Reference Manual

VERSION 7.0

GLOBE*trotter* Software, Inc.

November, 1999

Printed in the USA.
November, 1999

Introduction to FLEX*lm* 13
About this Manual 13
How to use this Manual 13
FLEX*lm* Terms and Definitions 13
FLEX*lm* APIs 15

Installing the Distribution Kit—VMS and Source 17
Binary Installation for VMS Systems 17
Installation via UNIX 17
Source Installation 18

Incorporating FLEX*lm* Into Your Application using the FLEXible API 19
FLEX*lm* Example Applications 19
Client Heartbeats and License Server Failures 20
Internationalization 20
Lingering Licenses 20
Multiple Jobs 21
Security and FLEXlm 23
Keeping your software secure 23
lmstrip—Security and Licensing Libraries 23
lmstrip Adds Security to Binaries (Unix only) 24
Licensing libraries with FLEX*lm* 24
Linking with a library that already uses FLEX*lm* 25

FLEXible API 27
FLEXible API Library Routines 27
Building Your Application 28
l_new_hostid 29
lc_auth_data 29
lc_check_key 30
lc_checkin 31
lc_checkout 31
lc_chk_conf 37
lc_convert 37
lc_cryptstr 39

lc_err_info 41
lc_errstring 42
lc_errtext 43
lc_expire_days 43
lc_feat_list 44
lc_first_job, lc_next_job 45
lc_free_hostid 45
lc_free_job 46
lc_free_mem 46
lc_get_attr 47
lc_get_config 48
lc_heartbeat 49
lc_hostid 50
lc_idle 51
lc_init — (license generator only) 52
lc_log 53
lc_new_job 53
lc_next_conf 55
lc_perror 55
lc_set_attr 56
lc_status 57
lc_userlist 58
lc_vsend 59
Rarely used functions 60
lc_baddate 60
lc_ck_feats 60
lc_copy_hostid 61
lc_crypt 61
lc_disconn 63
lc_display, lc_hostname, lc_username 63
lc_feat_set 64
lc_get_errno 64
lc_get_feats 65
lc_gethostid 65
lc_getid_type 66
lc_hosttype 67

lc_isadmin 68
lc_lic_where 69
lc_master_list 69
lc_remove 70
lc_set_errno 71
lc_shutdown 71
lc_startup 72
lc_timer 72

Controlling Licensing Behavior Of Your Application With lc_set_attr 75
LM_A_BEHAVIOR_VER 76
LM_A_CHECK_BADDATE 76
LM_A_CHECK_INTERVAL 77
LM_A_CHECKOUT_DATA 77
LM_A_CHECKOUTFILTER 78
LM_A_CKOUT_INSTALL_LIC 78
LM_A_DISPLAY_OVERRIDE 79
LM_A_FLEXLOCK 80
LM_A_FLEXLOCK_INSTALL_ID 80
LM_A_HOST_OVERRIDE 80
LM_A_LCM 81
LM_A_LCM_URL 81
LM_A_LF_LIST 81
LM_A_LICENSE_CASE_SENSITIVE 82
LM_A_LICENSE_DEFAULT 82
LM_A_LICENSE_FMT_VER 82
LM_A_LINGER 82
LM_A_LKEY_LONG 83
LM_A_LKEY_START_DATE 83
LM_A_LONG_ERRMSG 83
LM_A_PERROR_MSGBOX (Windows only) 84
LM_A_PROMPT_FOR_FILE (Windows only) 84
LM_A_RETRY_CHECKOUT 84
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL 85
LM_A_SETITIMER, LM_A_SIGNAL (Unix only) 85
LM_A_TCP_TIMEOUT 86

LM_A_UDP_TIMEOUT 86
LM_A_USER_EXITCALL 86
LM_A_USER_OVERRIDE 87
LM_A_USER_RECONNECT 87
LM_A_USER_RECONNECT_DONE 88
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO 88
Detecting OVERDRAFT for SUITEs 90
LM_A_VENDOR_ID_DECLARE 90
LM_A_VENDOR_GETHOSTID 91
LM_A_VERSION and LM_A_REVISION 91
LM_A_WINDOWS_MODULE_HANDLE 91

License Models and the License File 93
Demo Licensing 93
Limited Time, Uncounted Demos 93
Limited Functionality Demos 94
Lenient Licensing: REPORTLOG and OVERDRAFT 94
REPORTLOG 94
OVERDRAFT detection 95
Format of the License File 96
SERVER Line 97
VENDOR Line 98
USE_SERVER line 100
FEATURE or INCREMENT Line 100
UPGRADE Line 107
PACKAGE Line 107
FEATURESET Line 110
Comments 110
Continued Lines 110
Example License File 110
Decimal Format Licenses 111
Decimal Format Limitations 111
Example decimal licenses: 112
Format of a Decimal License 112
Hints on Using the Decimal Format 112
Locating the License File 113

License specification 114
Using License File List for Convenience and Redundancy 115
Hostids for FLEX*lm* Supported Machines 116
Special Hostids 118
Vendor-defined hostid 120
Intel Pentium III+ Hostid (HOSTID_INTEL) 121
Enabling the CPU Hostid 121
Hostid length 121
Converting from 96-bit to 32-bit 122
lmtools and lmhostid 122
Security issues 122
Types of License Files 122
License Key Length and Start Date 123
Changing license key behavior 123
Compatibility Issues 124
License in a buffer 124
E-mailing licenses 125
Newline additions 125
Adding ".txt" to the license file name 125
Other transformations 126
Order of lines in license file 126
sort=nnn 127

lmgrd 129
lmgrd - The License Daemon 129
No lmgrd on VMS or Netware 130
Starting vendor daemons on VMS 130
Starting Vendor Daemons on Netware 131
Starting lmgrd on Windows NT 131
Pre FLEX*lm* v2.0 Startup Arguments 132
Privileged License Administration Commands 132

Configuring Your Vendor Daemon 133
Building Your Vendor Daemon—VMS Systems 133
Building Your Vendor Daemon—Windows NT Systems 133
lsvendor.c variables 134

default_license_file (VMS only) 134
ls_a_behavior_ver 134
ls_a_check_baddate 134
ls_a_license_case_sensitive 134
ls_a_lkey_long 134
ls_a_lkey_start_date 134
ls_compare_vendor_on_increment 135
ls_compare_vendor_on_upgrade 135
ls_conn_timeout 135
ls_crypt_case_sensitive 135
ls_daemon_periodic 136
ls_do_checkroot (Unix only) 136
ls_dump_send_data 136
ls_enforce_startdate 136
ls_infilter 137
ls_incallback 137
ls_min_lmremove 137
ls_minimum_user_timeout 137
ls_read_wait 137
ls_outfilter 138
ls_show_vendor_def 138
ls_tell_startdate 138
ls_use_featset 138
ls_use_all_feature_lines 138
ls_user_init1 139
ls_user_init2 139
ls_vendor_msg 139
ls_user_crypt 139
ls_user_lockfile 140
Vendorkeys and Vendor Encryption Seeds 140
Vendor Daemon Support Routines 141

Redundant License Servers 143
Three-server redundancy 143
Selecting Server Nodes 143
Generating a license file for redundant servers. 143

Sample three-server license file 144
Redundancy via License File List in $LM_LICENSE_FILE 144
Comparing three-server to License File List 145

Debugging Hints 147
Debugging Your Application Code 147
Solving Problems In The Field 148
Multiple Vendors Using FLEX*lm* At A Single End-User Site 149
FLEX*lm* Version Compatibility 150

Communications Transport 151
How to Select UDP Connections 151
UDP Behavioral Differences 152

UNIX and VMS Platform-Specific Notes 155
Data General 155
Hewlett Packard 155
IBM 155
NCR 156
SGI 156
SCO 157
SVR4 Systems 157
VMS 157
Communications transport 157
Special AST Considerations 158
DECnet Logical Links 158
VMS Ethernet Device Support 159
lmswitch 159

Windows 95/98, and Windows NT 161
Supported Compilers 161
lc_new_job() and lc_free_job() Must Be Matched 161
FLEX*lm* Callback Routines 162
FLEX*lm* exit() Callback 162
Default License File 162
Time Zone Setting 162

Node Lock and Hostid for Standalone PCs 163
System Requirements for obtaining Ethernet Address 164
Windows 95 164
Quick-Win/ 32 Bit Console Applications 164
Networking Requirements 164
Hardware Hostids (Dongles) 164
General Information 164
FLEXID 164
Environment Variables (32-bit Platforms) 165
Special Syntax for Windows Version 165
Minimum Files Required for Customer Installation 165
Client 165
Server 165
Build Notes 165
Linking to your Program 166
If your application is a DLL 166
Special Operating Conditions 166
Timers 166
Ethernet Address for WIN32 Platform 167
Operation on NEC NT machines 168
Networking 168
FLEX*lm* TCP/IP Network Problem 168
FLEX*lm* Utilities 168
Servers and Services 169
Control Panel and Multiple Lmgrd's 169
WINDOWS 95 Multiple Servers 170
Default Operation when Server connection is lost 170
Version Information 170
Using languages other than "C" 171
Server Environment Variables 171

Netware NLM 173
Introduction and Requirements 173
Installation 173
Creating a License File For Netware 173
Starting the Vendor Daemon on a Netware Server 174

Preparing the Client System to Use IPX/SPX for FLEX*lm* 174
Accessing License Servers via TCP/IP and SPX/IPX Concurrently 175
Notes 176

Industry-Standard Licensing APIs 177
The FLEX*lm* FLEXible API 177
The FLEX*lm* Trivial and Simple APIs 177
Software License Working Group 177
LSAPI v1.1 179
LSAPI General Calls 180

The Debug Log File 183
Informational Messages 183
Configuration Problems 185
Daemon Software Errors 186

FLEX*lm* Status Return Values 189

FLEXlm Limits 201
License File Limits 201
End-User Options File Limits 202
lc_set_attr() limits 202
Other API limits 202
Vendor Daemon Limits 202
lmgrd 203
Sub-net, Domains, Wide-Area Networks 203
LM_LICENSE_FILE 203

Additional lc_set_attr() Attributes 205
LM_A_ALLOW_SET_TRANSPORT 205
LM_A_ALT_ENCRYPTION 205
LM_A_COMM_TRANSPORT 207
LM_A_CONN_TIMEOUT 207
LM_A_CRYPT_CASE_SENSITIVE 207
LM_A_DIAGS_ENABLED 207
LM_A_DISABLE_ENV 208

LM_A_EF_1, LM_A_EF_2, LM_A_EF_3, LM_A_EF_4, LM_A_EF_5 208
LM_A_ETHERNET_BOARDS 209
LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR 210
LM_A_MAX_TIMEDIFF 210
LM_A_NO_TRAFFIC_ENCRYPT 210
LM_A_PERIODIC_CALL 210
LM_A_PERIODIC_COUNT 211
LM_A_USE_START_DATE 211
LM_A_USER_CRYPT 211

Chapter 1

Introduction to FLEX*lm*

1.1 About this Manual

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

The FLEX*lm* Programmers Guide provides an introduction to FLEX*lm*, descriptions of the Trivial and Simple APIs, descriptions of the license administration tools which are bundled with FLEX*lm*, and guidelines for integration of FLEX*lm* into your application.

The FLEX*lm* End-User Manual contains information relevant to users of products that utilize FLEX*lm* as their licensing system. This manual 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.

1.3 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 the way that an application developer uses it. 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 usage of software 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 daemon that dispenses licenses for the requested features. This daemon is built by an application’s vendor (from libraries supplied by GLOBE*trotter* Software) and contains the vendor’s unique encryption seeds.
lmgrd	The daemon process, or license daemon, that sends client processes to the correct vendor daemon on the correct machine. The same license daemon process can be used by all applications from all vendors, as this daemon neither authenticates nor dispenses licenses.
server node	A computer system that is running the license server software. The server node will contain all the site-specific information regarding the usage of all the features. Multiple server nodes used for redundancy can logically be considered the *server node*.
license file	A site-specific file that contains descriptions of the server node(s) that can run the license daemons, the various vendor daemons, and the licenses (features) for all supported products.
license file list	A list of license files can be accepted in most places where a license file is appropriate. The list is separated with a colon ‘:’ on Unix, a semi-colon ‘;’ on Windows and a space on VMS. When a directory is specified, all files matching **.lic* in that directory are automatically used, as if specified as a list.
license key	A 12- to 20-character hexadecimal number which “authenticates” the readable license file text, ensuring that the license text has not been modified.
license server	The lmgrd and vendor daemon processes. License server refers to the processes, not the computer.

1.4 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 4 APIs available to the developer:

- Trivial API
- Simple API
- FLEX*ible* API
- Java API

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.

You should use the Trivial or Simple APIs when you can. The Simple and FLEXible APIs should only be used when the Trivial API is not feasible. The Simple and Trivial APIs (as well as the JAVA API) are documented in the FLEX*lm* Programmers Guide, while the FLEXible API is documented in detail in this manual. Following are guidelines for which API to use:

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

- A single process needs to separately license sub-functionality—that is, when 2 or more feature names may be checked out.
- The checkout call needs to be able to checkout more than 1 license of a feature.

Most commonly, the FLEXible API is required for

- Asynchronous queuing, especially in X or Windows-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.

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 policy upon server failure must be programmed into the application.

Chapter 2

Installing the Distribution Kit—VMS and Source

Note: For Unix and Windows installation, please refer to the FLEX*lm* Programmers Guide where this topic is covered in full.

FLEX*lm* is available via ftp from *ftp://ftp.globes.com/flexlm/current.*

2.1 Binary Installation for VMS Systems

A FLEX*lm* binary distribution kit consists of a single directory which contains all the libraries, include files, and programs. To install FLEX*lm*, you will create a directory for the kit, read the distribution media, then run the FLEX*lm* installation program.

