file for this daemon. If unspecified, a v6+ 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. Vendor daemons older than v6 will not look for *vendor*`.opt`.
port	Vendor daemon port number. The default, if *port* is not specified, is chosen by the system at runtime. 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.

UNIX EXAMPLES

```
VENDOR xyzd
VENDOR xyzd /etc/xyzd
VENDOR xyzd /etc/xyzd options=/a/b/c/licenses/xyzd.opts
```

WINDOWS EXAMPLES

```
VENDOR xyzd C:\Windows\system\xyzd.exe
VENDOR xyzd C:\Windows\system\xyzd.exe \
  options=C:\licenses\xyzd.opts
```

5.4 USE_SERVER Line

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

5.5 FEATURE or INCREMENT Lines

A FEATURE line describes the license to use a product. 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.

If the vendor daemon has ls_use_all_feature_lines set in `lsvendor.c`, then FEATURE lines function as INCREMENT lines, and the behavior of a FEATURE line is unavailable to that application. GLOBEtrotter Software strongly discourages the use of ls_use_all_feature_lines.

Only one FEATURE line for a given feature will be 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), then you must use multiple INCREMENT lines. INCREMENT lines form license groups, or *pools*, based on the feature name, version, node-lock hostid, USER_BASED, HOST_BASED, and CAPACITY fields. If two lines differ by any of these fields, they are counted separately in separate pools. A FEATURE line does not give an additional number of licenses, whereas an INCREMENT line always gives an additional number of licenses.

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

The current syntax of FEATURE and INCREMENT lines (FLEX*lm* v7.1) is:

```
FEATURE|INCREMENT feature vendor \
     feat_version exp_date num_lic [options...] \
     SIGN=sign
```

The optional *keyword=value* pairs must appear after all required fields, but can appear in any order. For optional pairs, if *keyword* is lowercase, its value can be modified and the license will remain valid.

5.5.1 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.

5.5.2 Vendor Daemon Name

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

5.5.3 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.

5.5.4 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,” or an expiration date with the year as 0, e.g., 1-jan-0. Two-digit years are assumed to be 19xx and are valid only up till 1999. For years 2000 and later, you must use four digits. 1-jan-0 = 1-jan-00 = 1-jan-0000 = permanent. FLEX*lm* fully supports dates beyond 2000. Prior to v6, the keyword “permanent” was not recognized.

5.5.5 Number of Licenses

Number of licenses for this feature. Use “uncounted” or 0, for unlimited use of node-locked licenses. Prior to v6, the keyword “uncounted” was not recognized.

5.5.6 Signature

Signature for this FEATURE line. The signature is produced by lc_cryptstr() in `lmcrypt` or `makekey`, or a vendor-defined routine. The signature is from 12-120 characters and is preceded by SIGN=. When using `lmcrypt`, put `SIGN=0` at the end of each FEATURE line, and `lmcrypt` will replace the `0` with the correct signature.

5.5.7 HOSTID

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

Optional field. Case-insensitive strings returned by `lmhostid`. Use if this feature is to be bound to a particular host or hosts, whether its use is counted or not. If the license is uncounted, then this field is required. If hostid is DEMO, ANY, or ID=, the license is valid on any system. If DEMO, the application can determine this is a demo license by calling lc_auth_data() and noting the hostid type. All other *special* hostids are supported: INTERNET=*###.###.###.###*, etc. This can be a list of hostids using a space separator and quotes, e.g.:

```
HOSTID="12345678 FLEXID=876321 HOSTNAME=joe"
```

If a list of hostids is used, the license is granted if the client is on any one of the hostids in the list. See Section 5.13, “Hostids for FLEXlm-Supported Machines,” for information about expected, special, and vendor-defined hostids.

5.5.8 CAPACITY

```
CAPACITY
```

Optional field. This is used in conjunction with the LM_A_CAPACITY attribute to lc_set_attr(), available in the FLEXible API. The most common purpose of CAPACITY is to charge more for a more powerful system. For example, with CAPACITY, you could automatically check out more licenses on a UNIX system than on a PC, thereby effectively charging more for the more powerful system. CAPACITY is a checkout multiplier—if lc_checkout() requests 1 license, and LM_A_CAPACITY is set to 3, three licenses will be checked out. CAPACITY is set 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);
```

If CAPACITY is missing from the FEATURE line, the attribute setting in the code will have no effect. Similarly, if CAPACITY is on the FEATURE line, but there is no call to lc_set_attr(`...LM_A_CAPACITY...`), it will have no effect.

The attribute must be set before the first connection to the server (usually lc_checkout()) and cannot be reset once set. If an INCREMENT appears where some licenses are CAPACITY and some are not, the vendor daemon tracks these in separate license pools.

5.5.9 DUP_GROUP

```
DUP_GROUP=NONE|SITE|[UHDV]
```

Optional field. You can specify the duplicate grouping (license sharing) parameter in the license in FLEX*lm* v4.0 or later. If DUP_GROUP= is specified in the license, this parameter overrides the *dup_group* parameter in the lc_checkout() call. 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 not consume additional licenses.

5.5.10 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.

5.5.11 ISSUED

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

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

5.5.12 ISSUER

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

Optional field. Issuer of the license.

5.5.13 MINIMUM

```
MINIMUM=n
```

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

5.5.14 NOTICE

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

Optional field. A field for intellectual property notices.

5.5.15 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."

5.5.16 PLATFORMS

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

Optional field. This allows you to restrict usage to particular hardware platforms. The platforms are defined as the same platforms that are used to license FLEX*lm* itself: `sun4_u5`, `i86_n3`, etc. The names can be found in Chapter 11, "UNIX Platform-Specific Notes," and in Chapter 12, "Windows Platform-Specific Notes." 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 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're are pooled and counted separately.

Examples:

```
FEATURE f1 ... PLATFORMS=sun4_u5
INCREMENT f2 ... 1 PLATFORMS="i86_n3 hp700_u9"
INCREMENT f2 ... 1 PLATFORMS="i86_n3"
```

Feature "f1" can be used on any Sparc station running Solaris.

Feature "f2" can be used on a PC running Windows or an HP machine. 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."

5.5.17 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.

5.5.18 START

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

Optional field. Feature start date.

5.5.19 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.

5.5.20 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`*.

Note that the start date is the one field which is not readable in the license file and is part of the signature.

The ISSUED field makes this more readable (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.”

5.5.21 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.

5.5.22 VENDOR_STRING

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

Optional field. Vendor-defined license data. If checkout is to be conditioned by what’s in 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`.

SEE ALSO

- Section 4.5, “LM_A_CHECKOUTFILTER”
- Section 9.2.4, “ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade”

5.5.23 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 or checksum.

5.5.24 ck

```
ck=nnn
```

Optional field. A checksum, useful with the `lmcksum` utility, which will verify that the license has been entered correctly by the license administrator. When using `lmcrypt`, put `ck=0` on each FEATURE line, and `lmcrypt` will replace the `0` with the correct checksum. Not encrypted.

5.5.25 dist_info

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

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

5.5.26 user_info

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

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

5.5.27 vendor_info

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

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

5.5.28 FEATURE/INCREMENT Examples

To illustrate INCREMENT, the two feature lines:

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

would only result in four licenses for v1.0 *or* five licenses for v2.0, depending on their order in the file, whereas:

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

would result in four licenses for v1.0 *and* five licenses for v1+ being available, giving a total of nine licenses for "f1."

To illustrate counted vs. uncounted licenses, the following FEATURE line:

```
FEATURE f1 demo 1.0 1-jan-2001 uncounted HOSTID=DEMO \
  SIGN=123456789012
```

This feature has unlimited usage on any hostid, requires no license servers (no SERVER or VENDOR lines) and is therefore a complete license file by itself. This FEATURE line also happens to be an expiring license and will not allow use of the FEATURE after 1-jan-2005.

In contrast the following FEATURE line requires a vendor daemon named "demo" (and SERVER and VENDOR lines as well):

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

and is limited to five users on any host with an Internet IP address matching 195.186.*.*, and it never expires.

SEE ALSO

- Section 5.13.2, "Special FLEXlm Hostids"
- Section E.3.10, "ls_use_all_feature_lines"
- Section 9.2.4, "ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade"
- Section E.2.2, "LM_A_CRYPT_CASE_SENSITIVE"

5.6 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 from any old version (>= *from_feat_version*) and creates a new version with that same number of licenses.

For example, the two lines:

```
INCREMENT f1 demo 1.0 1-jan-2005 5 SIGN=9BFAC03164ED ck=3
UPGRADE f1 demo 1.0 2.0 1-jan-2005 2 SIGN=1B9A30316207 ck=23
```

would result in three licenses of v1.0 of "f1" and two licenses of v2.0 of "f1."

UPGRADE will operate on the closest preceding FEATURE or INCREMENT line with a version number that is >= *from_feat_version*, and < *to_feat_version*.

5.7 PACKAGE Lines

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

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

A PACKAGE line, by itself, does not license anything—it requires a matching 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. It may be more convenient for everyone to keep PACKAGE lines in a separate file, which is supported as of FLEX*lm* v6. The path to the package file should be specified in the application to support this transparently, via LM_A_LICENSE_DEFAULT.

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

where:

package	Name of the package. The corresponding FEATURE/INCREMENT 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 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 line. *num_lic* is only legal if OPTIONS=SUITE is not set—in this case the resulting number of licenses will be the count on the COMPONENTS line multiplied by the number of licenses in the FEATURE/INCREMENT line. Examples:
	`COMPONENTS="comp1 comp2 comp3 comp4"`
	`COMPONENTS="comp1:1.5 comp2 comp3:2.0:4"`

`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.
`SUPERSEDE [="p1 p2 ..."]`	Optional field, but if used, use with ISSUED date. Replaces all PACKAGE lines for the same package name with ISSUED dates previous 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 previous to *date*.

EXAMPLES

```
PACKAGE office demo 1.0 COMPONENTS="comp1 comp2" \
     OPTIONS=SUITE SIGN=123456789ABC
FEATURE office demo 1.0 permanent 5 SIGN=987654321FED
```

This is a typical suite example. The user will have two features: “comp1” and “comp2,” which are each version 1.0, with five non-expiring licenses available. When “comp1” or “comp2” is checked out, “office” will also be checked out. The vendor will most likely want to turn on duplicate

grouping (either through the FEATURE line or lc_checkout()) so that the same user can use “comp1” and “comp2” while using only one license of the “office” FEATURE.

```
PACKAGE office demo 1.0 \
     COMPONENTS="comp1 comp2 comp3 comp4 comp5" \
     SIGN=271FA0F72594
INCREMENT office demo 1.0 permanent 1 HOSTID=12345678 \
     SIGN=1B3147ADBC94
INCREMENT office demo 1.0 permanent 1 HOSTID=87654321 \
     SIGN=68B82E55A417
```

This is a good way to distribute multiple node-locked, counted licenses. Rather than requiring five INCREMENT lines per machine, only one INCREMENT line is required per machine, and the features are indicated in the PACKAGE line.