2.1.1 Installation via UNIX

FLEX*lm*VMS kits are downloaded to UNIX systems from the ftp site. Use install_flexlm.ftp to unpack the files. After running install_flexlm.ftp, copy the resultant ascii file, along with the file "mftu.com" to your VMS system, using your normal networking commands. execute the following commands (substitute *alpha_v1* for *vax_v5* if you are extracting an alpha/openvms kit):

```
$ create/dir [.mftu]
$ copy mftu.com [.mftu]
$ set def [.mftu]
$ @mftu
$ @mftumake
$! In the next command, disk:[dir.mftu] refers to your default
$ mftu :== $disk:[dir.mftu]mftu
$ set def [-]
$ rename vax_v5 vax_v5.mftu
$ mftu
MFTU> decode vax_v5.mftu
$ backup vax_v5.bck/save [...]
```

2.2 Source Installation

A FLEX*lm* source distribution kit consists of a single directory which contains subdirectories for the client library, include files, and both license and vendor daemons. The BINARY_KIT script will produce a binary kit identical to the binary distribution in the subdirectory *arch_os*. *arch* is the machine architecture and *os* is the operating system version, as specified in the BINARY_KIT command. For example, *arch* could be one of sun4, hp700, vax, apollo, etc., and *os* could be u4 (for SunOS4). To install the source kit, you will create a directory to hold it, read the distribution media, then perform the build, which will result in a binary kit for the target platform.

Source is available upon request by an email request to support@globes.com. After installing the source files, create the binary kit

```
% cd flexlm
% v6.1/BINARY_KIT v6.1 arch_os
```

It is important to use the correct *arch_os* value when running BINARY_KIT, since the BINARY_KIT script may change make parameters depending on the machine architecture.

The gplatargs script in the *utils* directory is used to determine architecture specific flags. gplatform in the *utils* directory is also used to determine *arch_os*. Both gplatform and gplatargs may require modification for new platforms.

Chapter 3

Incorporating FLEX*lm* Into Your Application using the FLEXible API

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.

3.1 FLEX*lm* Example Applications

The FLEX*lm* distribution kit contains an example FLEXible API client application program in the machind directory called *lmflex.c. lmclient.c* is a small stand-alone Trivial API licensed program and is a good place to start to learn how to integrate FLEX*lm* with your application. A Simple API example program is also available in *lmsimple.c*. The source to these programs is in the *machind* directory.

Windows and Windows NT systems contain a second example application which uses Microsoft Visual C++ to build a slightly more complicated 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 makekey and lmcrypt programs is in the machind directory.

The lmcrypt and makekey programs generate the same license keys 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* kits also contain an *examples* directory at the top-level of the kit hierarchy. The *examples* directory contains example programs, which have been put on the kit in order to illustrate how to perform various operations with FLEX*lm*. These programs are **not supported**, and GLOBE*trotter* Software may not include them in future FLEX*lm* releases.

3.2 Client Heartbeats and License Server Failures

Your application will need to communicate regularly with the server via “heartbeats”, to ensure the server is still running. How the heartbeats occur and what action takes place when the 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 FLEX*lm* Programmers Guide. The FLEXible API heartbeat is addressed in the following sections:

- Section 4.22, “lc_heartbeat,” on page 49
- Section 5.3, “LM_A_CHECK_INTERVAL,” on page 77
- Section 5.24, “LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL,” on page 85
- Section 5.28, “LM_A_USER_EXITCALL,” on page 86
- Section 5.30, “LM_A_USER_RECONNECT,” on page 87
- Section 5.31, “LM_A_USER_RECONNECT_DONE,” on page 88

3.3 Internationalization

A message file for localization of FLEX*lm* messages has been provided for Solaris 2.*x*. The message file is *FLEXlm.po*, and it is contained in the machind directory. The text domain is “FLEX*lm*”. All messages from lmgrd, the vendor daemons, and lmutil are contained in this message file. Since the client library does not output any messages, and the lmcrypt, makekey and lmfeats programs are example programs for the ISV’s use only (which are shipped in source form, anyway), these have not been internationalized. All FLEX*lm* client library messages are contained in the file *machind/lmerrors.h*. In order to build a localized version of the daemons and utilities, do the following:

1. Edit *FLEXlm.po* to contain the text translations.
2. Run msgfmt on *FLEXlm.po* (produces *FLEXlm.mo*).
3. Put *FLEXlm.mo* into the appropriate message directory. For example, if you were doing a french translation:

   ```
   % cp FLEXlm.mo /usr/lib/locale/fr/LC_MESSAGES
   ```

4. Set your language environment:

   ```
   % setenv LANG fr
   ```

5. Run lmgrd or the utility program. You should see the translated messages.

3.4 Lingering Licenses

A lingering license allows you to specify how long a license will remained 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_new_joblc_set_attr(job, LM_A_LINGER, x)`

where:	**is the:**
(long) *x*	number of seconds to make the license linger.

In addition, the end-user can specify a longer linger interval in his daemon 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 only useful if DUP_GROUP is also set.

SEE ALSO

- Section 5.16, “LM_A_LICENSE_FMT_VER,” on page 82
- Section 8.4, “Vendor Daemon Support Routines,” on page 141
- Section 6.3.4, “FEATURE or INCREMENT Line,” on page 100

3.5 Multiple Jobs

lc_new_job(job, 0, y, &job_handle) 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, semicolon separated on Windows or Windows NT or a space separated list on VMS platforms) with more than one 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 servers to check out the necessary licenses. This can only be done with multiple jobs, since 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 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, 0, &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, 0, &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 4.18, “lc_free_job,” on page 46
- Section 4.20, “lc_get_attr,” on page 47
- Section 4.27, “lc_new_job,” on page 53
- Section 4.16, “lc_first_job, lc_next_job,” on page 45
- Section 4.30, “lc_set_attr,” on page 56

3.6 Security and FLEX*lm*

3.6.1 Keeping your software secure

No software is completely secure. FLEX*lm* is no exception. While GLOBE*trotter* Software has made every effort to ensure the integrity of FLEX*lm*, all points of attack can never be anticipated. Globetrotter Software also 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.)

3.6.2 lmstrip—Security and Licensing Libraries

lmstrip and the source, lmstrip.c, is included in the FLEX*lm* kits. lmstrip has 3 related, but different, uses:

- Adds security to applications (Unix only)
- Additional security for licensing libraries.
- Allows 2 companies to use 2 different FLEX*lm* versions in the same binary.

The usage for lmstrip is:

lmstrip *file* [-l] [-e | -n] [-r] [-m] [-mapfile *filename*] [*strings*...]

-l	List internal and external names to be stripped
-e	Don’t strip external names
-n	Don’t strip internal and external names
-r	Replaces 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.

-mapfile *filename*	Override default mapfile name to *filename*
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 2 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, truncated to no more than 6 characters long per name.

3.6.3 lmstrip Adds Security to Binaries (Unix only)

When run on a dynamically-linked binary, lmstrip adds more security than the normal Unix strip command, since 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 is about the same level of security anyway, since the most important functions, from a security viewpoint, are the *l_xxx* calls.

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

3.6.4 Licensing libraries with FLEX*lm*

UNIX

We recommend the following steps:

- ld -r file.o liblmgr.a -o ofile.o
  ofile.o then includes all necessary FLEX*lm* calls.
- lmstrip -r ofile.o
  This randomizes the names of the FLEX*lm* function calls.

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

WINDOWS

```
Usage:
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 2 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 since the function names are not meaningful nor deducible.

3.6.5 Linking with a library that already uses FLEX*lm*

UNIX

Follow the steps in Section 3.6.4, “Licensing libraries with FLEXlm” (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.

WINDOWS

Follow the steps in Section 3.6.4, “Licensing libraries with FLEXlm” (Windows, above), using lmstrip -r -m. *mylmgr*.lib won’t conflict with other companies’ use of FLEXlm, since the symbol names are altered.

Chapter 4

FLEXible API

This is the most powerful API available for license management. As such, it contains quite a bit of complexity. It is essentially unchanged from FLEX*lm* v4.0. Where possible, new applications should use the Simple or Trivial APIs; however, there is no reason to change applications which use the FLEXible API.

Some functionality is only available in this API. Also, the C interface to license generation is *lc_cryptstr()*, which is only available in the FLEXible API.

4.1 FLEXible API Library Routines

The application program is linked with the FLEX*lm* client library. The routines to manage licenses are all contained in the FLEX*lm* client library *liblmgr.a* (*lmgr.lib* for Windows NT). The following are the most commonly used routines, however the only required routines are *lc_new_job* and *lc_checkout* and the *LM_CODE()* macro:

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 system hostid.
lc_idle	Supports Administrator defined TIMEOUT option via options.dat files.
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.

Following is a list of rarely used routines, which mostly exist for historical reasons. It is rare that an application will require these functions, and care should be used when calling them: *lc_baddate, lc_ck_feats, lc_crypt, lc_disconn, lc_display, lc_feat_set, lc_get_errno, lc_get_feats, lc_gethostid, lc_getid_type, lc_hostname, lc_hosttype, lc_isadmin, lc_lic_where, lc_master_list, lc_remove, lc_set_errno, lc_shutdown, lc_startup, lc_timer, lc_username*. These are documented separately at the end of these chapter.

The include file *lmclient.h* contains all the symbolic definitions required for most calls. *lm_code.h* contains the vendor's encryption seeds and FLEX*lm* vendor key values. *lm_attr.h* contains the definitions used in the *lc_set_attr()* and *lc_get_attr()* calls.

4.2 Building Your Application

If you use any of the FLEX*lm* symbolic definitions, macros, or data structures, you must include *lmclient.h* and *lm_code.h* in your C module. *lc_set_attr()* calls require you to include *lm_attr.h*. Your encryption seeds, FLEX*lm* vendor keys and vendor name are in *lm_code.h*.

In order to build your application:

1. Insert the calls that you require into your code.

Link your code with the FLEX*lm* library *liblmgr.a* (or *lmgr327b.lib* on Windows NT systems, or *lmgr.olb* on VMS). If you have loaded the distribution kit into */usr/license/* then, on Unix, use a command of the following form:

% cc -o program program.o $(OBJS) -L/usr/license/lmgr/arch_os -llmgr

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

On Unix, it is strongly recommended that your application be linked dynamically. 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.

4.3 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 *) config	A HOSTID struct, or null.

ERROR RETURNS

LM_CANTMALLOC	malloc call failed.

SEE ALSO

- Section 5.33, "LM_A_VENDOR_ID_DECLARE," on page 90
- Section 6.8, "Vendor-defined hostid," on page 120

4.4 lc_auth_data

SYNTAX

```
config = 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 *) *config*	The config structure, or NULL if error. The config structure 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, then 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 (struct CONFIG *).

SEE ALSO

- Section 4.21, "lc_get_config," on page 48
- *lmclient.h* for the CONFIG struct definition.

4.5 lc_check_key

SYNTAX

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

DESCRIPTION

lc_check_key determines if the license key in the CONFIG is valid. To verify a license file upon installation, you could use code similar to the following example:

```
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()*.
(VENDORCODE *) *code*	From LM_CODE macro.

RETURN

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

ERROR RETURNS

LM_BADCODE	license-key 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 4.28, “lc_next_conf,” on page 55
- Section 4.9, “lc_convert,” on page 37
- Section 4.15, “lc_feat_list,” on page 44

4.6 lc_checkin

SYNTAX

```
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*. However if UDP is used, the licenses will not be returned to the pool for *LM_A_UDP_TIMEOUT* seconds.

For reporting purposes in the REPORTLOG file, it is preferable to checkin a license with lc_checkin() rather than simply exiting, since these are recorded differently in the REPORTLOG 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.

SEE ALSO

- Section 5.27, “LM_A_UDP_TIMEOUT,” on page 86
- Section Chapter 11, “Communications Transport,” on page 151

4.7 lc_checkout

SYNTAX

```
LM_CODE(key, ...)
status = lc_checkout(job, feature, version, num_licenses, flag, key,
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. [1] 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.

Notes:

- It is strongly recommended that the application first indicate the expected license file location, with
  lc_set_attr(*job*, LM_A_LICENSE_DEFAULT, *licpath*/license.dat).
  The *licpath* 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 1 license being checked out, not 2. A second call to request 2 licenses would result in a total of 2 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.

1. For TCP clients, the resulting checkin is immediate, but for UDP clients, if the client exits and is not able to call *lc_checkin*, the license server has to timeout the client, and this takes *LM_A_UDP_TIMEOUT* seconds.

(char *) *version*	The version of the feature desired in floating point format, maximum of 10 characters (e.g.: "12345.123" or "123.456789". This value must be <= the version number in the license file for the checkout to succeed.

Notes:
- letters are not allowed in versions; "v1.0" is illegal.
- Prior to FLEX*lm* v4.0, the version argument to *lc_checkout()* was of type double.

(int) *num_licenses*	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()*.
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.
(VENDORCODE *) *key*	The first argument to *LM_CODE*, which includes the vendor's encryption seed for this feature.
(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:

dup_group 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.)
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.
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. (The duplicate grouping value is reset whenever all licenses are checked back in.) 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.

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 ~/.FLEXLMRC

Environment variables can be taken either from the environment, or from the registry (on Windows) or ~/.flexlmrc (Unix). After a successful checkout, the $VENDOR_LICENSE_FILE variable is set for the location in the registry (Windows) or ~/.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	License key 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	"key" structure is incorrect type, or feature == NULL, or num_licenses == 0.
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	Can not 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

- Section 5.4, “LM_A_CHECKOUT_DATA,” on page 77
- Section 5.15, “LM_A_LICENSE_DEFAULT,” on page 82
- Section 5.10, “LM_A_HOST_OVERRIDE,” on page 80
- Section 5.7, “LM_A_DISPLAY_OVERRIDE,” on page 79
- Section 5.29, “LM_A_USER_OVERRIDE,” on page 87
- Section 4.6, “lc_checkin,” on page 31
- Section 6.3.4, “FEATURE or INCREMENT Line,” on page 100
- Section 4.31, “lc_status,” on page 57
- Section 3.5, “Multiple Jobs,” on page 21
- Section 5.6, “LM_A_CKOUT_INSTALL_LIC,” on page 78

4.8 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.

Notes:

- Normally *lc_ck_conf()* should only be used by a license generation program that calls *lc_crypt()*, such as lmcrypt, since 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.

4.9 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 "SERVER this_host" to the real hostname, on either decimal or readable licenses. It only does this 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.

The output, if readable is requested, 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.
(ptr to char *) *return_str*	*str* converted to desired format. Should be freed by caller; use *lc_free_mem()* on Windows.
(ptr 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.

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.
-1	Explanatory string is provided in *errors*.

SEE ALSO

- examples/advanced/exinstal.c, for an example program.
- Section 4.28, "lc_next_conf," on page 55
- Section 4.9, "lc_convert," on page 37
- Section 4.15, "lc_feat_list," on page 44
- Section 4.10, "lc_cryptstr," on page 39, because lc_convert() has a similar interface as lc_cryptstr().
- Section 5.16, "LM_A_LICENSE_FMT_VER," on page 82

4.10 lc_cryptstr

SYNTAX

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

DESCRIPTION

Generates license file as a string with license keys 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 license key must be replaced with a single ‘0’ (zero).

If *flag* has *LM_CRYPT_ONLY* set, then the function returns the license key 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 license keys. If *flag* has *LM_CRYPT_FORCE* set, then every line will have the license key re-computed, even if the key is not set to ‘0’ If *LM_CRYPT_FORCE* is set, and if a line already has a license key, the start date will be taken from the current key.

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”. If you want to specify a start date other than today, then in place of a ‘0’ in the license key, use the following syntax:

```
start:dd-mmm-yyy.
```

EXAMPLE:

```
start:1-jan-1995
```

The output, if readable is requested, 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*	Set *str* to a complete valid license file, where the license keys are replaced with ‘0’.
(ptr 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.
(VENDORCODE *) *code*	From *LM_CODE* macro, with *code.data[0]* and *code.data[1]* XOR’d with VENDOR_KEY5.

(int) *flag*	mask which can be binary OR'd ('|') with the following flags: *LM_CRYPT_ONLY*—If true, only return license key for first FEATURE in *str.* *LM_CRYPT_FORCE*—if set, recompute the license key for *every* line, even if the license key 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 filename.
(ptr 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, this *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

Since different errors can occur on every line of the input *str*, *lc_cryptstr* must be able to report all these errors independently, and does so via the *errors* parameter. The *errors* parameter is used for both errors and warnings. If there are only warnings, the return value from *lc_cryptstr* is success (0), but *errors* is set to a warning message. If it's an error, *lc_cryptstr* returns non-zero, and no license keys are generated in *return_str.* Here is an example of error reporting:

Input:

```
FEATURE f1 demo 1.a50 01-jan-1999 0 0 HOSTID=08002b32b161
```

Error reported:

```
stdin:line 1:Bad version number - must be floating point number, with no
letters
```

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

Before calling *lc_cryptstr()*, add the following code:

```
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
       VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
VENDORCODE vc;
[...]
(void) memcpy((char *)&vc, (char *)&code, sizeof(vc));
vc.data[0] ^= VENDOR_KEY5;
vc.data[1] ^= VENDOR_KEY5;
lc_cryptstr(job, ., &vc, ..); /* for example */
```

SEE ALSO

Section 4.5, “lc_check_key,” on page 30

Section 4.9, “lc_convert,” on page 37

Section 4.19, “lc_free_mem,” on page 46

Section 5.16, “LM_A_LICENSE_FMT_VER,” on page 82

machind/lmcrypt.c

machind/makekey.c

4.11 lc_err_info

SYNTAX

```
err_info = lc_err_infojob)
```

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 translation 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, and thereby provide improved support.
(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 list 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* are in machind/lm_errs.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 4.29, “lc_perror,” on page 55
- Section 4.13, “lc_errtext,” on page 43

4.12 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*.

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.

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 4.29, “lc_perror,” on page 55
- Section 4.13, “lc_errtext,” on page 43

4.13 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 FLEXlm. 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 GLOBE*trotter* 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 4.29, “lc_perror,” on page 55
- Section 4.11, “lc_err_info,” on page 41

4.14 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	*conf* has already expired.

4.15 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, daemon)
```

where:	**is the:**
(char *) *feature*	feature name.
(char *) daemon	daemon for "feature".

ERROR RETURNS

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

4.16 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 4.18, “lc_free_job,” on page 46
- Section 4.27, “lc_new_job,” on page 53
- Section 3.5, “Multiple Jobs,” on page 21

4.17 lc_free_hostid

SYNTAX

```
lc_free_hostid(job_handle, hostid_ptr)
```

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()*, since they free their own memory.

PARAMETERS

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

(HOSTID *) *hostid_ptr* From *l_new_hostid()*.

RETURN

None

ERROR RETURNS

LM_BADPARAM no such job.

SEE ALSO

- Section 4.3, “l_new_hostid,” on page 29

4.18 lc_free_job

SYNTAX

```
lc_free_job(job_handle)
```

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 or VMS, this call is only needed 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 4.25, “lc_init — (license generator only),” on page 52
- Section 4.27, “lc_new_job,” on page 53
- Section 4.30, “lc_set_attr,” on page 56
- Section 3.5, “Multiple Jobs,” on page 21

4.19 lc_free_mem

SYNTAX

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

DESCRIPTION

lc_free_mem() frees memory allocated by another FLEX*lm* function. *lc_free_mem()* is portable, and 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 4.9, “lc_convert,” on page 37
- Section 4.10, “lc_cryptstr,” on page 39

4.20 lc_get_attr

SYNTAX

```
#include "lm_attr.h"
status = lc_get_attr(job, key, 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.

Notes:

- *value* must be a pointer to the correct attribute type, and should be cast to a (short *).
- 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) *key*	Which attribute to get.

RETURN

(short *) *value*	Value of the attribute (cast to appropriate type).
(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 4.30, "lc_set_attr," on page 56
- Section Chapter 5, "Controlling Licensing Behavior Of Your Application With lc_set_attr," on page 75

4.21 lc_get_config

SYNTAX

```
config = 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 CONF struct, and *lc_next_config()* retrieves the next (*lc_next_config()* 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 license-key, and *lc_get_config()* 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.

RETURN

(struct config *) *config*	The config structure. If no feature found, then NULL. The config structure 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 4.4, "lc_auth_data," on page 29

- Section 4.28, “lc_next_conf,” on page 55

4.22 lc_heartbeat

Syntax

```
rc = lc_heartbeat(job, ret_num_reconnects, 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 *ret_num_reconnects* and *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 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 *) *ret_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 *ret_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) *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 *minutes*.

Return

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

How lc_heartbeat() Works

lc_heartbeat() sends a heartbeat 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 and VMS (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.

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: the application is communicating via UDP 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_UDP_TIMEOUT* and *LM_A_TCP_TIMEOUT* are large enough to accommodate your usage of *lc_heartbeat()*. Similarly, make sure *ls_minimum_user_timeout* in *lsvendor.c* is large enough so that users will not timeout applications that are in use.

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

See also

- Section 4.22, "lc_heartbeat," on page 49

4.23 lc_hostid

Syntax

```
stat = 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, since 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 are specified and described in lmclient.h.

RETURN

(int) *stat*	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

4.24 lc_idle

SYNTAX

```
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 of UDP clients.

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 FLEX*lm* 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(1);    /* Process is idle now */
... get input from user...
lc_idle(0);/* Process is no longer idle */
```

PARAMETERS

(int) *flag*	0 if process is not idle, non-zero if process is idle.

RETURN

None.

Caution: There is no detection of license loss (due to daemon shutdown, etc.) while the application is "idle". UDP clients can inadvertently lose their license if the application is idle longer than *LM_A_UDP_TIMEOUT*. When the application is active again, the license, if it is still available, will be re-checked out.

SEE ALSO

- Section 4.22, "lc_heartbeat," on page 49
- Section 5.27, "LM_A_UDP_TIMEOUT," on page 86
- Section 8.3.18, "ls_minimum_user_timeout," on page 137

4.25 lc_init — (license generator only)

SYNTAX

```
#include "lm_code.h"
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
        VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
LM_HANDLE *job = (LM_HANDLE *)NULL;
status = lc_init(prevjob, VENDOR_NAME, &code, &job)
```

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, as it offers enhanced security.

Please refer to *lc_new_job()* for information about *lc_init()*. The only differences are:

- *lc_init()* requires *#include "lm_code.h"*. *lc_new_job()* does not.
- With *lc_init()*, use the LM_CODE macro to create the 3rd argument to *lc_init()*. With *lc_new_job()*, the 3rd argument is instead a pointer to an uninitialized VENDORCODE struct.
- With *lc_new_job()*, the 2nd argument is unused, and is preferably 0. With *lc_init()*, it's VENDOR_NAME.
- With *lc_new_job()*, *lc_new.o* (*lc_new.obj* on Windows) must also be linked into the application.

SEE ALSO

- Section 4.27, "lc_new_job," on page 53

4.26 lc_log

SYNTAX

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

DESCRIPTION

Logs a message in the lmgrd 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

- Section 6.5, “Locating the License File,” on page 113

4.27 lc_new_job

SYNTAX

```
VENDORCODE code;
LM_HANDLE *job = (LM_HANDLE *)NULL;
status = lc_new_job(prevjob, unused, &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 lm_new.o (lm_new.obj) in also.

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().
unused	This argument is currently unused, but is there to keep the arguments the same as lc_init(). We recommending supplying a 0 argument.
pointer to (VENDORCODE) *code*	pointer to VENDORCODE struct. Initialized in this call. Used later as argument to *lc_checkout()*.

RETURN

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 daemon name is not “demo”.
LM_EXPIRED_KEYS	Vendor keys have expired.
LM_KEY_NO_DATA	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 4.25, “lc_init — (license generator only),” on page 52
- Section 4.18, “lc_free_job,” on page 46
- Section 3.5, “Multiple Jobs,” on page 21

4.28 lc_next_conf

SYNTAX

```
config = 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 license-key, 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.
(ptr to CONFIG *)*pos*	Declare CONFIG *pos = 0;.and use &pos for argument.

RETURN

(struct config *) *config*	The config structure. If none found, then NULL.
(long *) *pos*	Updated to next license file entry.

ERROR RETURNS

(See *lc_get_config()*).

EXAMPLE

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

SEE ALSO

- Section 4.4, “lc_auth_data,” on page 29

4.29 lc_perror

SYNTAX

```
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 and Windows NT 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 4.11, “lc_err_info,” on page 41

4.30 lc_set_attr

SYNTAX

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

DESCRIPTION

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

PARAMETERS

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

RETURN

(int) *status*	0—Ok, !=0, error.

ERROR RETURNS

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

SEE ALSO

- Section Chapter 5, “Controlling Licensing Behavior Of Your Application With lc_set_attr,” on page 75

4.31 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:

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

However, in this example the application must wait in the *lc_checkout* call. If the application needs to continue doing processing, you can use *lc_status()* to periodically check on the status. This might be coded in the following manner:

```
rc = lc_checkout(...,LM_CO_QUEUE,...)
switch (rc)
{
        case 0:
                break; /* got the license */
        case LM_MAXUSERS:
        case LM_USERSQUEUED:
        case LM_FEATQUEUE:
                printf("Waiting for license...");
                while (lc_status(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 4.7, “lc_checkout,” on page 31

4.32 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 is used by *lmstat -a*.

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*, since 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 structure.

The list of users returned by *lc_userlist()* includes a special record, indicated by an empty username (*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 variable job->lm_errno with the error code.

lc_userlist() only returns 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).

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.

4.33 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.
(char *) NULL	if unsuccessful.

ERROR RETURNS

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

SEE ALSO

- Section 8.3.27, “ls_vendor_msg,” on page 139

4.34 Rarely used 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.

4.34.1 lc_baddate

Obsolete and no longer needed. This check is now automatically performed when appropriate.

SEE ALSO

- Section 5.2, “LM_A_CHECK_BADDATE,” on page 76
- Section 8.3.3, “ls_a_check_baddate,” on page 134

4.34.2 lc_ck_feats

SYNTAX

```
stat = 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 to be checked.

RETURN

(int) *stat*	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	Vendorkeys do not support FEATURESET.

SEE ALSO

- Section 8.3.23, “ls_use_featset,” on page 138
- Section 6.3.7, “FEATURESET Line,” on page 110
- Section 3.5, “Multiple Jobs,” on page 21

4.34.3 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 4.17, “lc_free_hostid,” on page 45

4.34.4 lc_crypt

SYNTAX

```
CONFIG conf;
char *sdate;
LM_CODE(code, ...)
enc_code = lc_crypt(job, &conf, sdate, &vc);
```

DESCRIPTION

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.

lc_crypt() takes its 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*, and noting 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-1993");
```

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);
```

The last argument (VENDORCODE *) must be set up as in the following example:

```
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
VENDORCODE vc;
/*...*/
memcpy((char *)&vc, (char *)&code, sizeof(vc));
vc.data[0] ^= VENDOR_KEY5;
vc.data[1] ^= VENDOR_KEY5;
lc_crypt(job, &conf, l_bin_date(0), &vc);
```

Note: This use of VENDOR_KEY5 is a security precaution taken mostly for dynamic linking environments.

PARAMETERS

(LM_HANDLE *) *job*	FLEX*lm* job, from *lc_new_job()*.
(CONFIG *) *conf*	Filled-in CONFIG structure pointer.
(char *) *sdate*	Start date, in coded format. See below.
(VENDORCODE *) *vc*	Vendor's encryption seeds (from lm_code.h), where the data[0] and data[1] members have been XORd ("^") with VENDOR_KEY5.

RETURN

(char *) *enc_code*	The license key, which should match the license file.
(char *) NULL	Error.

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.

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

SEE ALSO

Section 4.10, “lc_cryptstr,” on page 39

4.34.5 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.

4.34.6 lc_display, lc_hostname, lc_username

SYNTAX

```
lc_display(job, flag)
lc_hostname(job, flag)
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

(char *)	Display name, host name, or user name.

4.34.7 lc_feat_set

SYNTAX

```
line = lc_feat_set(job, daemon, code, codes)
```

DESCRIPTION

Computes the FEATURESET *code* for the specified *daemon*, 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 *daemon*.

PARAMETERS

(LM_HANDLE *) *job*	From *lc_new_job()*.
(char *) *daemon*	The daemon desired.
(VENDORCODE *) *code*	Vendor's encryption seeds from LM_CODE macro.
(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 8.3.23, "ls_use_featset," on page 138
- Section 6.3.7, "FEATURESET Line," on page 110

4.34.8 lc_get_errno

SYNTAX

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

DESCRIPTION

The most recently set FLEX*lm* error is obtainable via l*c_get_errno()*. This can be used after any FLEX*lm* function. lc_err_info is now recommended instead, since 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 4.11, “lc_err_info,” on page 41
- Section 4.29, “lc_perror,” on page 55
- *lmclient.h*

4.34.9 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 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 6.3.7, “FEATURESET Line,” on page 110

4.34.10 lc_gethostid

SYNTAX

```
hid = 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 *) *hid*	A pointer to the hostid structure, 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

4.34.11 lc_getid_type

SYNTAX

```
hid = lc_getid_type(job, idtype)
```

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) *idtype*	The requested hostid type (see *lmclient.h*). Types are:
HOSTID_LONG	Longword hostid, e.g., SUN
HOSTID_ETHER	Ethernet address, e.g., VAX
HOSTID_USER	Username

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 *) *hid*	A pointer to the hostid structure, 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 6.7, “Special Hostids,” on page 118

4.34.12 lc_hosttype

SYNTAX

```
htype = lc_hosttype(job, benchmark)
```

DESCRIPTION

Returns information about the local host.

Some typical results when *benchmark* is set:

```
Dec Alphastation 400 4/233, Digital Unix v3: 2274
Sun Sparcstation 2, OS 4.1.x: 245
Intel 486/66, BSDI v2.x: 404
```

The number in the vendor_speed field is currently unused.

PARAMETERS

(LM_HANDLE *) *job*	From *lc_new_job()*.
(int) *benchmark*	Controls whether the internal FLEX*lm* benchmark is to be run. If benchmark is 0, no benchmark is run, otherwise the benchmark is run. The benchmark uses SIGVTALRM, and runs for approximately 0.5 seconds. If this interferes with your application, or if you do not use the benchmark result, then set benchmark to 0 when calling *lc_hosttype()*.

RETURN

(HOSTTYPE *) *htype*	A pointer to the hosttype structure, filled in for the current host. If *lc_hosttype()* cannot determine the host type, it will return a pointer to a HOSTTYPE struct with a type of LM_UNKNOWN.

ERROR RETURNS

LM_FUNCNOTAVAIL	Vendor keys do not support this function.

4.34.13 lc_isadmin

SYNTAX

```
stat = 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) *stat*	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

- Section 7.1, “lmgrd - The License Daemon,” on page 129

4.34.14 lc_lic_where

SYNTAX

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

DESCRIPTION

Returns pathname 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. **Note—***This returned string must not be modified by the caller.*
(char *)NULL	No license file set.

SEE ALSO

Section 5.13, “LM_A_LF_LIST,” on page 81

4.34.15 lc_master_list

SYNTAX

```
list = lc_master_list(job)
```

DESCRIPTION

Returns the list of server nodes.

PARAMETERS

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

RETURN

(LM_SERVER *) *list*	List of the server nodes. If NULL, error code is in *lc_get_errno().*

ERROR RETURNS

LM_NO_SERVER_IN_FILE	No SERVER lines in license file.
LM_BADFILE	Server hostname too long (> MAX_HOSTNAME).

SEE ALSO

- *lmclient.h* for the definition of *LM_SERVER* struct

4.34.16 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 *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

- Section 7.1.6, "Privileged License Administration Commands," on page 132
- Section 8.3.17, "ls_min_lmremove," on page 137
- Section 4.22, "lc_heartbeat," on page 49

4.34.17 lc_set_errno

SYNTAX

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

DESCRIPTION

The FLEX*lm* error is settable via *lc_set_errno()*. This should not normally be used, since 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

(void) none

SEE ALSO

- Section 4.11, “lc_err_info,” on page 41
- Section 4.29, “lc_perror,” on page 55
- Section 4.34.8, “lc_get_errno,” on page 64
- *lmclient.h*

4.34.18 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_FUNCNOTAVAIL	(VMS)—This function is not available with VMS.
LM_NOTLICADMIN	You are not an authorized license administrator.
LM_CANTREAD	Cannot read data from license server.

SEE ALSO

- Section 7.1.6, “Privileged License Administration Commands,” on page 132

4.34.19 lc_startup

SYNTAX

```
status = lc_startup (job, lmgrd_path, logfile, license_file)
```

DESCRIPTION

lc_startup() starts lmgrd from within your program. *lc_startup()* does not report the success of the lmgrd process. *lc_startup()* calls the appropriate *system()* call given its arguments, and is unable to detect success of the *system()* call, since the process is started in the background.

Note: *lc_startup()* is available on UNIX systems only. *lc_startup()* works for non-redundant servers only, and only if the application is running on the license server node.

PARAMETERS

(LM_HANDLE *) *job*	From *lc_new_job()*.
(char *) *lmgrd_path*	The pathname to lmgrd.
(char *) *logfile*	The daemon log file location. (Passing a NULL string or a NULL pointer will cause the Debug log file to go to stdout, which is probably not what you want.). If the first character is ‘+’, then the logfile will be appended, if it exists. If the logfile does not exist, it will be created.
(char *) *license_file*	The location of the license file (NULL for default).

RETURN

(int) *status*	0—lmgrd_path is NULL.
	<> 0—startup of lmgrd attempted.

4.34.20 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 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, -1)*, and *lc_set_attr(LM_A_RETRY_INTERVAL, -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 5 minutes, although occasional lapses are relatively harmless.

PARAMETERS

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

RETURN

num_failed_attempts	0—success; Heartbeat messages exchanged with server >0—failure; Number of failed attempts to contact the server.

SEE ALSO

- Section 8.3.18, “ls_minimum_user_timeout,” on page 137
- Section 5.27, “LM_A_UDP_TIMEOUT,” on page 86
- Section 5.3, “LM_A_CHECK_INTERVAL,” on page 77

Chapter 5

Controlling Licensing Behavior Of Your Application 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 4.30, “lc_set_attr,” on page 56.

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, “Additional lc_set_attr() Attributes” on page 205.

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 *).

5.1 LM_A_BEHAVIOR_VER

Type: char *

Default: LM_BEHAVIOR_V6

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

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

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

```
char *ls_a_behavior_ver = LM_BEHAVIOR_VX;
```

5.2 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 8.3.3, “ls_a_check_baddate,” on page 134

5.3 LM_A_CHECK_INTERVAL

Type: int

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.

Default: 120 second interval.

The results of possible settings of this variable are:

Variable	Setting	Result
check_interval	< 0	No SIGALRM timer installed.[1]
check_interval	>= 0, < 30	Old interval unchanged.
check_interval	>= 30	Timer interval.

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.

SEE ALSO

- Section 4.22, “lc_heartbeat,” on page 49
- Chapter 11, “Communications Transport” on page 151
- Section 5.25, “LM_A_SETITIMER, LM_A_SIGNAL (Unix only),” on page 85

5.4 LM_A_CHECKOUT_DATA

Type: (char *)

This option allows you to send some vendor-specific data to the vendor daemon in addition to the normal USER/HOST/DISPLAY data which is sent. 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()*. 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

1. 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()*.

independent licenses (if *LM_DUP_VENDOR* is in the duplicate mask), and to checkin 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).

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.

Default: None.

5.5 LM_A_CHECKOUTFILTER

Type: Pointer to a function returning int.

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).

Default: None.

Note: Using 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, CHECKOUTFILTER should be used with discretion.

5.6 LM_A_CKOUT_INSTALL_LIC

Type: Pint.

By default, a successful checkout automatically updates the registry $VENDOR_LICENSE_FILE setting 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

- Section , “Registry and ~/.flexlmrc,” on page 35

5.7 LM_A_DISPLAY_OVERRIDE

Type: (char *)

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.

Default: No override of display name.

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 correct 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, the *lc_set_attr(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);
```

5.8 LM_A_FLEXLOCK

Type: int

Turns on FLEXlock capability. This must be enabled to use FLEXlock, but application security is poorer.

Default: off.

See the Programmers Guide for additional information on FLEXlock.

5.9 LM_A_FLEXLOCK_INSTALL_ID

Type: short *

For additional security, each time that your application is installed, and the user activates the FLEXlock 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.

You can obtain this number by calling

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

After the FLEXlock 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 FLEXlock 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 FLEXlock functionality. (Familiarity with the registry editor is necessary for testing FLEXlock enabled features.)

See the Programmers Guide for additional information on FLEXlock.

5.10 LM_A_HOST_OVERRIDE

Type: (char *)

This string, if specified, will be used to override the hostname 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.

Default: No override of host name.

5.11 LM_A_LCM

Type: int

Used to turn off LCM.

Default: True

Note: LCM is only available on Windows.

5.12 LM_A_LCM_URL

Type: (char *)

Used to override the LCM URL default:

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

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

See the Programmers Guide for additional information on LCM.

Note: LCM is only available on Windows.

5.13 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()* only prints 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++);
}
```

5.14 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.

5.15 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.

5.16 LM_A_LICENSE_FMT_VER

Type: (char *)

Licenses generated by *lc_cryptstr()* will be compatible with the version specified. Valid arguments are LM_BEHAVIOR_V*n*, where *n* is 2, 3, 4, 5, 5_1, or 6. Note that this is not automatically set by LM_BEHAVIOR_VER 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 license key with a correct one.

See Also

- Section 4.10, “lc_cryptstr,” on page 39

5.17 LM_A_LINGER

Type: long

This option controls the license linger time for your application. Any checkout performed after setting 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 only checks for lingering licenses once per minute, which will limit the granularity of this setting.

Default: 0 (No linger).

SEE ALSO

- Section 3.4, “Lingering Licenses,” on page 20
- Section 8.4, “Vendor Daemon Support Routines,” on page 141

5.18 LM_A_LKEY_LONG

Type: int

Default: false

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.

Default: *false.*

5.19 LM_A_LKEY_START_DATE

Type: int

Default: *false*

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.

5.20 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.

5.21 LM_A_PERROR_MSGBOX (Windows only)

Type: int

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

Default: *True.*

5.22 LM_A_PROMPT_FOR_FILE (Windows only)

Type: int

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.

Default: *True.*

5.23 LM_A_RETRY_CHECKOUT

Type: int

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.

Default: *false* (for backward compatibility, but we recommend setting to *true*).

5.24 LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL

Type: int

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 re-connect 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.

The default for *LM_A_RETRY_COUNT* is 5; the default for *LM_A_RETRY_INTERVAL* is 60.

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, since there is no default behavior for exiting a Windows application.

SEE ALSO

- Section 4.22, "lc_heartbeat," on page 49
- Section 5.3, "LM_A_CHECK_INTERVAL," on page 77

5.25 LM_A_SETITIMER, LM_A_SIGNAL (Unix only)

Type: Pointer to a function returning void.

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);
```

Default: *setitimer()* and *signal()*.

5.26 LM_A_TCP_TIMEOUT

Type: int

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 timeout.

Default: 2 hours (60*60*2).

Maximum: 4 hours 15 minutes (15300 seconds). 0 means no TCP timeout.

SEE ALSO

- Section 4.22, “lc_heartbeat,” on page 49
- Section Chapter 11, “Communications Transport,” on page 151

5.27 LM_A_UDP_TIMEOUT

Type: int

This sets the amount of time (in seconds) during which a client must communicate with the vendor daemon for the UDP communications transport. If the client does not communicate within this interval (by calling *lc_timer()*), then the vendor daemon will release the license just as if an *lc_checkin()* call had been made.

Default: 45 minutes (2700).

SEE ALSO

- Section 4.22, “lc_heartbeat,” on page 49
- Section Chapter 11, “Communications Transport,” on page 151

5.28 LM_A_USER_EXITCALL

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

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)*

Default: No user exit handler (program exits).

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 4.22, "lc_heartbeat," on page 49

5.29 LM_A_USER_OVERRIDE

Type: (char *)

This string, if specified, will be used to override the username as derived from the UNIX password file. On Windows, the username is set to the hostname, but can be overridden with this attribute.

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

Default: No override of username.

5.30 LM_A_USER_RECONNECT

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

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.*

Default: No user reconnection handler.

The reconnection routine is called as follows:

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

where:	**is the:**
(char *) *feature*	Feature name.
(int) *pass*	Current attempt #.
(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 4.22, "lc_heartbeat," on page 49
- Section 5.3, "LM_A_CHECK_INTERVAL," on page 77

5.31 LM_A_USER_RECONNECT_DONE

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

This function will be called when reconnection is successfully completed.

Default: No user *reconnect_done* handler.

The reconnection done handler is called as follows:

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

where:	**is the:**
(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.

5.32 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(job, feature)
LM_HANDLE *job; /* if you want to use lc_* functions instead */
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;
                }
                fi.feat = conf;
```

```
                if (lc_get_attr(job, LM_A_VD_FEATURE_INFO, (short *)&fi))
                {
                        lc_perror(job, "LM_A_VD_FEATURE_INFO");
                }
                else
                {
/*
 *                      get specific feature info
 */
                } 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
                }
        }
```

5.32.1 Detecting OVERDRAFT for SUITEs

This is a special case for OVERDRAFT. With SUITEs, when you checkout a feature, you also silently checkout 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);
        }
```

5.33 LM_A_VENDOR_ID_DECLARE

Type: Pointer to LM_VENDOR_HOSTID struct.

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

Default: None.

SEE ALSO

- Section 6.8, “Vendor-defined hostid,” on page 120
- *lmclient.h* for LM_VENDOR_HOSTID definition.
- examples/vendor_hostid directory

5.34 LM_A_VENDOR_GETHOSTID

Type: Pointer to function returning (HOSTID *).

Obsolete. Use LM_A_VENDOR_ID_DECLARE instead.

This function gets the hostid. It should call *l_new_hostid()* to get a pointer to a malloc’d HOSTID struct. The memory will be freed when no longer needed by FLEX*lm*.

Default: None.

SEE ALSO

- Section 6.8, “Vendor-defined hostid,” on page 120
- Section 5.33, “LM_A_VENDOR_ID_DECLARE,” on page 90
- *lmclient.h* for HOSTID structure definition.

5.35 LM_A_VERSION and LM_A_REVISION

Type: short

FLEX*lm* version. Cannot be set. Only for use with *lc_get_attr()*.

Default: Version and revision of the libraries you have linked with.

5.36 LM_A_WINDOWS_MODULE_HANDLE

Type: long

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.

Default: 0

Chapter 6

License Models and the License File

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.

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-1999 uncounted AB0CC0C16807 HOSTID=DEMO ck=41
```

This indicates the expiration date and the fact that it's a demo license. The product is fully usable until January 1, 1999. 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-1998 uncounted AB1CC0916A06 HOSTID=DEMO ck=4
FEATURE f1 corp 1.0 1-feb-1998 uncounted ABDCC0116A06 HOSTID=DEMO ck=55
FEATURE f1 corp 1.0 1-mar-1998 uncounted BBDCA0D151ED HOSTID=DEMO ck=62
FEATURE f1 corp 1.0 1-apr-1998 uncoutned BBDCB0E155F1 HOSTID=DEMO ck=76
[...]
```

If the current date is February 1, 1998, then the salesperson would give an evaluator the third line, which expires in a month, March 1, 1998. 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.

PACKAGE 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 B0A0F011B491 COMPONENTS="A B C D E F"
```

Then appending a single demo FEATURE line can enable all these features:

```
FEATURE all corp 1.0 1-jan-1998 uncounted AB1CC0916A06 HOSTID=DEMO ck=4
```

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. However, for companies that are more concerned with security, it's a good idea to also 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 6.3.4, "FEATURE or INCREMENT Line," on page 100
- Section 6.3.6, "PACKAGE Line," on page 107
- Section 6.7, "Special Hostids," on page 118
- Section 4.4, "lc_auth_data," on page 29

6.2 Lenient Licensing: REPORTLOG and OVERDRAFT

6.2.1 REPORTLOG

The REPORTLOG (which is enabled with *lmswitchr* and/or an end-user options file REPORTLOG entry) provides a relatively secure method of tracking end-user usage. More and more companies prefer licensing that does not deny usage. The REPORTLOG 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*admin*, from which an invoice can be generated. (FLEX*admin* is a separate product available from GLOBE*trotter* 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 their costs resulting from OVERDRAFT usage by including a MAX_OVERDRAFT line in the options file.