```
PACKAGE office demo 1.0 COMPONENTS="c1:1.5:2 c2:3.0:4 c3" \
     SIGN=A30483555898
FEATURE office demo 1.0 1-jan-2005 3 ISSUER=dist \
     SIGN=2C817A5100D8
```

The component versions override the feature versions, and the number of licenses available for any component is the product of the three licenses for “office” and the number of licenses of that component. The result is equivalent to:

```
FEATURE c1 demo 1.5 1-jan-2005 6 ISSUER=dist SIGN=7649CFAF16DB
FEATURE c2 demo 3.0 1-jan-2005 12 ISSUER=dist SIGN=63992D55345B
FEATURE c3 demo 1.0 1-jan-2005 3 ISSUER=dist SIGN=0D037EACC547
```

SEE ALSO

- Section 4.15, “LM_A_LICENSE_DEFAULT”

5.8 Comment Lines

Comment lines can begin with #. Currently, all lines not beginning with a license file keyword are comment lines. Therefore, license files can be sent as email messages.

5.9 Line Continuation

Lines can be continued with a “\” character.

5.10 Order of Lines in the License File

In v7+, licenses are automatically sorted internally so that many of the most common license order problems are avoided. The sort 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.

This order can be overridden by adding the `sort=`*nnn* attribute to any or all FEATURE/INCREMENT lines. The default is 100. Lines less than 100 are sorted before all lines without this attribute, and lines greater than 100 appear after all unmarked lines. All lines with the same number are sorted as they appear in the file. Therefore, 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().

5.11 Example License File

This example illustrates the license file for single vendor with two features, and a set of three server nodes, any two of which must be running for the system to function.

```
SERVER pat 17003456 27009
SERVER lee 17004355 27009
SERVER terry 17007ea8 27009
VENDOR demo
FEATURE f1 demo 1.0 1-jan-2005 10 SIGN=1AEEFC8F9003
FEATURE f2 demo 1.0 1-jan-2005 10 SIGN=0A7E8C4F561F
```

See the *FLEXlm Programmers Guide* and `examples/licenses` for examples of different types of license files.

5.12 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. There are notable exceptions, however, which are explained below.

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 user.

5.12.1 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. These are characters unsupported by FLEX*lm*, although some companies have used them.

5.12.2 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-2001 uncounted HOSTID=DEMO \
  SIGN=6E06CC47D2AB
```

Decimal format:

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

5.12.3 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.

5.12.4 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."

CK=

Leave this optional attribute off. The decimal format has its own built-in checksum. This attribute will only make the decimal format longer.

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*

5.13 Hostids for FLEX*lm*-Supported Machines

FLEX*lm* uses different machine identifications for different machine architectures. For example, all Sun Microsystems machines have a unique integer hostid, whereas all DEC machines do not. For this reason, the ethernet address is used on some machine architectures as the “Hostid”. An ethernet address is a six-byte quantity with each byte specified as two hex digits. Specify all 12 hex digits when using an ethernet address as a hostid. For example, if the ethernet address is 8:0:20:0:5:ac, specify “0800200005AC” as the hostid.

Integer hostids (used on Sun, SGI, HP, etc.) are normally hexadecimal numbers. However, a license file can take a decimal number if the hostid has a “#” prefix. Certain systems, notably HP `uname` and SGI, return decimal numbers by default, and this can make license file distribution easier, since you don’t have to convert to hex. Note that whenever a FLEX*lm* utility prints such a hostid, it always prints a hexadecimal number.

The default hostid for Windows systems is the ethernet address of the system. FLEX*lm* also supports several other hostids as well as hardware keys available from GLOBEtrotter Software.

The program `lmhostid` will print the exact hostid that FLEX*lm* expects to use on any given machine. See the following table of methods to obtain the hostid that FLEX*lm* requires for each machine architecture.

5.13.1 Expected FLEX*lm* Hostids

Hardware Platform	Hostid	Type this command on the license server:	Example
AIX (RS/6000, PPC)	32-bit hostid	`uname -m` (returns 000276513100), then remove last two digits, and use remaining last eight digits	02765131
DEC Alpha	ethernet address	`netstat -i`	080020005532
HP	32-bit hostid	`uname -i` and convert to hex, or prepend with #	778DA450 or #2005771344
	ethernet address	`lanscan` (station address without leading “0x”)	0000F0050185
Linux	ethernet address	`/sbin/ifconfig eth0` and remove colons from HWaddr 00:40:05:16:E5:25	00400516E525
SCO	Hostid String	`uname -x` (Serial is SCO00354), then prefix with “ID_STRING=”	ID_STRING=SCO00354
SGI	32-bit hostid	`/etc/sysinfo -s`, convert to hex, or prefix #	69064C3C or #1762020412
SUN	32-bit hostid	`hostid`	170a3472

Hardware Platform	Hostid	Type this command on the license server:	Example
Windows	ethernet address	`lmutil lmhostid`	00B0A9DF9A32
	Disk serial number	DIR C: (look for "Volume Serial Number is", and remove "-")	DISK_SERIAL_NUM= 3e2e17fd
	Dongle—parallel port hardware key	`lmhostid -flexid`	FLEXID=7-b28520b9

	Pentium III+ CPU, V7.0d+ only. Use BIOS Setup to enable.	`lmhostid -cpu`
		`lmhostid -cpu96`
		(The 32-bit version is the last nine characters from the full id.)
			9077-5D77-0002-57C8-95D2-1D3D (96-bit)
			95D2-1D3D (32-bit)

SEE ALSO

- Section 3.16, "lc_free_hostid()"
- Section E.1.1, "l_new_hostid()"
- Section 5.13, "Hostids for FLEXlm-Supported Machines"
- Section 5.13.4, "Intel Pentium III+ Hostid (HOSTID_INTEL)"

5.13.2 Special FLEX*lm* Hostids

FLEX*lm* contains a number of "special" hostid types which apply to all platforms. These hostid types can be used on either a SERVER line or a FEATURE line, wherever a hostid is required. These are:

ANY	Locks the software to any node (i.e., does not lock anything).
DEMO	Similar to ANY, but only for use with uncounted FEATURE lines.

DISK_SERIAL_NUM=*SN*	Locks the software to a PC with C drive serial number *SN*. (Windows only). Use this serial number as a hostid with caution. Large companies often purchase in bulk PCs which have cloned disks and therefore identical disk serial numbers. Also, sophisticated users have access to third-party tools which can alter disk serial numbers. It is relatively safe to use this as a hostid with home users or unsophisticated users at small companies.
DISPLAY=*display*	Locks the software to display *display*.
FLEXID=*SN*	Locks the software to a PC with a hardware key (dongle) of serial number *SN*. (Windows only).
HOSTNAME=*host*	Locks the software to computer host name *host*.
ID=*n*	Functionally equivalent to the “ANY” hostid—it will run on any node. The difference is that the license is unique and can be used to identify the customer. This hostid can be used to lock the license server (on the SERVER line) or the client (on the FEATURE/INCREMENT line). The number can have dashes included for readability—the dashes are ignored. Examples: ID=12345678 is the same as ID=1234-5678 is the same as ID=1-2-3-4-5-6-7-8
ID_STRING=*string*	Used on SCO systems for hostid.

INTERNET=*###.###.###.###*	Locks the software to an Internet IP address, or group of IP addresses. Wildcards are allowed. For example, 198.156.*.* means any host with a matching internet IP address. The main use is to limit usage access by subnet, implying geographic area. For this purpose, it would be used on the FEATURE/INCREMENT line, as a hostid lock.
USER=*user*	Locks the software to user name *user*.

EXAMPLES

```
FEATURE f1 demo 1.0 1-jan-2005 uncounted HOSTID=HOSTNAME=globes
     SIGN=AB28E0011DA1
```

or

```
FEATURE f1 demo 1.0 1-jan-2005 uncounted HOSTID=USER=joe \
     SIGN=EB78201163B0
```

SEE ALSO

- Section 5.13, “Hostids for FLEXlm-Supported Machines”
- Section 5.5, “FEATURE or INCREMENT Lines”
- Section 5.2, “SERVER Lines”

5.13.3 Vendor-Defined Hostids

FLEX*lm* allows you to specify your own vendor-defined hostid types. In order to do this, follow these steps (see `examples/vendor_hostid/`):

1. Write a C source file similar to this example (`hostids.c`):

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

extern LM_HANDLE *lm_job; /* This must be the current job! */

#define OURTYPE HOSTID_VENDOR+1 /* Next one would use +2 */
#define OURSTRING "EXAMPLE_HOSTID"
#define OUR_FIXED_ID "1234" /* This example returns only 1 hostid */
```

```
/*
 *x_flexlm_gethostid() - Callback to get the vendor-defined hostid.
 *         (Sorry about all the windows types for this function...)
 *
 */

HOSTID * LM_CALLBACK_TYPE
/*
 *     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();

        if (idtype == OURTYPE)
        {
             h->type = OURTYPE;

             strncpy(h->id.vendor, OUR_FIXED_ID, MAX_HOSTID_LEN);
             h->id.vendor[MAX_HOSTID_LEN] = 0;
             return(h);
        }
        return((HOSTID *) NULL);
}
void
x_flexlm_newid()
{
  LM_VENDOR_HOSTID h;

        memset(&h, 0, sizeof (h));
        h.label = OURSTRING;
        h.hostid_num = OURTYPE;
        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");
}
```

2. Register your hostid in the client application, and license generators (`lmcrypt`, `makekey`, etc.). This job must be named `lm_job`, because the job in the vendor daemon is called `lm_job`.

```
LM_HANDLE *lm_job;
...
lc_new_job(..., &lm_job); /* lc_init() in license generator */
x_flexlm_newid();
...
```

3. Modify `machind/lsvendor.c` thus:

```
...
ls_user_init1 = x_flexlm_newid;
...
```

4. Modify vendor daemon and application makefiles to include `hostids.c` (the above example).

5.13.4 Intel Pentium III+ Hostid (HOSTID_INTEL)

REQUIREMENTS:

- FLEX*lm* v7.0d+
- 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 `X`'s are uppercase hex 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.

Chapter 6

License Models

6.1 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.

6.1.1 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-2001 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, 2001. 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-2001 uncounted HOSTID=DEMO SIGN=AB1CC0916A06
FEATURE f1 corp 1.0 1-feb-2001 uncounted HOSTID=DEMO SIGN=ABDCC0116A06
FEATURE f1 corp 1.0 1-mar-2001 uncounted HOSTID=DEMO SIGN=BBDCA0D151ED
FEATURE f1 corp 1.0 1-apr-2001 uncoutned HOSTID=DEMO SIGN=BBDCB0E155F1
[...]
```

If the current date is February 1, 2001, then the salesperson would give an evaluator the third line, which expires in a month, March 1, 2001. The evaluator could simply save the FEATURE line in `license.dat` where the product was installed, 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 `license.dat` file 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-2001 uncounted HOSTID=DEMO \
     SIGN=AB1CC0916A06
```

The FEATURE line must appear after the PACKAGE line to work correctly.

6.1.2 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:

```
config = lc_auth_data()
```

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 eval 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 for this (not lc_get_config() or lc_next_conf()):

```
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 5.5, "FEATURE or INCREMENT Lines"
- Section 5.7, "PACKAGE Lines"
- Section 3.3, "lc_auth_data()"
- Section 4.2, "LM_A_CHECK_BADDATE"
- Section 9.2.2, "ls_a_check_baddate"

6.2 Lenient Licensing: Report Log and OVERDRAFT

More and more companies prefer licensing that does not deny usage, but bills customers for their usage.