LIMITATIONS

The REPORTLOG file, while ASCII (so it can be easily e-mailed), is not human-readable. In addition, any modifications to the file are detected by FLEX*admin*. However, this does not mean that no tampering is possible. There are 3 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 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 REPORTLOG periods. One example policy is: "More than *x* hours per month of missing REPORTLOG 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 overdraft state. This can be done with *lc_auth_data()* and *lc_get_attr*(... LM_A_VD_FEATINFO...). *lc_auth_data()* gives the CONFIG struct for the license that has been used for the checkout call, and LM_A_VD_FEATINFO 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 6.3.4, “FEATURE or INCREMENT Line,” on page 100
- Section 5.32, “LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO,” on page 88

6.3 Format of the License File

Please refer first to the license file description in the FLEX*lm* Programmers Guide — especially the license file examples. This gives 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.

A license file consists of the following sections:

1. Optional license server section, with information about node where the SERVER (or redundant SERVERs) are running, along with a list of all VENDOR daemon(s) that the SERVER needs to run. This section is required if any features are *counted*.
2. USE_SERVER section, indicating that client applications should not process anything except preceding SERVER lines in the license file, and should checkout the license directly from the server.
3. Features section, consisting of any combination of FEATURE, INCREMENT, UPGRADE or PACKAGE lines. This section is required.
4. Optional FEATURESET section, consisting of at most one FEATURESET line per VENDOR line in the file. This section is required for vendor daemons that use FEATURESET, but is rarely used or needed.
5. Comments. The convention is to begin comments with a ‘#’ character. However, in practice all lines not beginning with a FLEX*lm* reserved keyword are considered comments.

Note: See the FLEX*lm* Programmers Guide for information on lmcrypt and makekey, the license generation utilities. Also see Section 4.10, “lc_cryptstr,” on page 39 for generating licenses with a C function call.

Vendors and license administrators will read the license file to understand how the licensing will behave: what features are licensed, for how many users, whether these features are node-locked, if the features are demo or regular, etc. The options are very broad for what can be specified in a license file.

End-users often need to edit this file. Nearly all of the file is authenticated; if these portions are edited by the license administrator, a *LM_BADCODE* error will result. However, license administrators often must edit the license file for the following reasons:

- To change the SERVER nodename
- To change the (optional) SERVER TCP/IP port address
- To change the path to the vendor daemon

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

Only five data items in the license file are editable by the end-user:

- hostnames on SERVER lines
- (optional) port-numbers (TCP/IP) on SERVER lines
- pathnames on VENDOR lines
- options file pathnames on VENDOR lines
- Decnet object numbers (VMS) or port numbers (Unix, Windows) on VENDOR lines

---

Note: The SERVER hostid(s) and everything on the FEATURE line (except the daemon name) is input to the authentication algorithm to generate the license key for that FEATURE.

6.3.1 SERVER Line

A SERVER line specifies the node on which a license server process can run. If a license file has three SERVER lines, then two of the three servers must be running in order for the licensing system to operate. The SERVER node name in the license file can be any network alias for the node.

```
SERVER nodename hostid [optional-port-number]
```

where:	**is the:**
nodename	string returned by the UNIX hostname or uname -n commands, or an IP address in ###.###.###.### format (v5 or later). This can be edited by the license administrator. IP address is recommended for sites where NIS or DNS have trouble resolving a hostname, or if the server node has multiple network interfaces, and hence multiple hostnames. “this_host” can be used when the hostname is unknown.

	This allows the product to be installed and start the servers. Clients on the same host as the 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 hostname. Note that lminstall and lc_convert() will automatically change "this_host" to the real hostname when appropriate.
hostid	string returned by the lmhostid command. (Case insensitive). Alternate special hostids can also be specified here, including *ANY, HOSTNAME=hostname*, etc. This can be a list of hostids; if so, the server will run on any of the hostids in the list. Hostid lists are space-separated, and enclosed in quotes, e.g., "12345678 FLEXID=87654321 HOSTNAME=speedy". Note that a hostid list on the SERVER line not supported.
optional-port-number	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 a 3-server redundant license, or if the vendor daemon or clients are older than FLEX*lm* version 6. Using a port number in the range 27000-27009 is recommended when specifying a port number, since v6 utilities and clients can then use *@host* to find the server.

Notes:

- The SERVER line must apply to all lines in the license file. It is permitted to combine license files from different vendors, but this only works if the SERVER hostids are identical in all files that are to be combined.
- On VMS, the server node name can be entered with the trailing "::", if desired, as is the practice of many VMS system administrators.

SEE ALSO

- Section 6.6, "Hostids for FLEXlm Supported Machines," on page 116
- Section 6.7, "Special Hostids," on page 118

6.3.2 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: Previous to version 6, this was called a DAEMON line. DAEMON is still recognized, and DAEMON is required for lmgrds and vendor daemons older than version 6.

UNIX AND WINDOWS/NT PLATFORMS SYNTAX

VENDOR `daemon-name[path]` [`[options=]options-file`] [[*port=*]`port-number`]

where:	**is the:**
daemon-name	Name of the daemon used to serve some feature(s) in the file.
path	Pathname to the executable code for this daemon. If blank, the $PATH environment variable, plus the current directory, is used by lmgrd to start the daemon process. Prior to version 6, this path field was required.
options-file	Pathname of the (optional) license administrator specified options file for this daemon. If unspecified, the vendor daemon will also look for a file called *vendor*.opt (where vendor is the vendor daemon name) in the same directories where the license files were located. If found, this file is used for the end-user options file. This feature in new in v6, and older vendor daemons will not use *vendor*.opt.
port-number	Sites with Internet firewalls need to specify the port number the daemon uses. The default, if this *port-number* is not specified, is chosen by the system at run-time.

EXAMPLE UNIX

```
VENDOR xyzd /etc/xyzd
VENDOR xyzd /etc/xyzd /usr/local/flexlm/options/xyz.opts
```

EXAMPLE WINDOWS/NT

```
VENDOR xyzd c:\windows\system\xyzd.exe
VENDOR xyzd c:\windows\system\xyzd.exe c:\opts\xyz.opts
```

VMS PLATFORMS SYNTAX

The syntax for a DAEMON line on VMS is:

`DAEMON` *`daemon-name path`*`[object] [options_file]`

where:	**is the:**
object	DECnet object number: 0—daemon uses object number 0, task name is “name” from the DAEMON line. 128-255—daemon uses this object number. [**Default**: 0]

EXAMPLES (VMS)

```
DAEMON xyzd du:[xyz]xyzd.exe 200
DAEMON xyzd du:[xyz]xyzd.exe 0 [xyz]xyz.opts
```

Note: If an options file is specified, the object number must be specified.

6.3.3 USE_SERVER line

USE_SERVER takes no arguments. It has no impact on the server. When the client sees a USE_SERVER line, it ignores everything except the preceding SERVER lines in the license file. Thus, the USE_SERVER line should be placed after all SERVER lines in the license file. In effect, USE_SERVER forces the application to behave as though LM_LICENSE_FILE were set to *port@host* or *@host*. The advantages to USE_SERVER are:

- The client's license file does not need to match the one the server uses.
- The client's license file only requires SERVER and USE_SERVER lines.

For this reason, the USE_SERVER line is recommended.

6.3.4 FEATURE or INCREMENT Line

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 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. GLOBE*trotter* 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. If 2 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 syntax of FEATURE and INCREMENT lines in FLEX*lm* pre-v3.0 is:

```
FEATURE|INCREMENT name daemon version exp_date #lic key \
"vendor_string" [hostid]
```

The current syntax of FEATURE and INCREMENT lines (FLEX*lm* v6.1) is:

`FEATURE|INCREMENT` *`name daemon version exp_date #lic key [options...]`*

name	name given to the feature. Legal feature names in FLEX*lm* must contain only letters, numbers, and underscore characters.
vendor	vendor -name from a VENDOR line. The specified vendor daemon serves this feature.
version	latest (highest-numbered) version of this feature that is supported. (floating point format, 10 character maximum).
exp_date	expiration date in the format: {dd-mmm-yy[yy] | permanent} For example, 22-mar-2000. For no expiration, use "permanent", or an expiration date with the year as 0, e.g., 1-jan-0. 2-digit years are assumed to be 19*xx* and are valid up till 1999. For years 2000 and later, you *must* use 4 digits. 1-jan-0 = 1-jan-00 = 1-jan-0000 = permanent. FLEX*lm* *fully supports dates beyond 2000.* Prior to version 6, "permanent" was not supported.
#lic	number of licenses for this feature. Use "uncounted" or 0, for unlimited, node-locked licenses. Prior to version 6, the keyword "uncounted" was not recognized.
key	license key for this feature line. The key is produced by *lc_crypstr* in lmcrypt or makekey, or a vendor-defined routine. (Case insensitive). The start date is encoded into the key; thus identical keys created with different start dates will be different. The key field can be made case sensitive by calling `lc_set_attr(`LM_A_CRYPT_CASE_SENSITIVE`, 1)`. Remember to select the "case sensitive" option when building your daemon, if you use a case sensitive license key comparison.
"vendor-string"	Vendor-defined string, enclosed in double quotes. This string can contain any characters except a quote. White space will be ignored when ciphering this string in the *key.* The application can retrieve this string by calling *lc_auth_data()*. In the pre-v3.0 format, this is required, although it can be empty. In v3.0+, it's optional.

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

HOSTID=*hostid(s)*

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. (Case insensitive). If the number of licenses is 0, then this field is required. If hostid is DEMO, 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: DEMO, INTERNET=*n.n.n.n*, etc. This can be a list of hostid in v5 or later using space-separator and quotes, e.g.,

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

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

CAPACITY

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 checkout 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, 3 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.

DUP_GROUP=...

You can specify the Duplicate Grouping 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.

HOST_BASED[=*n*]

If HOST_BASED appears, then licenses can only be used 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 *#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.

ISSUED=*dd-mmm-yyyy*

Date issued. Can be used in conjunction with SUPERSEDE.

ISSUER="..."

Issuer of the license.

MINIMUM=*n*

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

NOTICE="..."

A field for intellectual property notices.

OVERDRAFT=*nnn*

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 FLEX*admin* reporting tool. In addition, you can determine if you are currently in an overdraft condition by calling

lc_get_attr(job, LM_A_VD_FEATURE_INFO, *...)*. The returned structure has, among others, 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”.

PLATFORMS=”*plat1*...*platn*”

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_u4, i86_n3, etc. The names can be found in the Platform Specific Notes section. 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 2 INCREMENT lines for the same feature-name, they’re are pooled separately and counted separately.

Examples:

```
FEATURE f1 ... PLATFORMS=sun4_u4
INCREMENT f2 ... 1 PLATFORMS=”i86_w alpha_u”
INCREMENT f2 ... 1 PLATFORMS=”i86_w”
```

f1 can be used on any sparc station running SunOS or Solaris.

f2 can be used on a PC running Windows (not NT) and Dec Alpha running OSF1 or DEC-Unix. There’s 1 license that can be shared between all Windows and Alpha-Unix systems, and 1 more license just for Windows. That is, at most 1 f2 can be used on the Alpha-Unix systems, but at most 2 f2s can be used by 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”.

SN=*serial-number* — Useful for differentiating otherwise identical INCREMENT lines. Its only use by FLEX*lm* is to be encrypted in the *license key.* Similar to HOSTID=ID=*n.*

START=*dd-mmm-yyyy* — Optional start date.

SUITE_DUP_GROUP=... Similar to DUP_GROUP, but affects only the enabling FEATURE for a PACKAGE SUITE. Note: If SUITE_DUP_GROUP is not specified, the parent will have the same duplicate grouping as the components.

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

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 dates* 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 license key.

The ISSUED=*date* field makes this more readable. (e.g., ISSUED=1-jan-1996). If the ISSUED date is set, then SUPERSEDE uses it, otherwise it uses the *start date*.

For example

```
INCREMENT f1 ... 1 ... ISSUED=1-jan-1995
INCREMENT f1 ... 4 ... SUPERSEDE ISSUED=1-jan-1999
```

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

```
FEATURE f1 ... 1 ... ISSUED=1-jan-1999
FEATURE f2 ... 1 ... ISSUED=1-jan-1999
FEATURE f3 ... 4 ... SUPERSEDE="f1 f2" \
       ISSUED=2-jan-1999
```

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

USER_BASED[=*n*] If USER_BASED appears, then licenses can only be used by people 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 *#lic* field. If an INCREMENT appears where some licenses are USER_BASED and some are not, the vendor daemon tracks these in separate license *pools*.

VENDOR_STRING="..." Vendor-defined license data (same as "vendor-string" in the old format, above). If checkout is to be conditioned by what's in the vendor-string, then LM_A_CHECKOUTFILTER is the best way to do this. See Section 5.5, "LM_A_CHECKOUTFILTER," on page 78.

asset_info="..." Additional information provided by the software end-user's system administrator for asset management. Not encrypted into the feature's license key or checksum (ck=*n*).