6.2.1 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 GLOBEtrotter Software.

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.

6.2.2 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 checkout call, 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 5.5, “FEATURE or INCREMENT Lines”
- Section 4.29, “LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO”

Chapter 7

Distributing and Locating the License File

7.1 Emailing Licenses

Emailers can and do alter license files. We attempt to accommodate most emailer alterations, but not all are accommodated.

7.1.1 Newline Additions

Emailers often insert newlines into text, such as a license file. With v7+, this will not cause a problem. However, because of this enhancement, with v7+ it is now important that comments that appear between license file lines are prefixed with "#." Comments appearing before or after all lines do not require this (except the first line after the last FEATURE or INCREMENT line). Therefore, emails can be saved with email headers intact, and this is a good way to recommend saving a license file.

7.1.2 Adding ".txt" to the License File Name

When saving a text file, either in the emailer, or with Notepad on Windows, it's common that a `.txt` ending is appended, often with no notice or warning to the user.

Version 7+ FLEX*lm* ignores this suffix. That is, if a file called `demo.lic` is in the license path, and `demo.lic.txt` is found, it will be used. If both `demo.lic` and `demo.lic.txt` exist, both are used.

7.1.3 Other Transformations

QUOTES

ASCII quotation marks are sometimes substituted with other special characters. v7+ handles this correctly.

WORD FORMAT, RICH TEXT, ETC.

If the license is not saved as ASCII text, but turned into Word, Rich Text Format, or any other similar encoding, FLEX*lm* will not recognize the license file, and should be avoided.

7.2 Locating the License File

The rules that FLEX*lm* client applications use for finding the license file are:

1. If either `LM_LICENSE_FILE` or *`VENDOR`*`_LICENSE_FILE` (where *`VENDOR`* is the vendor name) environment variables is set, these are used instead of the default location. Note that environment variables can also be set in the registry (Windows) or in `$HOME/.flexlmrc` (UNIX). If set in both locations, both are used.
2. If both *`VENDOR`*`_LICENSE_FILE` and `LM_LICENSE_FILE` are set, both are used instead of the default location, with *`VENDOR`*`_LICENSE_FILE` used first.
3. If application sets lc_set_attr(`..., LM_A_DISABLE_ENV...`), then environment variables are ignored. Not recommended except in license file lists.
4. The license location(s) can be set in the application with LM_A_LICENSE_FILE or LM_A_LICENSE_FILE_PTR or LM_A_LICENSE_DEFAULT. If any of these are set, the default location is ignored. LM_A_LICENSE_DEFAULT is normally recommended, because it automatically recognizes the environment variables plus the indicated license path(s). LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR will set the path if the environment variable is either not set or is disabled.
5. In the FLEXible API, the license file location cannot be changed once the license file is read. The license file is not read until one of the following functions is called: lc_checkout(), lc_get_config(), lc_next_conf(), lc_userlist(). The only way to effectively change the license file once it has been read, is to start a new job with lc_new_job(). That new job will read the new or modified license when required to.
6. Calling any of lc_set_attr(`..., LM_A_LICENSE_*, ...`) more than once overrides the previous setting. For example,

```
lc_set_attr(..., LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)lic_path1);
lc_set_attr(..., LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)lic_path2);
```

Only *lic_path2* is used.

```
lc_set_attr(..., LM_A_LICENSE_FILE, (LM_A_VAL_TYPE)lic_path1);
lc_set_attr(..., LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)lic_path2);
```

Again, only *lic_path2* is used.

7. For the Simple and Trivial APIs, the rules are the same, with the rule that the license path argument to the checkout call behaves like LM_A_LICENSE_DEFAULT.
8. The `-c` option will override the setting of `LM_LICENSE_FILE` for all FLEX*lm* utilities such as: `lmgrd`, `lmdown`, `lmstat`, etc.

7.3 License Specification

Wherever a license path can be specified, it can consist of:

- A single file.
- A list of files, separated by a colon on UNIX and a semi-colon on Windows.
- A directory, where *dir*`/*.lic` are used in alphabetical order, as if specified like a license file list. On Windows, case doesn't matter and `*.LIC` files are also recognized. On UNIX, case does matter and `*.LIC` files are not recognized.
- `@`*host*, where *host* is the host name of the license server, when the SERVER has no port number, or the port number is between 27000 and 27009 (introduced in v6—unsupported in older versions).

  `@localhost` will always work if the server is running on the same system as the client.
- *port*`@`*host*, where *port* is the port number and *host* comes from the SERVER line.
- The actual license file text, with `START_LICENSE\n` as a prefix, and `\nEND_LICENSE` as suffix, where the embedded newlines are required.

7.3.1 Using License File List for Convenience and Redundancy

Client programs can process a series of license files, for example, by setting `LM_LICENSE_FILE` to a path, as in:

```
% setenv LM_LICENSE_FILE file1:file2:..:dir1:..:filen
```

Client programs will then try using *`file1`*; if it fails, *`file2`* will be tried, etc. Directories are automatically expanded to use all files matching `*.lic` in that directory as part of the list. On UNIX, the license files are separated by colons; on Windows, the license files are separated by semi-colons.

Aside from being convenient, this is an important method of redundancy, and has many advantages over the more formal three-server redundancy. License file list redundancy can also be used in combination with three-server redundant systems.

A non-redundant server could be specified as `@`*`host`* or *`port`*`@`*`host`*, and each server of a set of three redundant servers should be specified as *`port`*`@`*`host`* (not just `@`*`host`*).

For example, if you have a single server node named “serverhost,” and you are running FLEX*lm* on port 27000, you could specify your license file as:

```
@serverhost
```

or

```
27000@serverhost
```

You could have a license file path which looked like the following:

```
@serverhost:/usr/local/license.dat:/myprod/licensedir:27000@shost2
```

If in the license file list there is a set of three redundant servers, `1700@host1`, `1700@host2`, and `1700@host3`, the path might look like:

```
@serverhost:1700@host1:1700@host2:1700@host3:27000@shost2
```

Note: Unless `@`*`host`*, *`port`*`@`*`host`*, or USE_SERVER are used (so the client doesn’t read the FEATURE lines), both the client and server need to be reading the *same* license file, because the client passes the signature from the FEATURE line to the vendor daemon.

SEE ALSO

- Section 3.29, “lc_set_attr()”
- Chapter 10, “Debugging Hints”

7.3.2 License in a Buffer

The license file does not need to be located on disk—it can be specified in the program itself. Any place a license path can be set can be a license file instead, as in this example:

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

Note that the license begins with `START_LICENSE\n` and ends with `\nEND_LICENSE`. The embedded newlines are required. A license like this can be specified in place of a license path wherever a license path is valid. This can also be a license file list; as in the following example:

```
lc_set_attr(job, LM_A_LICENSE_DEFAULT,(LM_A_VAL_TYPE)
     "path/to/license.dat:\
     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.dat` is first in the list, followed by the license in the string.

License in a buffer is particularly useful when selling libraries, and end-user royalties are not required. Since all end users for a particular ISV will have the same license file, it’s convenient to store it in a character buffer in the program, rather than in a license file, which would require the ISV to distribute an extra file that might get misplaced.

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

Chapter 8

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`.

8.1 Starting lmgrd on UNIX

The command-line syntax for `lmgrd` is:

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

where:

`-c` *`license_file_list`*	Use the specified license file(s). If a directory is specified, all matching `*.lic` files are used. The list is colon-separated on UNIX and separated by semi-colons on Windows. If redundant servers, must be a single license file. SERVER line hostids for all files must apply to the same host, but the hostids need not be identical.

`-l` *`debug_log_path`*	Write debugging information to file *`debug_log`*. This option uses the letter `l`, not the numeral 1. The default output location is stdout. See the *FLEXlm End Users Guide* for descriptions of the messages in the debug log file.
`-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`.
`-x lmdown`	Disallow 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 Task Manager or NT Service (Windows). On UNIX, be sure the `kill` command does not have a `-9` argument.
`-x lmremove`	Disallow the `lmremove` command (no user can run `lmremove`).
`-v`	Prints `lmgrd` version number and copyright and exits.

`-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.

Note: The license file path name 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`.

8.2 Starting lmgrd on Windows

`lmgrd` can be started as an application from the Windows NT console. For example:

```
D:\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 the NT, and on Windows 95, `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, `lmtools` or the utility program, `installs.exe` (located in the `i86_n3` directory). Using `lmtools` to install `lmgrd` as a service is the recommended technique (see the *FLEXlm Programmers Guide*). If you prefer to do a manual installation of `lmgrd` as a service, see Section 12.10, "Manually Installing lmgrd as a Service."

8.3 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 GLOBEtrotter Software 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 Chapter 10, "License Servers," in the *FLEXlm Programmers Guide* for recommendations about configuring license server machines.

Chapter 9

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.

9.1 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).

9.2 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.

9.2.1 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, or 7_1. The default (0) is LM_BEHAVIOR_CURRENT, which is V7_1 in version 7.1.

SEE ALSO

- Section 4.1, “LM_A_BEHAVIOR_VER”

9.2.2 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 4.2, “LM_A_CHECK_BADDATE”
- Section 6.1.2, “Limited Functionality Demos”

9.2.3 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 4.14, “LM_A_LICENSE_CASE_SENSITIVE”

9.2.4 ls_compare_vendor_on_increment and ls_compare_vendor_on_upgrade

```
(int) ls_compare_vendor_on_increment = 0; /*p Compare vendor-defined */
(int) ls_compare_vendor_on_upgrade = 0; /* Compare vendor-def fields */
```

If VENDOR_STRING is used in your license files, then these two variables may need to be modified. If one is set, set both.

INCREMENT lines are combined if the following is true:

- The feature names match
- The feature versions match
- Any node-lock hostid, if present, matches
- USER_BASED, HOST_BASED, and CAPACITY matches
- Optionally, the vendor-defined strings match

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 vendor string must match; if 0, then no comparison is done on the vendor string.

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 vendor string must match; if 0, then no comparison is done on the vendor string.

SEE ALSO

- Section 5.5.22, “VENDOR_STRING”
- Section 5.6, “UPGRADE Lines”

9.2.5 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.

9.2.6 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 9.2.10, “ls_outfilter.”

9.2.7 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 9.2.10, “ls_outfilter.”

9.2.8 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).

9.2.9 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.

9.2.10 ls_outfilter

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

Note: Please contact GLOBEtrotter 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, server,
     dup_sel, linger, sign, 0,0);
```

PARAMETERS

`(char *) feature`	Feature desired.
`(char *) num_lic`	Number of licenses.
`(char *) wait`	"Wait until available" flag if (`*wait == '1'`), the request will be queued if a license is not available.
`(CLIENT_DATA *) who`	The user.
`(char *) version`	Version number of feature.
`(SERVERNUM) server`	Server requesting checkout.
`(char *) dup_sel`	Duplicate license selection criteria.
`(char *) linger`	How long the license is to linger.
`(char *) sign`	Signature from FEATURE line.

RETURN

0 -> checkout not available, > 0 -> checkout done, < 0 -> request queued.

Note: ls_get_attr() can be used to retrieve all the parameters that ls_checkout() requires.

9.2.11 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.

9.2.12 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.

9.2.13 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.

9.2.14 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.

9.2.15 ls_user_lockfile

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

The vendor daemons use a lock file to prevent multiple copies from running on a license server host. The lock file names are (where *`vendor`* is the vendor daemon name, as on the VENDOR line in the license file):

UNIX	`/usr/tmp/lock`*`vendor`*. On some newer systems, including DEC Alpha, the location is `/usr/tmp/.flexlm/.lock`*`vendor`*.
Windows	`C:\flexlm\`*`vendor`*

If ls_user_lockfile is NULL, or points to a null string, the default lock file will be used.

The date on the lock file is updated every six hours to make it less likely that cron jobs will remove it.

If you wish to change the location of the lock file, set ls_user_lockfile to the new location. Be sure to use a full path name for this file (i.e., on UNIX, the path name should start with /); otherwise, multiple vendor daemons could be run from different directories.

9.2.16 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 will process 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 3.33, “lc_vsend()”

Chapter 10

Debugging Hints

10.1 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 11, “UNIX Platform-Specific Notes,” or Chapter 12, “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* timers with LM_A_CHECK_INTERVAL set to -1 with lc_set_attr() and call lc_timer() periodically.
- On UNIX, FLEX*lm* installs a handler for SIGPIPE and SIGALRM. If your application uses FLEX*lm* timers 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.
- On UNIX, FLEX*lm*, by default, uses SIGALRM to check the health of the connection. This cannot be tolerated by certain applications (for example, applications that use XView or FORTRAN). These applications should set the LM_A_CHECK_INTERVAL and LM_A_RETRY_INTERVAL attributes to -1 with lc_set_attr(). After checking out a license, the application must periodically call lc_timer() to keep checking the health of the connection.
- 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 reentrancy 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 FLEX*lm* timers and call lc_timer() directly.

SEE ALSO

- Section 3.21, “lc_heartbeat()”

10.2 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 encryption 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.

If your customer has this error, use the `lmcksum` command to locate the line that was mis-typed.

10.3 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. There are compatibility issues that may arise with this method if some vendor daemons and/or applications are older than v6.
- 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 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.

SEE ALSO

- Chapter 7, “Distributing and Locating the License File”

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)."

10.4 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 most recent (highest version number) `lmgrd` and utilities should be used. The version of `lmgrd` must be greater than or equal to the version of the vendor daemon and the version of the vendor daemon must be greater than or equal to the version of client application that incorporates FLEX*lm*.

To determine the version of any FLEX*lm*-based product, use the following command:

```
% lmver program_name
```

On UNIX systems, you can also use:

```
% strings program_name | grep Copy
```

Chapter 11

UNIX Platform-Specific Notes

11.1 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.

11.2 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.

11.3 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):

Intel Linux Redhat v6; Caldera v2.3+	`i86_r6.tar`
Intel Linux Glibc 2.3; Redhat v5.x:	`i86_g2.tar`
Intel Linux Libc 1.x:	`i86_l1.tar`

- Caldera v1.x and v2.0-2.2
- Redhat v4.x

GLOBEtrotter 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 GLOBEtrotter 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.

11.4 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 5	Started shipping early ‘90s; it is similar to SVR4, uses shared libraries, and is 32-bit (o32 object files).
IRIX 6	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(?).
MIPS3	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

`-o32`	Native to IRIX 5, it is the “old 32-bit object” format.
`-n32`	Native to IRIX 6, it is the “new” 32-bit format.
`-64`	Native to IRIX 6; 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 three SGI directories:

`sgi32_u5`	32-bit (`-o32`) IRIX-5, MIPS1. Requires FLEX*lm* sgi vendor key.
`sgi32_u6`	32-bit (`-n32`) IRIX-6, MIPS3 and MIPS4. `liblmgr.a` is MIPS3 and `liblmgr_n32mips4.a` is MIPS4. Requires FLEX*lm* sgi vendor key.
`sgi64_u6`	64-bit (`-64`) IRIX-6, MIPS3 and MIPS4. Requires FLEX*lm* sgi64 vendor key. `liblmgr.a` is MIPS3 and `liblmgr_64mips4.a` is MIPS4.

FLEX*LM* VENDOR KEYS FOR SGI

sgi	All SGI 32-bit applications, including `sgi32_u*`.
sgi64	All SGI 64-bit applications, including `sgi64_u*`.

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.*

11.5 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 GLOBEtrotter Software) and `ENCRYPTION_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.

UDP communications are not supported because of an apparent flaw in the SCO OS.

Chapter 12

Windows Platform-Specific Notes

FLEX*lm* supports the Windows platforms using two sets of 32-bit libraries (`lmgr.lib` and `lmgr327b.dll`). The 32-bit libraries supports clients and servers on Windows NT 3.5, 3.51, 4.0 and Windows 95/98/2000.

12.1 Supported C Compilers

The FLEX*lm* client library on Windows NT is implemented as a static library or a DLL. The DLL can interface with almost any compiler, but is less secure. However, to build your vendor daemon on a Windows system, FLEX*lm* supports only the Microsoft Visual C++ compiler 5.0 or greater.

In order for FLEX*lm* API include files to compile properly on Windows platforms, two compile-time flags must be defined: `PC`, and `_WINDOWS`. On Windows NT systems, (32-bit) an additional compile time flag, `WINNT`, must be defined. This flag will be used to include proper macro definitions in `lmclient.h` for FLEX*lm* on Windows systems.

12.2 Using Languages Other Than C

There is a FLEX*lm* API that is designed for non-C languages such as Visual Basic to eliminate the usage of pointers. It is documented in the `examples/vb/vb4.0/Visual_Basic.doc`.

12.3 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 multi-threading 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.

  GLOBEtrotter Software also provides a library compiled with /MD, multi-threaded, 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 `lmgr327b.dll` with its associated import library `lmgr327b.lib`. The `lmgr327b.dll` library is built using the multi-threaded 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@globes.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 4.32, “LM_A_WINDOWS_MODULE_HANDLE.”

12.4 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)
```

12.5 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.

12.6 Hardware Hostids (Dongles)

The software for the various hardware based hostid's are stored in the `FLEXID7` and `FLEXID8` directories. Consult those locations for more details.

A hostid that contains FLEXID indicates a dongle hostid. It has the form `FLEXID=`*`n`*`-`*`xxxxxxxx`*, where *n* indicates which dongle type is being used.

FLEXID=7-...

In 16-bit modes, you will need to use `SUPERPRO.DLL` (4.032 bytes, dated 5-8-1995).

For operating under Windows NT, you will need to install a set of NT drivers, `SENTTEMP.HLP`, and `SENTTEMP.SYS`.

FLEXID=8-...

For use in 32-bit mode on Windows 95, `VSAUTHD.386` must be installed.

For use on NT in 32-bit mode, the `DS1410D.SYS` driver must be installed.

12.7 FLEX*lm* TCP/IP Network Problem

The Microsoft TCP connect() behavior has an important bug for which there is no fix. The Microsoft TCP implementation does not allow the programmer any effective control over the duration of a connect call, timeouts, retry attempts, etc.

The Microsoft TCP connect behavior causes needless delays under the following condition: when clients attempt to talk to a license server, and the server node is running, but `lmgrd` is not, there is a 1.5 second delay on Windows, and this delay does not occur on other operating systems.

In FLEX*lm*, this is most notable when using the environment variable `LM_LICENSE_FILE` set to `@`*`host`*, and *`host`* is up, but `lmgrd` is not running. This causes a delay up to 15 seconds, 1.5 seconds for the attempted connect to each of the ten default ports, 27000-27009.

12.8 Environment Variables (32-Bit Platforms)

When running Windows applications, it is sometimes difficult to set environment variables. On NT, because each user can have his own environment, it is sometimes confusing as to which variables are set. You can now either set environment variables in the traditional way, i.e., the `set` command in `autoexec.bat`, the Control Panel→System→Environment (NT), or by using the registry. The priority is set to favor normal environment

variables over registry entries. To set an environment variable using the registry, make an entry in `HKEY_LOCAL_MACHINE→SOFTWARE→FLEXlm License Manager` as a String Value. Any FLEX*lm* application will then see this in its environment. This will be especially useful for setting the license file (if not using the default) using `LM_LICENSE_FILE`, or other parameters when the server is running as an NT Service.

When combining license files as a license file list in `LM_LICENSE_FILE`, use " ; " instead of " : ," e.g., `file1;file2`.

SEE ALSO

- Section 3.30, "lc_set_registry()"

12.9 Server Environment Variables

Many aspects of FLEX*lm* can be controlled using environment variables. These are mentioned in the Reference Manual. There are limitations to them depending on the platform. You can set these environment variables before entering Windows 95/98 in the `autoexec.bat` files.

We have allowed an alternate way of setting environment variables, registry entries. When FLEX*lm* looks for a 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.

If it finds it, it uses that value as the value of the environment variable.

In this version of FLEX*lm*, a new function was added to simplify this procedure. The function is lc_set_registry() (see Section 3.30, "lc_set_registry()."

This function allows you to write into the registry (assuming your program has the appropriate security attributes). If you do not, the error returned is -68, LM_NOADMINGROUP.