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

dist_info="..." Additional information provided by the software distributor. Not encrypted into the feature's license key or checksum (ck=*n*).

user_info="..." Additional information provided by the software end-user's system administrator. Not encrypted into the feature's license key or checksum (ck=*n*).

vendor_info="..." Additional information provided by the software vendor. Not encrypted into the feature's license key or checksum (ck=*n*).

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 4 licenses for v1 *or* 5 licenses for v2, 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 4 licenses for v1 *and* 5 licenses for v2 being available, giving a total of 9 licenses for f1.

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

```
FEATURE f1 demo 1.0 1-jan-1995 uncounted 123456789012 HOSTID=DEMO
```

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-1995.

In contrast the following FEATURE line requires a vendor daemon named “demo” (and naturally a SERVER and VENDOR as well) and

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

is limited to 5 users on any host with an internet IP address matching 195.186.*.*, and never expires.

SEE ALSO

- Section 8.3.24, “ls_use_all_feature_lines,” on page 138
- Section 8.3.7, “ls_compare_vendor_on_increment,” on page 135
- Section E.5, “LM_A_CRYPT_CASE_SENSITIVE,” on page 207
- Section 6.7, “Special Hostids,” on page 118

6.3.5 UPGRADE Line

`UPGRADE` *`name daemon fromversion version exp_date #lic code […]`*

All the data is the same as for a FEATURE or INCREMENT line, with the addition of the *fromversion* field. An UPGRADE line removes up to the number of licenses specified from any old version (>= *fromversion*) and creates a new version with that same number of licenses.

For example, the two lines:

```
INCREMENT f1 demo 1.0 1-jan-1994 5 9BFAC03164ED ck=3
UPGRADE f1 demo 1.0 2.0 1-jan-1994 2 1B9A30316207 ck=23
```

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

UPGRADE will operate on the most recent FEATURE or INCREMENT line (i.e., closest preceding FEATURE or INCREMENT line) with a version number that is >= fromversion, and < version.

6.3.6 PACKAGE Line

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

1. to license a product SUITE, or
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 parameters.

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 licenses 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* version 6. The path to the PACKAGE file should be specified in the application to support this transparently, via LM_A_LICENSE_DEFAULT.

```
PACKAGE pkg_name vendor pkg_version pkg_key COMPONENTS=pkg_list \
                        [ OPTIONS=pkg_options ]
```

where:	**is the:**
pkg_name	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	version of the PACKAGE. The corresponding FEATURE/INCREMENT line must have the same version.
pkg_key	key generated by one of the license generators: makepkg, lmcrypt, or the vendor's customized license generator.

COMPONENTS=*pkg_list*	the space separated list of components. The format of each component is:
	`feature[:version[:count]]`
	The PACKAGE must consist of at least one COMPONENT. *version* and *count* are optional, and if left out, their values come from the corresponding FEATURE/INCREMENT line. *Count* 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 comp1 comp1:2.0:4"`

OPTIONS=*pkg_options*	Currently the only legal option is SUITE. This is what distinguishes a SUITE PACKAGE from a PACKAGE used to ease distribution. With OPTIONS=SUITE, the *pkg_name* FEATURE is checked out in addition to the component feature being checked out.

If OPTIONS=SUITE is not set, then the FEATURE of the same name as the package is removed once the PACKAGE is turned on, and it also is not checked out when a component feature is checked out.

EXAMPLES

Example 1:

```
PACKAGE office demo 1.0 20CHARCODEXXXXXXXXXX\
       COMPONENTS="comp1 comp2" OPTIONS=SUITE
FEATURE office demo 1.0 permanent 5 20CHARCODEXXXXXXXXXX
```

This is a typical SUITE example. The user will have 2 features: comp1 and comp2, which are each version 1.0, with 5 non-expiring licenses available. When comp1 or comp2 are checked out, “office” will also be checked out. The vendor will most likely want to turn on DUP_GROUP (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.

Example 2:

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

This is a good way to distribute multiple node-locked, counted licenses. Rather than requiring 5 INCREMENT lines per node, only one INCREMENT line is required per node, and the features are indicated in the PACKAGE line.

Example 3:

```
PACKAGE office demo 1.0 A30483555898\
       COMPONENTS="comp1:1.5:2 comp2:3.0:4 comp3"
FEATURE office demo 1.0 1-jan-1995 32C817A5100D8 ISSUER=distrib1
```

The component versions override the FEATURE versions, and the number of licenses available for any component is the product of the 3 licenses for office and the number of licenses of that component. The result is equivalent to:

```
FEATURE comp1 demo 1.5 1-jan-1995 6 7649CFAF16DB ISSUER=distrib1
FEATURE comp2 demo 3.0 1-jan-1995 12 63992D55345B ISSUER=distrib1
FEATURE comp3 demo 1.0 1-jan-1995 3 0D037EACC547 ISSUER=distrib1
```

SEE ALSO

- Section 5.15, “LM_A_LICENSE_DEFAULT,” on page 82

6.3.7 FEATURESET Line

The FEATURESET line is required if *ls_use_featset* is set in `lsvendor.c`. Otherwise, FEATURESET is unused.

```
FEATURESET daemon-name key
```

where:	is the:
daemon-name	Name of the daemon used to serve some feature(s) in the file.
key	License key for this FEATURESET line. This key encrypts the keys of all FEATUREs that this 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 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.

Notes:

- The use of FEATURESET is discouraged.
- The SERVER hostid(s) and everything on the FEATURE line (except the daemon name) is input to the authentication algorithm to generate the code for that FEATURE.

SEE ALSO

- Section 8.3.23, "ls_use_featset," on page 138
- Section 6.3, "Format of the License File," on page 96

6.3.8 Comments

Comment lines can begin with a '#'. Currently, all lines not beginning with a license keyword are comment lines. Therefore, typically, license files can be sent as e-mail messages.

6.3.9 Continued Lines

Lines can be continued with a "\" character.

6.3.10 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-1999 10 1AEEFC8F9003
FEATURE f2 demo 1.0 1-jan-1999 10 0A7E8C4F561F
```

6.4 Decimal Format Licenses

Licenses can be represented in decimal format, to make license delivery easier for customers without access to e-mail. 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* arg 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.

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.

6.4.1 Decimal Format Limitations

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

FEATURESET, which is normally discouraged, 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 would be much longer (up to three times longer) in the decimal format, defeating the purpose of a decimal format, which is to make licenses easier to type in.

FEATURE names including '-' cannot be represented in decimal format. These are technically unsupported characters, although some companies have used them.

6.4.2 Example decimal licenses:

COUNTED LICENSE:

```
SERVER this_host 12345678
VENDOR demo
FEATURE f0 demo 1.0 permanent 1 A7F6DFD8C65E
FEATURE f1 demo 1.0 permanent 1 AA8BD581EE6
```

Decimal:

```
demo-f0-49473-46846-62783-50738-57369-53593-728
demo-f1-02369-24236-07508-18291-8
```

Note that the first decimal line includes the SERVER/VENDOR information, and the 2nd (and any subsequent lines) are much shorter.

DEMO LICENSE:

```
FEATURE f2 demo 1.0 1-jun-1998 uncounted 6E06CC47D2AB HOSTID=DEMO
```

Decimal:

```
demo-f2-02753-41313-52979-22534-1299
```

6.4.3 Format of a Decimal License

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

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

vendor	is the vendor daemon name
feature	is the feature name
#####	Groups of 5 decimal numbers (0-9) followed by a hyphen. The last group may be less than 5 digits.

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

6.4.4 Hints on Using the Decimal Format

There are some “tricks” that are used internally to make decimal lines short. 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 5 characters in the decimal format. VENDOR_STRING=abcd (valid hex characters) will also consume about 5 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: Note: Mixed case hex characters will not be stored efficiently. VENDOR_STRING=abcD will take about 12 decimal characters, instead of 5.

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 since 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 FLEX*lm* Programmer's Guide or End-User Manual

6.5 Locating the License File

The rules for finding the license file are:

1. Look in the default location:

```
/usr/local/flexlm/licenses/license.dat (Unix)
C:\FLEXLM\LICENSE.DAT (Windows)
SYS:\FLEXLM\LICENSE.DAT (Netware)
SYS$COMMON:[SYSMGR]flexlm.dat (VMS)
```

2. If either $LM_LICENSE_FILE or $*VENDOR*_LICENSE_FILE 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 ~/.flexlmrc (Unix). If set in both locations, both are used.
3. 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.
4. If application sets lc_set_attr(..., LM_A_DISABLE_ENV...), then environment variables are ignored. (Not recommended except in special circumstances).
5. The license location(s) can be set in the application with LM_A_LICENSE_FILE or LM_A_LICENSE_FILE_PTR or LM_LICENSE_DEFAULT. If any of these are set, the default location is ignored. LM_A_LICENSE_DEFAULT is normally recommended, since 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.
6. 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.
7. 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, licensepath1);
   lc_set_attr(..., LM_A_LICENSE_DEFAULT, licensepath2);
   ```

   Only *licensepath2* is used.

   ```
   lc_set_attr(..., LM_A_LICENSE_FILE, licensepath1);
   lc_set_attr(..., LM_A_LICENSE_DEFAULT, licensepath2);
   ```

   Again, only *licensepath2* is used.
8. For the Simple and Trivial APIs, the rules are the same, with the rule that the license-path argument behaves like LM_A_LICENSE_DEFAULT.
9. The *-c* option will override the setting of *LM_LICENSE_FILE* for all FLEX*lm* utilities such as: *lmgrd*, *lmdown*, *lmstat* etc.

6.5.1 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, a semi-colon on Windows, or a space on VMS.When assigning a license file path on VMS, use a command similar to the following:

  $ assign "file_1 file_2 file_3" lm_license_file.

  where the quotes around all the license file names are required.
- 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 only *.LIC files are not recognized.
- *@host*, where *host* is the hostname of the license server, when the SERVER has no port number, or the port number is between 27000 and 27009. (*New* 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 hostname come from the SERVER line
- *port@host*,*port@host*,*port@host*, when a 3-server redundant server is being used.
- The actual license file text, with "START_LICENSE\n" as a prefix, and "\nEND_LICENSE" as suffix, where the embedded newlines are required.

6.5.2 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.

Aside from being convenient, this is an important method of redundancy, and has many advantages over the more formal 3-server redundancy. It can also be used in combination with 3-server redundant systems.

A non-redundant server would have only a single *@host*, whereas redundant servers could be specified as 3 sets of *port@host**[1], separated by commas. 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

1. Note that redundant servers require a port number and can therefore use *port@host*, but not simply *@host*. That's why as examples, *@host* is used with a single server, but *port@host* is used with redundant.

```
27000@serverhost
```

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

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

or, if the second server was a set of 3 redundant servers, the path might look like this:

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

In this last example, the "*1700@host1*,*1700@host2*,*1700@host3*" part specifies a set of 3 redundant servers.

Note: Both the client and server need to be reading the SAME license file, since the client passes the license key from the FEATURE line to the vendor daemon. However, *@host*, *port@host* or USE_SERVER solve this problem, since the client never reads the license file features in these cases.

SEE ALSO

- Section 4.30, "lc_set_attr," on page 56
- Chapter 9, "Redundant License Servers" on page 143

6.6 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 6-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 and Window NT systems is the ethernet address of the system. FLEX*lm* also supports several other hostids as well as hardware keys available from GLOBE*trotter* Software. For more details, see Section 13.7, "Node Lock and Hostid for Standalone PCs," on page 163.

The VMS version of FLEX*lm* includes support for certain specified devices. For more information, see Section 12.8.4, "VMS Ethernet Device Support," on page 159.

The program lmhostid will print the exact hostid that FLEX*lm* expects to use on any given machine. The following table lists alternate methods to obtain the required hostid for each machine architecture:

Hardware Platform	**Hostid**	**Type this command on the license server:**	**Example**
Apollo	20-bit node ID	*lcnode -me*	BE70
Alpha Dec Unix	ethernet address	*netstat -i. Use Address column, and remove ':'*	08002be5a721
HP	32-bit hostid	*echo 'uname -i' 16o p | dc*	778DA450
	ethernet address	*lanscan* (use station address without leading "0x")	0000F0050185
Linux	Ethernet	*/sbin/ifconfig eth0.* Use HWaddr, and remove colons, e.g., 00:40:05:16:E5:25 becomes 00400516e525	00400516e525
RS/6000—PPC running AIX	32-bit hostid	*uname -m* (returns 000276513100), then remove last 2 digits, and use remaining last 8 digits	02765131
SCO (v3 and later)	Hostid String	`uname -x` (returns SCO00354), then prefix with `"ID_STRING="`	ID_STRING=SCO00354
SCO (FLEX*lm* pre-v.3.0)		`uname -x` (returns SCO00354), then remove any non-hex proceeding letters from the serial number	00354
SGI	32-bit hostid	*echo '/etc/sysinfo -s' 16o p | dc*	69022F72
SUN	32-bit hostid	*hostid*	170a3472

Hardware Platform	Hostid	Type this command on the license server:	Example
VAX/VMS Alpha/OpenVMS	ethernet address	*ncp*	0800200055327
Windows	ethernet	*lmutil lmhostid*	0800200055327
	DISK_SERIAL_NUM—32-bit hostid	`DIR C:\` (look for "Volume Serial Number is", and remove '-')	DISK_SERIAL_NUM=1CA25283
	FLEXid dongles	*lmutil lmhostid -flexid* (Also printed on the dongle)	FLEXID=7-B2850003
	Intel Pentium III+ CPUID (FLEXlm v7.0d+)	*lmhostid -cpu[32|64|96]*	1B34-A0E3-8AFA-6199-9C93-2B2C (96-bit) 8AFA-6199-9C93-2B2C (64-bit) 9C93-2B2C (64-bit)

SEE ALSO

- Section 13.7, "Node Lock and Hostid for Standalone PCs," on page 163
- Section 4.17, "lc_free_hostid," on page 45
- Section 4.3, "l_new_hostid," on page 29
- Section 6.6, "Hostids for FLEXlm Supported Machines," on page 116
- Section 6.9, "Intel Pentium III+ Hostid (HOSTID_INTEL)," on page 121

6.7 Special 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 and NT only).
DISPLAY=*disp*	Locks the software to display "*disp*".