The Server software (`lmgrd.exe` and your vendor daemon) can also use registry values when they are started as a SERVICE on a NT system. 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);
```

12.10 Manually Installing lmgrd as a Service

To install `lmgrd` as a service manually (rather than using the supplied `lmtools`), use the `installs.exe` command provided by FLEX*lm*. For example:

```
D:\flexlm> installs d:\flexlm\lmgrd.exe
```

Note that the full path name to `lmgrd.exe` must follow the `installs` command. You must also make sure that the full path name including drive letter for your vendor daemon is specified correctly in your license file.

After `installs.exe` is run successfully, `lmgrd` is installed as a Windows NT service and will be started automatically each time your system is booted. To start `lmgrd` right after running `installs.exe` without rebooting your system, you may use the Service icon from the Windows NT Control Panel. Look for the FLEX*lm* licensing service from the dialog after you double-click the Service icon.

To remove `lmgrd` from the registered service list, simple type `installs remove`. If you wish to customize the `installs` program, the source code is included in the `machind` directory.

When `lmgrd.exe` is installed as a service on your system, you may use the following procedure to set the license file path for `lmgrd` by updating the system registry:

1. Install `lmgrd.exe` as a service as described previously.
2. Run Registry Editor, `system32\regedt32.exe`.
3. Select HKEY_LOCAL_MACHINE:SYSTEM:

   CurrentControlSet:Control:Session Manager:Environment
4. From the menu bar, select Edit and then Add Value.
5. When the Add Value dialog box is displayed, enter `LM_LICENSE_FILE` as Value Name, and select REG_SZ as Data Type. Click OK.
6. At this point, the String Editor dialog box should appear. Enter the full path to the license file that you wish to select. Click OK.
7. Exit the Registry Editor.
8. Shut down your system and restart. `lmgrd.exe` should now be using the selected license file. To verify this, you can view `lmgrd.log` in the `system32` directory.

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 ISV 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 the *FLEXlm Programmers Guide* 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 3, “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."

Caution: 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, GLOBEtrotter 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

UDP Communications

FLEX*lm* supports the UDP connectionless transport, in addition to the default TCP connection-based transport. The use of UDP is normally not recommended.

TCP, the default, is recommended for the following reasons:

- With TCP, the server knows immediately when the client exits. It is therefore not essential that the client call CHECKIN().
- In our tests, TCP and UDP behave identically with regards to speed.
- The only drawback to TCP is that the vendor daemon process requires a file descriptor per client. On modern OSs, a single process can handle 1000 or more file descriptors by default, and this number can usually be increased with a kernel parameter. Therefore this is not usually a drawback of any consequence.

B.1 How to Select UDP Connections

UDP can be selected by the application or by the end user, in the following ways:

- The application can call:

```
lc_set_attr(LM_A_COMM_TRANSPORT, LM_UDP);
```

- The end user can set the comm transport with the FLEXLM_COMM_TRANSPORT environment variable:

```
% setenv FLEXLM_COMM_TRANSPORT UDP
```

The application can prevent the user from setting the TRANSPORT mode via:

```
lc_set_attr(LM_A_ALLOW_SET_TRANSPORT, 0);
```

The environment variable setting is disabled with this call.

FLEX*lm* defaults to TCP, and all the above options can be set to TCP. The order of precedence is (higher number takes precedence over lower number):

1. Default TCP
2. Environment variable specified — `FLEXLM_COMM_TRANSPORT`
3. LM_A_ALLOW_SET_TRANSPORT == false. Environment variable is disabled.
4. Set in application — LM_A_COMM_TRANSPORT
5. Vendor daemon is pre-v3.0 — TCP only.

B.2 UDP Behavioral Differences

- Servers that use UDP clients require a small, fixed number of sockets. This can be preferable on systems with limited resources and a large number of clients (500 or more). In addition, they never need to spawn additional vendor daemons, as TCP servers do when they use up the maximum number of file descriptors.
- When a UDP client exits without doing a checkin, the server does not become immediately aware of this, but will time the license out. Therefore, applications should always call lc_checkin() before exiting. The default for a timeout is 45 minutes, but can be set in the application via:

  ```
  lc_set_attr(LM_A_UDP_TIMEOUT, (LM_A_VAL_TYPE)num_sec);
  ```

  Therefore, a UDP client which does not, or is unable to, call lc_checkin(), will have a license checked out for 45 minutes (by default) after the program exits. This behavior will also be affected by LM_A_CHECK_INTERVAL. In particular, for UDP clients, LM_A_CHECK_INTERVAL must never be longer than LM_A_UDP_TIMEOUT, or the client will be timed out for no good reason. This is not normally fatal, but a client could lose a license to someone else in trying to retrieve it after being timed out.

  Applications that call lc_timer() directly must ensure that they call it often enough to prevent being timed out.
- When a client has to reconnect to a server (which can happen for numerous reasons), the reconnection process requires ten seconds for the client to timeout the read from a server which is down. With TCP, this recognition is instantaneous. The result is that a server's failure will cause a client using UDP to hang for a minimum of ten seconds, while with a TCP client, this recognition is transparent.

In GLOBEtrotter Software's testing environment, we have found no discernible performance advantage in checking out licenses or getting status information via either TCP or UDP.

Based on our test results, we find TCP to be a far superior mode of communication, and should be used wherever possible.

SEE ALSO

- Section 3.29, "lc_set_attr()"
- Section 3.21, "lc_heartbeat()"
- Section 4.3, "LM_A_CHECK_INTERVAL"
- Section 5.13.2, "Special FLEXlm Hostids"

Appendix C

FLEX*lm* Limits

Any limitations such as string lengths are listed here. Items that are unlimited are also listed for clarification.

C.1 License File Limits

The limits on names for the major parameters employed in the FLEX*lm* license file are:

Host name length	32 characters
FEATURE name length	30 characters
FEATURE/INCREMENT/ UPGRADE/PACKAGE line length	2048 characters
VENDOR name length	10 characters
Version	10 characters, in floating point format, e.g., `123.4567`, or `2.10`
Latest expiration date	31-dec-9999 (but we recommend using “permanent” instead)
Number of users	32-bit integer
Number of FEATURE/INCREMENT/ UPGRADE lines	Unlimited
Number of VENDORs	Unlimited
Number of SERVERs	3 (Redundant server licenses are limited to 3 servers)

OVERDRAFT	32-bit integer
HOST_BASED=*n*	32-bit integer
USER_BASED=*n*	32-bit integer
MINIMUM=*n*	32-bit integer
Other optional FEATURE attributes	Limited only by the total length of the FEATURE line.

C.2 Decimal Format License Limits

Max readable length that can be converted to decimal	Approximately 100 characters. 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.

C.3 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.

C.4 lc_set_attr() limits

LM_A_DISPLAY_OVERRIDE	32 characters
LM_A_HOSTNAME_OVERRIDE	64 characters
LM_A_USERNAME	20 characters
LM_A_CHECKOUT_DATA	32 characters
LM_A_CHECK_INTERVAL	>20 seconds

C.5 Other API Limits

Vendor-defined hostid length	40 — including the *NAME* prefix.
Number of licenses in one lc_checkout() request	9999
Long error message length	1024 characters (length of string returned from lc_errstring())

C.6 Vendor Daemon Limits

Number of Clients Per Vendor Daemon

When using TCP, a single vendor daemon can support as many clients as the system limit for file descriptors and sockets, which varies from around 250 on sunos4 (the only known system with a limit this low) to 4000 on OSF/1 (SCO comes with a configurable default of around 8).

In practice, we encourage end-users to put servers on systems configured with enough file descriptors per process to support the number of end users connecting to the vendor daemon, which may require reconfiguring the kernel to increase the number of file descriptors per process.

Nearly all systems can handle 250 clients per vendor daemon without performance problems, and large systems can easily support over a thousand.

When using UDP, there is no limit to the number of clients. Note that multiple daemons can be run on a single network, making the number of even 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.

C.7 lmgrd Limits

`lmgrd` processes per node	Unlimited
Default port number range	27000-27009
License files per `lmgrd` process	Unlimited

C.8 Subnet, Domain, Wide-Area Network Limits

FLEX*lm* has no limitations regarding subnets (becacuse 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.

C.9 LM_LICENSE_FILE, *VENDOR*_LICENSE_FILE

Number of licenses in path	Unlimited

Appendix D

FLEX*lm* Status Return Values

D.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.
-37 LM_BADFEATPARAM	"Duplicate selection mismatch for this feature" The checkout request for this feature has specified a duplicate mask that does not match the mask specified by an earlier checkout. This is probably the result of using different versions of your client software, or from having an uninitialized variable in the dup_group field for lc_checkout().

-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 option 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 option 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 GLOBEtrotter Software.
-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 GLOBEtrotter Software 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 lm_code.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.
-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 Globetrotter 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 ISV'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 5.5, "FEATURE or 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.
-86 LM_OBJECTUSED	"This license object already in use" Java only. A second checkout against a license object generates this error. If multiple checkouts are needed, multiple license objects need to be created.
-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	"ENCRYPTION_SEEDs are non-unique" Returned from lc_new_job() or lp_checkout() when vendor name is not demo, but seeds are default 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 5.12, "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 in 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.
-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. These drivers are available at http://www.globetrotter.com or from your software vendor.
-112 LM_NODONGLEDRIVER	"Missing Dongle Driver" In order to read the dongle hostid, the correct driver must be installed. These drivers are available at http://www.globetrotter.com or from your software vendor.
-113 LM_FLEXLOCK2CKOUT	"FLEXlock checkouts attempted" Only one checkout is allowed with FLEX*lock*-enabled apps. 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.

Appendix E

Rarely Used Functions and Attributes

The functions, attributes, variables, and features listed in this section are obsolete or rarely needed. Contact GLOBEtrotter technical support (support@globes.com) before using any of them—they can cause problems if inappropriately used.

E.1 Rarely Used FLEXible API Functions

The following functions are more rarely needed, and many exist only for compatibility with earlier FLEX*lm* versions. They should be used with care, and questions are welcomed before their use.

E.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 4.30, “LM_A_VENDOR_ID_DECLARE”
- Section 5.13.3, “Vendor-Defined Hostids”

E.1.2 lc_baddate()

Obsolete and no longer needed. This check is now automatically performed when a feature is expiring and LM_A_CHECK_BADDATE and ls_a_check_baddate are set.

SEE ALSO

- Section 4.2, “LM_A_CHECK_BADDATE”
- Section 9.2.2, “ls_a_check_baddate”

E.1.3 lc_ck_feats()

SYNTAX

```
status = lc_ck_feats(job, vendor)
```

DESCRIPTION

Checks the FEATURESET line for a given vendor.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(char *) vendor`	The vendor daemon to check.

RETURN

`(int) status`	Status of the FEATURESET line: 1 if OK, 0 if bad.

ERROR RETURNS

LM_NOFEATSET	No FEATURESET line found for this vendor.
LM_CANTCOMPUTEFEATSET	Cannot compute FEATURESET code for this vendor.

LM_BADFEATSET	The code on the FEATURESET line is incorrect.
LM_FUNCNOTAVAIL	Vendor keys do not support FEATURESET.

SEE ALSO

- Section E.3.11, “ls_use_featset”
- Section E.4.2, “FEATURESET Line”
- Section 2.6, “Multiple Jobs”

E.1.4 lc_cleanup()

SYNTAX

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

DESCRIPTION

Normally, after the calling program finishes its use of the FLEX*lm* library or DLL, the calling program exits. At this point, all handles, malloced memory, and threads are released or terminated. However some applications do not operate in this manner. To provide a method of cleaning up, FLEX*lm* has added a new function, lc_cleanup(). It can only be called after you have called lc_free_job() for all jobs that you have used. This function will free all malloced memory, release all handles, and kill all threads that it has created.

For normal Windows programs, this call is optional. Only special drivers which reside dormant in memory require this call. Do not call any FLEX*lm* functions after this call.

E.1.5 lc_copy_hostid()

SYNTAX

```
copy = lc_copy_hostid(job, orig)
```

DESCRIPTION

Returns a copy of a hostid list, and allocates memory as necessary. Using lc_hostid(), this function should not be needed.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(HOSTID *) *orig*	Hostid list to be copied.

RETURN

`(HOSTID *) copy`	Copy of *orig*, or 0 upon failure. Memory was allocated as needed.

ERROR RETURNS

LM_CANTMALLOC	Out of memory.

SEE ALSO

- Section 3.16, “lc_free_hostid()”

E.1.6 lc_crypt()

SYNTAX

```
CONFIG conf;
char *sdate;
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
/*...*/
LM_CODE_GEN_INIT();
enc_code = lc_crypt(job, &conf, &sdate, &code);
```

DESCRIPTION

Note: lc_cryptstr() is the recommended function to generate license files.

Computes the license key for a FLEX*lm* feature line. lc_crypt() is the older form of the FLEX*lm* authentication routine—lc_crypstr() is now the preferred method. We strongly encourage converting to lc_cryptstr(), and support for lc_crypt() is limited.

lc_crypt() takes input parameters and creates the license key that appears in the license file. Vendors generally will not call lc_crypt() directly, unless they are writing a custom license generation program, in which case lc_cryptstr() is preferred.

The `CONFIG *` parameter should be a pointer to a struct that has been correctly filled in. To do so, first make sure it is set to zeroes with `memset(&conf, 0, sizeof(conf))`. Then fill in each item in the struct, using the definition as it appears in `lmclient.h`. Note that many items are now optional and do not need setting.

The `sdate` parameter can be obtained by calling l_bin_date() with the date string, e.g.,

```
l_bin_date("1-jan-2001");
```

To obtain the start date from a license file license key (the license key on the FEATURE line), use l_extract_date():

```
char code[21];
l_extract_date(code);
```

PARAMETERS

`(LM_HANDLE *)` *`job`*	FLEX*lm* job, from lc_new_job().
`(CONFIG *)` *`conf`*	Filled-in `CONFIG` struct pointer.
`(char *)` *`sdate`*	Start date, in coded format. See above.
`pointer to (VENDORCODE) code`	From LM_CODE() macro. (With v7.1, *do not* XOR `code.data[0]` and `code.data[1]` with *`VENDOR_KEY5`*.)

RETURN

`(char *)` *`enc_code`*	The license key, which should match the license file; 0 if unsuccessful.

ERROR RETURNS

LM_CANTMALLOC	malloc() call failed.
LM_LGEN_VER	Attempt to generate license with incompatible attributes for the specified version of FLEX*lm*.

LM_BADPARAM	Parameters and job attributes are inconsistent.
LM_BADDATE	Invalid start date.

SEE ALSO

- Section 3.9, “lc_cryptstr()”

E.1.7 lc_disconn()

SYNTAX

```
status = lc_disconn(job, flag)
```

DESCRIPTION

Drops the connection to the server. A count of “logical” connections is maintained and if other features are active the connection is maintained, unless *flag* is non-zero.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(int) *flag*	Non-zero to force disconnection.

RETURN

(int) *status*	<0 — error == 0 — success > 0 — # of “logical” connections remaining.

E.1.8 lc_display(), lc_hostname(), lc_username()

SYNTAX

```
display_name = lc_display(job, flag)
hostname = lc_hostname(job, flag)
username = lc_username(job, flag)
```

DESCRIPTION

Returns environment information about the current process.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(int) *flag*	0 — Return system's idea of value. != 0 — Return FLEX*lm*'s idea of value.

RETURN

One of:

(char *) *display_name*	Display name.
(char *) *hostname*	Host name.
(char *) *username*	User name.

E.1.9 lc_feat_set()

SYNTAX

line = lc_feat_set(*job*, *vendor*, *code*, *codes*)

DESCRIPTION

Computes the FEATURESET *code* for the specified *vendor*, if *codes* is NULL. If *codes* is a pointer to a char *, the pointer is set to an array of license keys for each FEATURE line for this *vendor*.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().
(char *) *vendor*	The vendor daemon desired.
(VENDORCODE *) *code*	From lc_new_job().
(char **) *codes*	Concatenated license keys (returned). If a pointer to a char * is passed, this gets set to a string which is a concatenation of the license keys for each FEATURE line for this vendor daemon—to be used for calculating a checksum using your own checksum algorithm.

RETURN

(char *) *line*	The FEATURESET line for the license file, or NULL for error.

ERROR RETURNS

LM_CANTMALLOC	malloc() call failed.

SEE ALSO

- Section E.3.11, “ls_use_featset”
- Section E.4.2, “FEATURESET Line”

E.1.10 lc_get_errno()

SYNTAX

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

DESCRIPTION

The most recently set FLEX*lm* error is obtainable via lc_get_errno(). This can be used after any FLEX*lm* function. lc_err_info() is now recommended instead, because it includes full error information.

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 3.10, “lc_err_info()”
- Section 3.28, “lc_perror()”
- `lmclient.h`

E.1.11 lc_get_feats()

SYNTAX

`fs_code` = `lc_get_feats(`*`job`*, *`vendor`*`)`

DESCRIPTION

Gets the license key from the FEATURESET line for the specified vendor.

PARAMETERS

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

RETURN

`(char *)` *`fs_code`*	The code from the FEATURESET line for this vendor.

ERROR RETURNS

LM_NOFEATURE	FEATURESET line for requested vendor cannot be found.

SEE ALSO

- Section E.4.2, “FEATURESET Line”

E.1.12 lc_gethostid()

SYNTAX

```
hostid = lc_gethostid(job)
```

DESCRIPTION

lc_hostid() should normally be used instead.

Returns the hostid for the local host. lc_gethostid() is simply a call to lc_getid_type(*default_hostid_type*).

PARAMETERS

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

RETURN

`(HOSTID *) hostid`	A pointer to the `HOSTID` struct, filled in for the current host, or NULL on failure.

Note: The memory returned by lc_getid_type() is shared by lc_gethostid(), and both functions free this memory when called. Therefore, do not call lc_getid_type(), and then lc_gethostid(), and expect the first pointer to remain valid.

ERROR RETURNS

LM_CANTFINDETHER	Cannot find ethernet device. If this error is returned, a null `HOSTID` pointer will be returned. (Prior to v5, this was a `NOHOSTID` type, rather than a NULL pointer).

SEE ALSO

- `lmclient.h` for the definition of the `HOSTID` struct

E.1.13 lc_getid_type()

SYNTAX

hostid = lc_getid_type(*job*, *id_type*)

DESCRIPTION

lc_hostid() should normally be used instead.

Returns the HOSTID of the specified type for the local host.

Note: The memory returned by lc_getid_type() is shared by lc_gethostid(), and both functions free this memory when called. Therefore, do not call lc_getid_type(), and then lc_gethostid(), and expect the first pointer to remain valid.

PARAMETERS

(LM_HANDLE *) *job*	From lc_new_job().	
(int) *id_type*	The requested hostid type (see lmclient.h). Types are:	
	HOSTID_LONG	Longword hostid, e.g., SUN
	HOSTID_ETHER	Ethernet address
	HOSTID_USER	User name
	HOSTID_DISPLAY	Display name
	HOSTID_HOSTNAME	Node name
	HOSTID_ID_MODULE	HP300 Id-Module hostid
	HOSTID_STRING	String ID, MAX HOSTID_LEN, used for SCO
	HOSTID_FLEXID1_KEY	FLEXid Dongle
	HOSTID_FLEXID2_KEY	FLEXid Dongle

HOSTID_FLEXID3_KEY	FLEXid Dongle
HOSTID_FLEXID4_KEY	FLEXid Dongle
HOSTID_DISK_SERIAL_NUM	Windows and NT disk serial number
HOSTID_INTERNET	Internet IP address
HOSTID_SERNUM_ID	ID=n hostid. `idptr->id.string` contains this hostid
HOSTID_VENDOR	Start of vendor-defined hostid types
HOSTID_INTEL32	Intel Pentium III+ CPUID (v7.0d+), 32-bit format
HOSTID_INTEL64	Intel Pentium III+ CPUID (v7.0d+), 64-bit format
HOSTID_INTEL96	Intel Pentium III+ CPUID (v7.0d+), 96-bit format

RETURN

`(HOSTID *)` *`hostid`*	A pointer to the `HOSTID` struct, filled in for the current host, or NULL on failure. See `lmclient.h` for the definition of the `HOSTID` struct.

ERROR RETURNS

LM_CANTFINDETHER	Cannot find ethernet device.

Note: lc_getid_type() does NOT process either ANY or DEMO hostid types.

SEE ALSO

- `lmclient.h` for definition of `HOSTID` struct
- Section 5.13.2, “Special FLEXlm Hostids”

E.1.14 lc_isadmin()

SYNTAX

`status` = `lc_isadmin(`*`job`*`, `*`user`*`)`

DESCRIPTION

Verifies that the specified user is a license administrator. A license administrator is a member of the “lmadmin” group. If there is no “lmadmin” group in the `/etc/groups` file, then anyone in group 0 is a license administrator.

PARAMETERS

`(LM_HANDLE *)` *`job`*	From lc_new_job().
`(char *)` *`user`*	Login name of user to test.

RETURN

`(int)` *`status`*	Indication of whether the user is an administrator: 0 if the user is not an administrator, <>0 if an administrator. Always returns 1 on non-UNIX systems.

SEE ALSO

- Chapter 8, “The License Manager Daemon”

E.1.15 lc_lic_where()

SYNTAX

```
path = lc_lic_where(job)
```

DESCRIPTION

Returns path name of FLEX*lm* license file. This function does not support the license file list in the `LM_LICENSE_FILE` environment variable — it only reports on the first file in the list, or, if a feature was already checked out, the file that was used for the checkout. Use lc_get_attr(`LM_A_LF_LIST,...`) for the full list.

PARAMETERS

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

RETURN

`(char *) path`	Path of the license file that FLEX*lm* will use; NULL if no license file set. **Note—** This returned string must not be modified by the caller.

SEE ALSO

- Section 4.13, “LM_A_LF_LIST”

E.1.16 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 8, “The License Manager Daemon”
- Section 9.2.8, “ls_min_lmremove”
- Section 3.21, “lc_heartbeat()”

E.1.17 lc_set_errno()

SYNTAX