FLEXID=*sn*	Locks the software to a PC with a hardware key (dongle) of serial number *sn*. (Windows and NT only).
HOSTNAME=*node*	Locks the software to computer hostname "*node*".

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 node (on the FEATURE/INCREMENT line)—anywhere a hostid can be used. 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=*nnn.nnn.nnn.nnn*	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. **WARNING**: The Internet hostid can be used on the SERVER line, as a hostid to lock the vendor daemon. In this case, the wildcards should not be used, otherwise the customer could easily start license managers on more than one node and obtain "extra" licenses.
USER=*name*	Locks the software to username "*name*".

EXAMPLES

```
FEATURE f1 demo 1.0 1-jan-1999 uncounted AB28E0011DA1 \
        HOSTID=HOSTNAME=globes
```

or

```
FEATURE f1 demo 1.0 1-jan-1999 uncounted EB78201163B0 \
        HOSTID=USER=joe
```

SEE ALSO

- Section 6.6, "Hostids for FLEXlm Supported Machines," on page 116
- Section 6.3.4, "FEATURE or INCREMENT Line," on page 100

- Section 6.3.1, “SERVER Line,” on page 97
- Section 13.7, “Node Lock and Hostid for Standalone PCs,” on page 163

6.8 Vendor-defined hostid

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
#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(id)
HOSTID *id;
{
  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.)

```
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).

6.9 Intel Pentium III+ Hostid (HOSTID_INTEL)

REQUIREMENTS:

- FLEX*lm* v7.0d+
- Windows
- CPU hostid must be enabled.

6.9.1 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.

6.9.2 Hostid length

The true CPUID is a 96-bit value, in the format

```
XXXX-XXXX-XXXX-XXXX-XXXX-XXXX
```

where the X's are uppercase hex characters. According to Intel, all 96-bits (24 hex characters) are required to acheive 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 can be 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.

6.9.3 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

6.9.4 lmtools and lmhostid

lmhostid takes the following arguments:

-cpu	32-bit hostid
-cpu32	32-bit hostid
-cpu64	64-bit hostid
-cpu96	96-bit hostid

6.9.5 Security issues

Where available, this is the preferred hostid, since 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 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.

6.10 Types of License Files

Depending on the information in this file, the contents will be interpreted differently by FLEX*lm*. The license file supports network licensing, node locking, network licensing on a limited set of hosts, and demo/evaluation software. See the FLEX*lm* Programmers Guide and *examples/licenses* for examples of different types of license files.

6.11 License Key Length and Start Date

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 version 6, 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

- lc_set_attr(job, LM_A_LKEY_LONG, (LM_A_VAL_TYPE)1), or
- lc_set_attr(job, LM_A_BEHAVIOR_VER, (LM_A_VAL_TYPE) *behavior*); where *behavior* is LM_BEHAVIOR_V5_1 or less, or
- 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 2 ways:

1. Instead of a 64-bit security key on each feature line, there's a 48-bit security key.
2. The 20-character license key included 4 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 2 ways:

1. There is now an optional "START=" attribute for FEATURE/INCREMENT/UPGRADE lines. This is the preferred method for a start date.
2. 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. (This is to remove any ambiguity about what the key contains.)

6.11.1 Changing license key behavior

Here's how to turn on long license keys and/or license key start dates in applications, license generators and vendor daemons:

APPLICATION AND LICENSE GENERATORS:

long license keys:

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

hidden start dates:

```
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.

Vendor daemon (lsvendor.c in machind directory):

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

6.11.2 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, 0) and lc_set_attr(job, LM_A_LKEY_START_DATE, 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.

6.12 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., or lp_checkout can specify the actual license, 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 50A35101C0F3 HOSTID=ANY \
        VENDOR_STRING=”Acme Inc"\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 50A35101C0F3 HOSTID=ANY \
        VENDOR_STRING=”Acme Inc”\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 and which 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*).

6.13 E-mailing licenses

E-mailers can and do alter license files. We attempt to accommodate most e-mailer alterations, but not all are accommodated.

6.13.1 Newline additions

E-mailers often insert newlines into text, such as a license file. With v7+, this is okay and 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 e-mail headers intact, and this is a good way to recommend saving a license file.

6.13.2 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+ FLEXlm ignores this suffix. That is, if a file called “demo.lic” is in the license path, and “demo.lic.txt” is found, this will be used. If both demo.lic and demo.lic.txt exist, both are used

6.13.3 Other transformations

QUOTES

Ascii quote 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, or Rich text, or any other similar encoding, FLEXlm will not recognize the license file, and should be avoided.

6.14 Order of lines in license file

In some cases, the order of FEATURE and INCREMENT lines in a license file can matter. The order of the other lines (VENDOR, SERVER, PACKAGE) do not matter.

The application will attempt a checkout with each FEATURE/INCREMENT line that matches the feature name. For example:

```
FEATURE f1 xyzd 1.0 permanent uncounted 707042C8BEB9
               HOSTID=08002b32b161
FEATURE f1 xyzd 1.0 permanent 5 707042C8BEB9
```

In this example, the uncounted, node-locked license will be attempted before the counted, floating license. This is preferable: if the user is on host 08002b32b161, it’s better to not consume any of the counted licenses. If the licenses were in the reverse order, the uncounted licenses would only get used when all the floating licenses were used. If a user on host 08002b32b161 were using a floating licenses, then users on other hosts would get denied licenses unnecessarily.

For this reason, in v7+, licenses are, internally, automatically sorted so that many of the most common license order problems are avoided. The sort is as follows:

1. License file. This means that the automatic sorting does not occur between 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 any of these 3 dates: ISSUED=, START=, hidden start-date in license-key.

6.15 sort=*nnn*

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.

Chapter 7

lmgrd

7.1 lmgrd - The License Daemon

The purpose of lmgrd is to:

- start and maintain all the vendor daemons listed in the VENDOR lines of the license file, and
- 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 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. The command line for lmgrd is:

```
lmgrd [-app] [-2 -p] [-c license_file_list] [-l logfile]
  [-s timestamp_interval] [-t timeout_interval] [-x lmdown|lmremove ]
  [-v] [-z]
```

where:	**is the:**
-app (Windows only)	32-bit Windows (NT and 95) only—to run *lmgrd.exe* in a DOS window or console.
-p -2 (Unix only)	Restricts usage of lmdown, lmreread, and lmremove to a FLEX*lm* administrator who is by default root. If there is a UNIX group called “lmadmin” then use is restricted to only members of that group. If root is not a member of this group, then root does not have permission to use any of the above utilities. (Default: no license administrator required). Note that all Windows and VMS users are automatically considered part of the lmadmin group.
-c *license_file_path*	license file or directory path(s). (default: */usr/local/flexlm/licenses/license.dat* on UNIX systems, or *C:\FLEXLM\LICENSE.DAT* on Windows and Windows NT systems). If a directory is specified, all files matching **.lic* are used. This list is colon-separated on Unix, and semi-colon separated on Windows. If redundant servers, must be a single file. Server line hostids for all files must apply to the same host, but the hostids need not be identical.

-l *logfile*	Debug log file produced by the daemons. (default: stdout).
-s *timestamp_interval*	logfile time stamp interval, in minutes (default: 360 minutes).
-t *timeout_interval*	time interval (in seconds) during which daemons must complete connections to each other. The default value is 10 seconds. A longer value may be desirable if the daemons are being run on busy systems and/or a very heavily loaded network.
-x lmdown | lmremove	disable lmdown or lmremove commands.
-z	Run in the foreground. lmgrd runs in the background by default.
-v	Prints lmgrd's version number and copyright, then exit.

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

7.1.1 No lmgrd on VMS or Netware

Neither VMS nor Netware versions of FLEX*lm* use lmgrd, so clients connect directly with the vendor daemons. This implies that each vendor daemon is started independently as described in the next two sections.

7.1.2 Starting vendor daemons on VMS

The command line for starting a daemon is:

```
daemon_name [-c license_file_path] [-l log_file_path]
```

where:	**is the:**
daemon_name	name of the daemon.
license_file_path	an alternate license file, if the *LM_LICENSE_FILE* environment variable is not set or if you want to override the setting of *LM_LICENSE_FILE*.
log_file_path	an optional log file name. If this option is not selected, the logging information will be written to the screen.

The default license file on VMS is *SYS$COMMON:[SYSMGR]flexlm.dat*. You can set a new default in the vendor daemon by editing *lsvendor.c* and changing the value of the (char *) variable *default_license_file* to be the pathname you wish for the default license file for your daemon. The end-user's license administrator can still override this setting with either the "-c" command-line option, or by setting the *LM_LICENSE_FILE* logical name.

Note: On VMS, the SYSNAM and SYSLCK privileges are required to run the daemons.

7.1.3 Starting Vendor Daemons on Netware

Your vendor daemon can be started at the Netware console by loading it:

```
> load vendor.nlm
```

assuming your vendor daemon was named *vendor.*

You can have it automatically start at power up by including a reference to it in the *AUTOEXEC.CNF*

7.1.4 Starting lmgrd on Windows NT

lmgrd can be started as an application from the Windows NT console with the *-app* option. For example:

```
D:\flexlm> lmgrd -app -c my_lic.dat
```

Alternatively, lmgrd can be installed as a service on a Windows NT system so that it can be managed from the control panel on NT. You can use the FLEX*lm* Control Panel (described in the FLEX*lm* Programmers Guide) to install lmgrd as a service. This is the recommended technique. If you prefer to do a more manual installation of lmgrd as a service, the remainder of this section describes how to accomplish this.

To install lmgrd as a service manually (rather than using the supplied FLEX*lm* Control Panel), use the *INSTALL.EXE* command provided by FLEX*lm*. For example:

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

Note that the full pathname to *lmgrd.exe* must follow the install 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, *c:\flexlm\license.dat*.

After *INSTALL.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 *INSTALL.EXE* without rebooting your system, you may use the Service Icon from the Windows NT control panel. Look for *FLEXlm Licensing Service* from the dialogue box after you double click on the Service icon.

To remove lmgrd from the registered service list, simple type *install remove.* If you wish to customize the *INSTALL* program, the source code and makefile for *INSTALL.EXE* is included in the *UTILS* 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*

Select *HKEY_LOCAL_MACHINE:SYSTEM:*
CurrentControlSet:Control:Session Manager:Environment

From the menu bar, select Edit and then 'Add Value'

When the Add Value dialog box is displayed, enter LM_LICENSE_FILE as Value Name, and select REG_SZ as Data Type. Click OK.

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.

Exit Registry Editor.

3. Shutdown 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.

7.1.5 Pre FLEX*lm* v2.0 Startup Arguments

The startup arguments for vendor daemons changed between FLEX*lm* v1.x and v2.0. If you are running a configuration with a mixture of v1.x and v2.x vendor daemons, then you will need to use the *-b* option (**Note:** this is the default in FLEX*lm* v3.0 and later). Note that it is very rare to encounter FLEX*lm* v1.x daemons.

7.1.6 Privileged License Administration Commands

If the “-2 -p” startup argument is used to lmgrd, then the user who issues an lmdown, lmreread, or lmremove command must be a “license administrator”. A license administrator is:

- a member of the “lmadmin” group, or, if no lmadmin group exists,
- a member of group 0.

Chapter 8

Configuring Your Vendor Daemon

The normal installation program (install_flexlm.ftp on Unix, or setup on Windows) builds your vendor daemon. The vendor daemon can be modified, via variables in mac*hind/lsvendor.c*, but these are normally not suggested or required. Most of the variables in this file appear for historic and compatibility reasons, and should *not* be used except where required for compatibility reasons.

When you have completed the necessary edits to *lsvendor.c*, the makefile will create *lmrandom* and then run the *lmrandom* program to randomize your daemon configuration information (for additional security), then it builds your vendor daemon. A temporary file *lsr_vendor.c* is created with the randomized data. One of the functions of *lmrandom* is to remove the vendor name from the daemon executable file; the daemon name is constructed at run-time.

Note: You can use the CONFIG_DAEMON -x script to automate editing the *lsvendor.c* file, although we recommend editing the file with a text editor.

8.1 Building Your Vendor Daemon—VMS Systems

To build your vendor daemon (assuming the FLEX*lm* kit is in *[lmgr]):*

```
$ set def [lmgr]
```

edit *lsvendor.c* and *lm_code.h* to select your options and enter your vendor keys and encryption seeds.

```
$ @make
```

If you have added initialization or callback routines, you can edit *make.com* to add your new object files.

8.2 Building Your Vendor Daemon—Windows NT Systems

build.bat will build your vendor daemon. If you want to build it “by hand” use the following commands (assuming the FLEX*lm* kit is in *C:\flex_sdk*):

```
C:> cd \flex_sdk\i86_n3 (for Intel NT)
C:> nmake /f lmrandom.mak
```

```
C:> lmrandom < lsvendor.c > lsr_vend.c
C:> nmake /f demo.mak
C:> ren demo.exe your_vendor_daemon_name.exe
```

8.3 lsvendor.c variables

Instead of using *CONFIG_DAEMON -x*, you can edit *lsvendor.c*. We do not recommend modifying any of these variables. Under nearly all conditions, this file should be left as-is. Most of the variables exist for historic reasons.

8.3.1 default_license_file (VMS only)

This variable is set to the pathname of the (new) default license file, if the software vendor desires to override the FLEX*lm* default of:

```
SYS$COMMON:[SYSMGR]FLEXLM.DAT.
```

8.3.2 ls_a_behavior_ver

This can be set to LM_BEHAVIOR_V*x*, where *x* is 2, 3, 4, 5, 5_1 or 6. The default (0) is LM_BEHAVIOR_CURRENT, which is V6 in Version 6.

SEE ALSO

- Section 5.1, “LM_A_BEHAVIOR_VER,” on page 76

8.3.3 ls_a_check_baddate

Default is 0. If 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 5.2, “LM_A_CHECK_BADDATE,” on page 76

8.3.4 ls_a_license_case_sensitive

If 1, licenses are case-sensitive. Default is 0, not case-sensitive.