```
(void) lc_set_errno(job, error)
```

DESCRIPTION

The FLEX*lm* error is settable via lc_set_errno(). This should not normally be used, because the error should be set by the FLEX*lm* libraries. You may want to set the error to 0 before calling a FLEX*lm* function.

PARAMETERS

`(LM_HANDLE *) job`	From lc_new_job().
`(int) error`	See `lmclient.h` for a list of possible FLEX*lm* error codes.

RETURN

None.

SEE ALSO

- Section 3.10, “lc_err_info()”
- Section 3.28, “lc_perror()”
- Section E.1.10, “lc_get_errno()”
- `lmclient.h`

E.1.18 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 (as of v6).
(int) *print*	Unused (as of v6).

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 8, “The License Manager Daemon”

E.1.19 lc_timer()

SYNTAX

num_failed_attempts = lc_timer(*job*)

DESCRIPTION

This routine is called by vendors that cannot tolerate the use of interval timers by FLEX*lm*. The purpose of lc_timer() is twofold:

- Ensure that the vendor daemon is continually running — otherwise an end user may kill the license server when all licenses are in use, restart the server, and obtain unauthorized licenses.

- Keep license server informed that the client is still using its license — otherwise the license server may timeout the client and drop its license.

If no FLEX*lm* timers are used, then lc_timer() must be called periodically. To avoid FLEX*lm*'s use of timers, call lc_set_attr(LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE)-1), and lc_set_attr(LM_A_RETRY_INTERVAL, (LM_A_VAL_TYPE)-1).

If the default timer is left installed, lc_timer() is called by the FLEX*lm*-installed timer. lc_timer() also performs all the reconnection functions, so it is important to keep calling lc_timer(), even if your reconnection handler is called.

lc_timer() will not do anything if it is called less than 20 seconds since it was last called. This prevents unnecessary networking delays, and no delays will occur because of lc_timer(). Under certain circumstances, lc_timer() must be called often enough to avoid the application losing its license — the vendor daemon will time out clients due to either UDP timeout or the end user TIMEOUT option. The application must ensure that LM_A_UDP_TIMEOUT, LM_A_TCP_TIMEOUT, and ls_minimum_user_timeout are large enough that the application will not inadvertently lose its license. In general, it is a good idea to call lc_timer() once every five minutes, although occasional lapses are relatively harmless.

PARAMETERS

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

RETURN

(int) *num_failed_attempts*	0—success; heartbeat messages exchanged with server. >0—failure; number of failed attempts to contact the server.

SEE ALSO

- Section 9.2.9, “ls_minimum_user_timeout”
- Section 4.3, “LM_A_CHECK_INTERVAL”

E.2 Rarely Used FLEXible API Attributes

E.2.1 LM_A_CONN_TIMEOUT

Type: `(int)`

Default: 10 seconds

If specified as a positive integer, conn_timeout will set the timeout for connection to a vendor daemon, as well as subsequent reads from the daemon. For connect(), this timeout overrides the TCP/IP default of 45 seconds. lc_set_attr() will return LM_BADPARAM, which will also be returned from lc_get_errno(), if this value is <0.

Note: Prior to FLEX*lm* v4.0, this timeout only applied to connect().

E.2.2 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 routine to be compared to the code in the license file with a case-sensitive comparison.

E.2.3 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. GLOBEtrotter Software 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.

E.2.4 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 routines 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.

SEE ALSO

- Section E.2.5, "LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR"
- Chapter 7, "Distributing and Locating the License File"

E.2.5 LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR

Type: `(char *)`

Note: It is recommended that LM_A_LICENSE_DEFAULT be used instead of LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR.

SEE ALSO

- Section 4.15, "LM_A_LICENSE_DEFAULT"
- Section E.2.4, "LM_A_DISABLE_ENV"
- Chapter 7, "Distributing and Locating the License File"

E.2.6 LM_A_LKEY_LONG

Type: `(int)`

Default: False

Obsolete with signatures; only used with license keys. If True, license file license keys will be long—64-bit, and short keys will not be accepted. Also turns on start date in the license key (which can be turned on separately with LM_A_LKEY_START_DATE. If used, ls_a_lkey_long in `lsvendor.c` must also be set to 1. This attribute is automatically turned on by setting LM_VER_BEHAVIOR in `lm_code.h` to LM_BEHAVIOR_V5_1 or less.

E.2.7 LM_A_LKEY_START_DATE

Type: `(int)`

Default: False

Obsolete with signatures; only used with license keys. If true, license keys will contain start dates, and will automatically turn on LM_A_LKEY_LONG, so that license keys will be 20 hex characters long. Useful for generating licenses in pre-v6 format. This attribute is automatically turned on by setting LM_VER_BEHAVIOR in `lm_code.h` to LM_BEHAVIOR_V5_1 or less.

E.2.8 LM_A_MAX_TIMEDIFF

Obsolete. This check is now automatically performed when needed.

E.2.9 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_timer() is called. lc_timer() is called directly or automatically depending on the value of LM_A_CHECK_INTERVAL.

SEE ALSO

- Section 4.3, “LM_A_CHECK_INTERVAL”
- Section 3.21, “lc_heartbeat()”

E.2.10 LM_A_PERIODIC_COUNT

Type: `(int)`

Default: 0 (no PERIODIC_CALL)

This is the count of how many times lc_timer() must be called before the function specified by LM_A_PERIODIC_CALL is called. lc_timer() is called directly or automatically depending on the value of LM_A_CHECK_INTERVAL.

SEE ALSO

- Section 4.3, “LM_A_CHECK_INTERVAL”
- Section 3.21, “lc_heartbeat()”

E.2.11 LM_A_USE_START_DATE

Type: `(short)`

Default: 1, use start date.

This field allows you to use the start date that is built into the license file for each feature. If the current system date is earlier than the start date, then checkouts of the feature will be disabled. If set to a non-zero value, the start date will be used. LM_A_USE_START_DATE can only be turned off with lc_set_attr().

Note: If you use your own authentication routine, you must either disable the use of the start date, or create an license key which contains a valid start date. See the description of the LM_A_USER_CRYPT attribute.

E.2.12 LM_A_USER_CRYPT

Type: Pointer to a function returning `char *`. Return value is the license key.

Default: FLEX*lm* standard authentication routine.

The function pointer `crypt` can be set to point to a vendor-supplied authentication routine to be used in place of the default routine.

The crypt() routine is called as follows:

```
(*crypt)(job, conf, sdate, code);
```

where:

`(LM_HANDLE *) job`	FLEX*lm* job.
`(CONFIG *) conf`	CONFIG structure pointer.
`(char *) sdate`	4-byte encoded start date.

(VENDORCODE *) *code*	Pointer to the first argument to the LM_CODE() macro in lm_code.h, where *code.data*[0] and *code.data*[1] have been XOR'd with *VENDOR_KEY5*, so that the encryption seeds are as specified in lm_code.h.

Because of the complexities of the current CONFIG format, we recommend the following procedure for setting a vendor-defined authentication routine:

1. Set LM_A_USER_CRYPT to your function, e.g., our_crypt().
2. our_crypt() should do the following:

```
our_crypt(job, conf, sdate, key)
{
char *lic_key;

      lc_set_attr(job, LM_A_USER_CRYPT, (LM_A_VAL_TYPE)0);
      lic_key = lc_crypt(job, conf, sdate, key);
      lc_set_attr(job, LM_A_USER_CRYPT,
             (LM_A_VAL_TYPE)our_crypt);
      /* modify the license-key */
      return (modified(lic_key));
}
```

The example above first calls the standard authentication routine (lc_crypt()). Then our_encrypt() modifies the license key and returns the modified value. A simple, but useful way to modify the key is to turn it into a 32-bit integer and XOR it with a fixed number. If the license key contains an embedded start date, then you'll have to first remove the embedded start-date from the license key, perform the modification, and then re-insert the start-date into the license key.

The returned string must be 12 (short with no license key), 16 (long with no embedded start-date) or 20 (long with start-date) characters long.

SEE ALSO

- Section 3.9, "lc_cryptstr()"
- Section E.3.12, "ls_user_crypt"

E.3 Rarely Used Vendor Variables

E.3.1 ls_a_lkey_long

```
(int) ls_a_lkey_long = 0; /* like LM_A_KEY_LONG */
```

Obsolete with signatures. If 1, license keys in the license file are 64-bit. Default is 0, short, 48-bit license keys.

SEE ALSO

- Section E.2.6, “LM_A_LKEY_LONG”

E.3.2 ls_a_lkey_start_date

```
(int) ls_a_lkey_start_date = 0; /* like LM_A_KEY_START_DATE */
```

Obsolete with signatures. If 1, license keys contain a hidden start date, and are four characters longer. Default is 0, no hidden start date.

SEE ALSO

- Section E.2.7, “LM_A_LKEY_START_DATE”

E.3.3 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.

E.3.4 ls_crypt_case_sensitive

```
(int) ls_crypt_case_sensitive = 0; /* Is license key case-sensitive? */
```

If you have written your own authentication routine, and the output code from it is case-sensitive, set ls_crypt_case_sensitive to a non-zero value.

SEE ALSO

- Section E.2.12, “LM_A_USER_CRYPT”

E.3.5 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. GLOBEtrotter Software 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 complete 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.

E.3.6 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.

E.3.7 ls_enforce_startdate

```
(int) ls_enforce_startdate = 1; /* Enforce start date in features */
```

To use the start date present in the FEATURE line, set ls_enforce_startdate to a non-zero value.

E.3.8 ls_read_wait

```
(int) ls_read_wait = 10; /* How long to wait for solicited reads */
```

This variable controls how long the vendor daemon will wait for a connection to be completed with another vendor daemon. The default is 10 seconds. If your daemon supports a large number of features, you may need to increase this value, since the remote daemon’s feature initialization can happen during this timeout interval.

E.3.9 ls_tell_startdate

```
(int) ls_tell_startdate = 1;
     /* Tell the user if it is earlier than start date */
```

To inform the user that a feature's start date is later than the system date, set ls_tell_startdate to a non-zero value. If ls_tell_startdate is 0, then the feature will not be enabled in the daemon, and no warning message will appear in the log file.

E.3.10 ls_use_all_feature_lines

```
(int) ls_use_all_feature_lines = 0;
     /* Use ALL copies of feature lines that are...
```

Note: GLOBEtrotter Software strongly discourages your use of this option, which has been made unnecessary and obsolete by the INCREMENT and UPGRADE features. The ls_use_all_feature_lines option will cause 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.

SEE ALSO

- Section 5.5, "FEATURE or INCREMENT Lines"

E.3.11 ls_use_featset

```
int ls_use_featset = 0;
     /* Use the FEATURESET line from the license file */
```

To require the FEATURESET line in the license file, set ls_use_featset to a non-zero value. FEATURESET is not recommended.

SEE ALSO

- Section E.4.2, "FEATURESET Line"

E.3.12 ls_user_crypt

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

To use your own authentication routine, initialize ls_user_crypt with a pointer to your routine, and make sure an object file with this routine is linked with your vendor daemon. This must be the same as LM_A_USER_CRYPT.

SEE ALSO

- Section 3.9, “lc_cryptstr()”
- Section E.2.12, “LM_A_USER_CRYPT”

E.4 Rarely Used License File Features

E.4.1 License Key Length and Start Date

This section applies only to licenses generated with license keys rather than signatures (SIGN= fields).

The license key is the set of hex digits which appear on every FEATURE/INCREMENT/UPGRADE/PACKAGE line and authenticates the text, making the line secure.

For example:

```
FEATURE f2 demo 1.0 permanent uncounted 6E06CC47D2AB HOSTID=1234
                                        ^^^^^^^^^^^^
                                         license key
```

Previous to v6, license keys were always 20 characters. The license key is now 12 characters instead of 20 by default, but 20-character keys can still be used. 20-character license keys are always accepted while shorter license keys can be disallowed, 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
- Setting `LM_VER_BEHAVIOR` in `machind/lm_code.h` to LM_BEHAVIOR_V5_1 or less.

Shorter license keys are preferred where acceptable, since they’re easier to type in, convert to much shorter decimal format keys, and provide sufficient security for most ISVs.

Shorter 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.
- The 20-character license key included four characters for the license “start date.” This is now optional, and is turned off by default in v6.

We believe that a start date has little practical application for most companies and was rarely used. However, those desiring a start date can now get one in two ways:

- There is now an optional “START=” attribute for FEATURE/INCREMENT/UPGRADE lines. This is the preferred method for a start date.
- You can continue to use a start date in the license key. However, we have imposed the requirement that a start date in the license key *must* be accompanied by a 64-bit license key to remove any ambiguity about what the key contains.

Implementing Long License Keys and Start Dates

Here’s how to turn on long license keys and/or license key 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 hidden start dates with:

  ```
  lc_set_attr(job, LM_A_LKEY_START_DATE, (LM_A_VAL_TYPE) 1);
  ```

- For `lmcrypt` and `makekey`, modify the source in the `machind` directory.
- For the vendor daemon (`lsvendor.c` in machind directory):

  ```
  ls_a_lkey_long = 1;             /* long license keys */
  ls_a_lkey_start_date = 1;       /* hidden start dates */
  ```

Compatibility Issues

- V6 applications (even those accepting short license keys) will accept licenses with long license keys.
- Pre-v6 applications will not accept licenses with short license keys.
- License generators (`lmcrypt`, `makekey`) will issue long license keys when `verfmt` is set to a version less than 6.
- LM_BEHAVIOR_V5_1 (or older) in `lm_code.h` 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.

E.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

- Section E.3.11, "ls_use_featset"
- Chapter 5, "The License File"

Appendix F

Migrating to the Counterfeit Resistant Option

This information in this appendix is for companies who have already shipped products with pre-v7.1 versions of FLEX*lm*. If you are new to FLEX*lm*, you can skip this information. This appendix describes the changes in FLEX*lm* v7.1 from FLEX*lm* v7.0 and recommends migration paths for implementing these changes.

FLEX*lm* v7.1 contains a separately priced option called the Counterfeit Resistant Option (CRO), which can be used to reduce the possibility of counterfeit licenses. Counterfeiting is the most serious threat to FLEX*lm* security, and CRO utilizes industry-standard cryptography recommended by the US government for its own secrets to reduce the possibility of counterfeiting. CRO utilizes Certicom's public-key system based upon elliptical curve cryptography (ECC), which is considered secure, fast, and efficient (see http://www.certicom.com).

CRO is enabled (but not turned on) by default in the evaluation SDK. If you purchase FLEX*lm* with the CRO option, it is enabled by the separate CRO keys you receive from GLOBEtrotter. If you attempt to use CRO without these special CRO keys, you will receive a compile-time error.

F.1 v7.1 Upgrade Overview

When you upgrade to FLEX*lm* v7.1, you will have to decide whether or not to purchase and implement CRO.

F.1.1 If You Do Not Want CRO

If you're upgrading to v7.1, and do not want CRO, in `machind/lm_code.h`, just set `LM_STRENGTH` to LM_STRENGTH_LICENSE_KEY. This is the only change you need make. This setting maintains the use of the FLEX*lm* license key and does not use the new SIGN= attribute.

F.1.2 Migrating to v7.1 with CRO

The recommended implementation of CRO uses public-key encryption technology and utilizes the new SIGN= attribute in addition to the license key on each FEATURE line in the license file, a v7.1 CRO vendor daemon, and a v7.1 CRO application. Planning your strategy for deploying applications with CRO can minimize the cost of supporting your customers during the migration. ISVs who have already shipped FLEX*lm*-based products and who want to implement CRO must make the following decisions, both involving trade-offs:

1. How strong to make the CRO security?

   The tradeoff is security vs the length of the new SIGN= attribute, which can be 58-120 characters long.

2. When to enable CRO in a release of an application?

   The tradeoff is license security vs. end user and software vendor convenience.

F.2 How To Migrate To CRO

First you have to make the two decisions:

- What length SIGN= attribute
- When to enable CRO in your application

F.2.1 Choosing SIGN= Attribute Length

This decision determines what strength of protection you want against counterfeiting. Here are samples of FEATURE lines with each length:

LM_STRENGTH_113BIT (58 chars):

```
FEATURE f1 xyzd 1.0 permanent  uncounted 1234567890AB \
HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
90AB CDEF 1234 5678 90AB CDEF"
```

LM_STRENGTH_163BIT (84 chars):

```
FEATURE f1 xyzd 1.0 permanent  uncounted 1234567890AB \
HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 \
5678 90AB CDEF 1234"
```

LM_STRENGTH_239BIT (120 chars):

```
FEATURE f1 xyzd 1.0 permanent  uncounted 1234567890AB \
HOSTID=ABCDEF1234 SIGN="1234 5678 90AB CDEF 1234 5678 \
90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 \
5678 90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234
5678"
```

Once you've decided on a length, edit `machind/lm_code.h` to set `LM_STRENGTH` to the proper value, make sure that you've added `CRO_KEY`s and four `ENCRYPTION_SEED`s (32-bit numbers that you make up), then build the FLEX*lm* SDK using `make` (UNIX) or `nmake` (Windows).

F.2.2 Disabling/Enabling CRO in Your Application

Support problems occur with CRO-enabled applications where the client:

- Does not yet have a SIGN= license
- Is not using a v7.1 CRO vendor daemon

These problems can be delayed or sometimes avoided completely by delaying enabling CRO in applications. Delaying also delays protection, so if your company requires the highest security available immediately, then do not delay enabling CRO.

If you purchase CRO, the default behavior is that CRO is enabled in your application.

To disable CRO in a particular application:

- Simple/Trivial API

  OR (|) LM_USE_LICENSE_KEY into the policy, for example:

  ```
  CHECKOUT(LM_RESTRICTIVE|LM_USE_LICENSE_KEY,...)
  ```

- FLEXible API

  Call:

  `lc_set_attr(`*`job`*`, LM_A_KEY_LEVEL, (LM_A_VAL_TYPE)0)`

When you are ready to enable CRO in your application, remove these policy modifiers or attributes.

Here are some guidelines for delaying CRO in applications:

- All applications which ship for the first time after v7.1 should have CRO enabled from the start. This gives the maximum protection immediately and no migration problems due to CRO should occur, because the product is new.
- As noted above, make sure you ship immediately a v7.1 CRO vendor daemon and licenses that contain both the SIGN= and license key fields. That way, when you do enable CRO in a particular application, your customers are less likely to encounter a problem.
- If licenses for a particular application all expire over the next year, then you can safely turn on CRO after a year, because users will require new licenses to run anyway.
- If the version in the checkout call will change, then it's safe to enable CRO, because the users will require new licenses with the new version in order to run anyway.
- If licenses don't expire, and versions don't change, then the longer you delay, the smoother the transition will tend to go, because more customers over time will have the requisite SIGN= licenses and v7.1 CRO vendor daemons. Of course, the longer you delay, the longer your applications to subject to possible counterfeit licenses.

Remember:

- Make sure all licenses ship with both license keys and SIGN= attributes.
- Ship v7.1 CRO-enabled vendor daemon and v7.1 `lmgrd` immediately and make them easily available to all your customers.
- Delaying CRO in applications will ease transition, but offers reduced security during this transition period.

F.3 CRO Questions and Answers

WHAT EXACTLY IS CRO?

CRO utilizes industry recognized public-key encryption. There are three different signature lengths offered. The longer the signature, the higher degree of security. The lengths of the SIGN= attribute in the license are:

- 58 chars (113 bits ECC)
- 84 chars (163 bits ECC)
- 120 chars (239 bits ECC)

How Secure Is CRO?

To put this in perspective, the supplier of the elliptical curve cryptography employed by the Counterfeit Resistent Option has an ongoing challenge with a maximum $100,000 reward to crack their cryptography. In 1999, a researcher required 195 volunteers, 40 days of calculation, 16000 MIPS years of computation, and 740 computers located in 22 countries to solve a 97-bit key, which is weaker than any of the CRO options.

Why Wouldn't I Want CRO?

There may be those who feel no need to upgrade their security and see the length of the SIGN= attribute as inconvenient.

Can I Get the New FLEX*lm* v7.1 without CRO?

Yes, CRO is an optional addition to the FLEXlm product.

What Do I Have to Do to Take Advantage of CRO?

There are three components that compose CRO:

- v7.1 license file with the additional field:

  ```
  SIGN="1234 5678 90AB CDEF 1234 5678 90AB CDEF 1234 5678 90AB CDEF"
  ```

- v7.1 CRO-enabled vendor daemon
- v7.1 CRO-enabled application

What Problems am I Likely to Encounter During Migration?

If a v7.1 CRO-enabled application attempts authentication from a pre-v7.1 vendor daemon and/or license file, an error message will be displayed and the application will not run. The error message will inform you that either license file and/or vendor daemon is not v7.1 CRO compliant.

It's important to remember that the v7.1 CRO application itself is the "trigger" that requires both a v7.1 license file and vendor daemon in order to run.

What Happens if a v7.1 Vendor Daemon Encounters a Pre-v7.1 License (without "SIGN=")?

The v7.1 license server will support pre-v7.1 applications, but will fail if CRO applications attempt to use it.

What Happens if a Pre-v7.1 Application Encounters a License File with SIGN= and a v7.1 Vendor Daemon?

The system will perform exactly the way it does now—checkouts will succeed.

The following chart applies to applications implementing CRO:

Application	Vendor Daemon	License File	Result
Pre-v7.1	Pre-v7.1	Pre-v7.1	No change
Pre-v7.1	Pre-v7.1	SIGN=	No change
Pre-v7.1	CRO	Pre-v7.1	No change
Pre-v7.1	CRO	SIGN=	No change
CRO	Pre-v7.1	SIGN=	***FAIL***
CRO	CRO	Pre-v7.1	***FAIL***
CRO	CRO	SIGN=	CRO

The goal is to avoid the two ***FAIL*** possibilities (shown in the chart above) by having v7.1 license files with SIGN= attributes and v7.1 vendor daemons in place BEFORE distribution of CRO applications. For many companies, it will be advantageous to delay enabling CRO in applications, to reduce support calls and customer inconvenience, with the cost of a delay in the actual implementation of the extra security provided by CRO. The most effective method of accomplishing this may depend on how your company typically updates it's customer licenses.

WHAT IS PUBLIC-KEY TECHNOLOGY?

Public-key is based on mathematics, not "hiding" (obfuscation), which is what is used without CRO.

There are two tasks to be accomplished for the SIGN= attribute:

- Generate digital signature (license key)—`lmcrypt`
- Authenticate digital signature—application and vendor daemon

Without public-key, there's one "key," which is the same key in `lmcrypt` that is hidden in the applications and vendor daemon.

With public-key, there are two different keys: private and public. The private key, used only in the license generators (`lmcrypt`), generates the SIGN= attribute. The public key, used by applications and vendor daemon authenticates the SIGN= attribute. It is difficult (but theoretically not impossible) to derive private from the public key; the longer the signature the

harder it is to derive. In practice, the signatures used by FLEX*lm* would require large (sometimes, impossibly large) resources, considerable mathematical skill, and time.

Index

W

X