SEE ALSO

- Section 5.14, “LM_A_LICENSE_CASE_SENSITIVE,” on page 82
-

8.3.5 ls_a_lkey_long

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

SEE ALSO

- Section 5.18, “LM_A_LKEY_LONG,” on page 83

8.3.6 ls_a_lkey_start_date

If 1, license keys contain a hidden start-date, and are 4-characters longer. Default is 0, no hidden start-date.

SEE ALSO

- Section 5.19, “LM_A_LKEY_START_DATE,” on page 83

8.3.7 ls_compare_vendor_on_increment

```
int ls_compare_vendor_on_increment = 0; /* Compare vendor-defined */
```

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.

SEE ALSO

- Section 6.3.4, “FEATURE or INCREMENT Line,” on page 100

8.3.8 ls_compare_vendor_on_upgrade

```
int ls_compare_vendor_on_upgrade = 0; /* Compare vendor-def fields */
```

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 6.3.5, “UPGRADE Line,” on page 107

8.3.9 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. This parameter has no effect on VMS or Netware versions of FLEX*lm*.

8.3.10 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.16, “LM_A_USER_CRYPT,” on page 211

8.3.11 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.

8.3.12 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 filesystem 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.

Note: The check performed by *ls_do_checkroot* will FAIL on a diskless node. This prevents diskless nodes from acting as license servers. GLOBE*trotter* 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.

8.3.13 ls_dump_send_data

```
int ls_dump_send_data = 0; /* Set to non-zero value for debug output */
```

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

8.3.14 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.

8.3.15 ls_infilter

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

To install a vendor-defined checkin filtering routine, initialize *ls_infilter* with a pointer to your routine. The checkin filter is called with no parameters. If it returns 0, the current checkin is aborted; a return of 1 allows the current checkin to continue. If the filter aborts the operation (returns 0), then it should set the error code, 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 8.4, “Vendor Daemon Support Routines,” on page 141.

8.3.16 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 8.4, “Vendor Daemon Support Routines,” on page 141.

8.3.17 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).

8.3.18 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.

8.3.19 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.

8.3.20 ls_outfilter

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

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 described in Section 8.4, “Vendor Daemon Support Routines,” on page 141.

Note: Please contact Globetrotter support before using ls_outfilter. Callbacks in this area are rarely needed, and we’re happy to provide assistance when they are.

8.3.21 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.

8.3.22 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.

8.3.23 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.

SEE ALSO

- Section 6.3.7, “FEATURESET Line,” on page 110

8.3.24 ls_use_all_feature_lines

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

Note: GLOBE*trotter* 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 INCREMENT lines.

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 6.3.4, "FEATURE or INCREMENT Line," on page 100

8.3.25 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.

8.3.26 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.

8.3.27 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 4.33, "lc_vsend," on page 59

8.3.28 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 4.10, “lc_cryptstr,” on page 39
- Section E.16, “LM_A_USER_CRYPT,” on page 211

8.3.29 ls_user_lockfile

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

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

Unix	/usr/tmp/lock*daemon*. On some newer systems, including DEC Alpha, the location is /usr/tmp/.flexlm/.lock*daemon.*
Windows (32-bit)	C:\FLEXLM*daemon*
Netware	SYS:\\FLEXLM*daemon*
VMS	Unused in VMS, since the VMS lock manager is used.

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 6 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 pathname for this file (i.e. on Unix the pathname should start with “/”) otherwise, multiple vendor daemons could be run from different directories.

8.3.30 Vendorkeys and Vendor Encryption Seeds

```
VENDORCODE vendorkeys[] =.....
```

Example from *lm_code.h*:

```
/*
*       VENDOR’s private encryption seeds – must be changed by vendor
*       vendor should make up two unique 32-bit hex numbers.
*/
#define ENCRYPTION_SEED1 0x87654321
#define ENCRYPTION_SEED2 0x12345678
/*
 *      FLEXlm vendor keys
 */
#define VENDOR_KEY1 0xe7503c03
#define VENDOR_KEY2 0x75811bed
```

```
#define VENDOR_KEY3 0xb8ea89c8
#define VENDOR_KEY4 0xc9a1b6d3
#define VENDOR_KEY5 0xc5dda903
```

The ENCRYPTION_SEEDs are to be made up by the FLEX*lm* developer; the VENDOR_KEYS are supplied by GLOBE*trotter* Software when you purchase or upgrade FLEX*lm*.

CONFIG_DAEMON edits lm_code.h (or you can edit it manually) with the encryption seeds and vendor keys that you specified for your daemon. When your vendor daemon comes up, it checks all feature lines that specify its daemon name. If the license key in the license file is valid for any feature lines that the daemon checks, and the hostid is correct, then the vendor daemon sets the number of licenses to the value in the license file. All components of FLEX*lm* must share the same lm_code.h, where the VENDORCODE is specified. *This encryption seed is the unique part of your daemon*, so it is wise to

- keep it confidential
- make sure the encryption seeds are unique numbers you have made up.

Each entry in the VENDORCODE array consist of a pair of 32-bit vendor-defined encryption seed values and a set of five 32-bit GLOBE*trotter* Software supplied FLEX*lm* Vendor Keys. The two 32-bit vendor-defined encryption seeds should be set to any values that do not contain long strings of either 0 or 1 bits (e.g. do not use 0x5 or 0xffffff3f). The FLEX*lm* Vendor Keys supplied by GLOBE*trotter* Software enable various portions of the FLEX*lm* software to run. Both the first set of vendor-defined encryption seeds and the FLEX*lm* Vendor Keys will be defined in the include file lm_code.h by *CONFIG_DAEMON*.

Note: Only the encryption seeds, which you define, affect the licenses you create. The FLEX*lm* vendor keys have no effect on the license files you create.

8.4 Vendor Daemon Support Routines

ls_get_attr is required for applications using the *ls_outfilter()* callback:

```
ls_get_attr(attr, &value)
```

where:	**is the:**
attr	an attribute specified in *ls_attr.h*
value	(char *)—the 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 available for use in daemon filter routines:

```
ls_checkout(feature, ndesired, wait, who, version, server, dup_sel,
linger, key, 0,0);
```

where:	**is the:**
(char *) *feature*	feature desired
(char *) *ndesired*	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*	Feature’s version number
(SERVERNUM) *server*	Server requesting the checkout
(char *) *dup_sel*	Duplicate license selection criteria
(char *) *linger*	How long license is to linger
(char *) *key*	License key 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.

Chapter 9

Redundant License Servers

FLEX*lm* supports two methods of redundancy: A set of three redundant license servers, and redundancy via a license file list in the $LM_LICENSE_FILE setting.

With three-server redundancy, if any two of the three license servers are up and running, the system is functional and hands out its total complement of licenses (Two out of three license servers is referred to as a "quorum").

With $LM_LICENSE_FILE, the application attempts a checkout from every server in the license file list. When used for redundancy, the ISV divides the licenses among any number of servers, and the end-user set $LM_LICENSE_FILE to address all the servers.

9.1 Three-server redundancy

9.1.1 Selecting Server Nodes

If all the end-user data is on a single file server, then there is no need for redundant servers, and GLOBE*trotter* 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.

These three redundant servers should be on the same local-area network, and should have excellent communications. This form of redundancy requires that the servers exchange heartbeats periodically, and poor communications can cause performance problems.

9.1.2 Generating a license file for redundant servers.

To generate a license file that uses redundant servers, simply specify three servers when you create your license. Note that the use of two license servers is not recommended, since a license file with two servers would require that you always had both running.

Note: The VMS and Netware versions of FLEX*lm* do not support redundant servers.

When redundant servers are started, they elect a *master,* which performs all licensing operations. The other one or two servers are there to provide a secure licensing mechanism in the event of hardware failure or in case the server node needs to be rebooted. Should the master fail, if two servers are still running, one of the remaining two will be elected master, and licensing operations will continue.

The order of SERVER lines in the license file (for redundant servers) specifies the end-user's desired selection order for the master server node. If the order of these lines does not agree on all three server nodes, then FLEX*lm* uses alphabetical order to determine the master, and the following messages are generated in the log file:

```
6/26 11:00 (lmgrd) License File SERVER line order mismatch.
6/26 11:00 (lmgrd) Using alphabetical order
```

If the server order does not match, the daemons will come up initially, but reconnection in the event of server node failure may not work, depending on which node fails and who was the master before the failure. If the automatic failover in the event of node failure is important, please ensure that the order of the server lines is consistent in the license file on all server nodes.

When only two of the three license server machines are up, it is possible for the client to experience a timeout before connecting to the license server. Specifically, if the first license server in the license file is down, the client will timeout before attempting to connect to the second server in the license file. This timeout is set to 10 seconds by default, so there will be a 10-second delay before the license is granted. If the first server is to be down for a while, the order of the SERVER lines in the license file which the client reads could be changed to avoid this timeout.

9.1.3 Sample three-server license file

```
SERVER pat 17003456 1700
SERVER lee 17004355 1700
SERVER terry 17007ea8 1700
VENDOR demo /etc/mydaemon
FEATURE f1 demo 1.0 1-jan-1999 10 1AEEFC8F9003
FEATURE f2 demo 1.0 1-jan-1999 10 0A7E8C4F5613
```

Note that there may need to be 3 separate versions of this file, identical, except for the path to the VENDOR, which may be different on nodes "pat," "lee" and "terry."

9.2 Redundancy via License File List in $LM_LICENSE_FILE

This is best explained by example. In the previous license, there is a count of 10 for both f1 and f2. For redundancy, the ISV would issue 2 licenses with a count of 5 for f1 and f2, and each license would have 1 server line. The server nodes (unlike three-server redundancy) can be physically distant. The license files would look like:

- License 1 for chicago

```
SERVER chicago 17007ea8
VENDOR demo
FEATURE f1 demo 1.0 1-jan-1999 5 26C7DD9CD665
FEATURE f2 demo 1.0 1-jan-1999 5 0739D2F78CE4
```

- License 2 for tokyo

```
SERVER tokyo 17007ea8
VENDOR demo
FEATURE f1 demo 1.0 1-jan-1999 5 16BE40E1DAEE
FEATURE f2 demo 1.0 1-jan-1999 5 6DB6F3E40E61
```

The user in Chicago could set $LM_LICENSE_FILE to

```
@chicago:@tokyo
```

the user in Tokyo could set $LM_LICENSE_FILE to

```
@tokyo:@chicago
```

The application attempts the first server in the list, and if that fails for any reason, the second server is tried.

9.3 Comparing three-server to License File List

ARE THERE ANY DRAWBACKS TO USING THE LICENSE FILE LIST FOR REDUNDANCY?

Yes. By default, once a job (lc_new_job()) has successfully checked out a license from one host, all subsequent checkouts must be satisfied from the same host. If the application requires more than one FEATURE, then this could result in a license denial when the license may be available on another server. An application can bypass this restriction with the use of multiple FLEX*lm* “license jobs”.

If the application supports license queueing, all licenses are only queued from the first host on the list.

Finally, if one server becomes unavailable, some licenses will be unavailable.

WHEN IS IT RECOMMENDED TO USE A LICENSE FILE LIST FOR REDUNDANCY RATHER THAN TRUE REDUNDANT SERVERS?

When there’s less system administration time available to monitor license servers, and when the applications are not mission-critical. The license file list has some other advantages: it’s more forgiving if you lose quorum; it’s not limited to 3 servers (any number will work); and for wide-area networks, you can make servers available locally, with remote servers available as backup.

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 12, “Unix and VMS Platform-Specific Notes” on page 155 or Chapter 13, “Windows 95/98, and Windows NT” on page 161.
- FLEX*lm* makes use of, and depends on, the C standard I/O library and system calls. If you redefine any of the section 2 or section 3 calls, *you will get unpredictable results*.
- 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, 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.
- 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 4.22, “lc_heartbeat,” on page 49

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 2 common problems that occur in the field.

- “License server does not support this feature”

  This indicates that the client and servers are reading 2 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 error “encryption code in license file is inconsistent” (*LM_BADCODE*, -8) in a number of situations.

  In general, the *LM_BADCODE* error occurs when:

  —The license file has been mis-typed, or changed since it was created.

  —The encryption seeds in your application, vendor daemon, and license creation 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 insure 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 re-built 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.

  If you suspect that the license file has been generated incorrectly, here’s how you can tell:

  ```
  % lmcrypt -i file -f -o tmp
  % diff file tmp
  ```

lmcrypt should regenerate an identical file. If not, the file is bad, and the new file, in this example *tmp*, is good.

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 two solutions:

- The end-user can keep both license files separate, running one lmgrd with a license-file list of both files. There are compatibility issues that may arise with this method if some vendor daemons and/or applications are older than version 6.
- The end-user can keep both license files separate, running two lmgrds, one for each file. There are no drawbacks to this approach, since the lmgrd processes require few system resources.

Note: When using 2 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, UPGRADE, and FEATURESET 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 and Windows NT) 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

- Section 6.5, “Locating the License File,” on page 113

CASE 2

The products use different license server node(s).

In this case, separate license files will be required, one for each distinct set of license servers (where multiple software vendors use the same set of license server nodes, the technique described in case 1 above can be used to combine their license files).The resulting (multiple) 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 lfpath1:lfpath2:....:lfpathN
```

where: **is the:**

```
lfpath1 - is the path to the first license file
lfpath2 - is the path to the second license file.
.
.
lfpathN - is the path to the last (Nth) license file
```

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 rule is that the most recent (highest version number) lmgrd and utilities should be used.

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

Communications Transport

FLEX*lm* supports the UDP connectionless transport, in addition to the default TCP connection-based transport. Note that this does not apply to VMS systems. See Section 12.8.1, “Communications transport,” on page 157 for a discussion of VMS communications transport.

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.

11.1 How to Select UDP Connections

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

1. The application can call:

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

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

```
% setenv FLEXLM_COMM_TRANSPORT UDP
```

3. The comm transport can be selected in the daemon options file, with the line:

```
TRANSPORT UDP
```

or

```
TRANSPORT TCP
```

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

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

The environment variable and options file settings are 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. Options file specified — *COMM_TRANSPORT*
3. Environment variable specified — *FLEXLM_COMM_TRANSPORT*
4. *LM_A_ALLOW_SET_TRANSPORT* == false, Options file and Environment variable are disabled.
5. Set in application — *LM_A_COMM_TRANSPORT*
6. Vendor daemon is pre-v3.0— TCP only.

11.1.1 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, <# seconds>);
  ```

  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 10 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 10 seconds, while with a TCP client, this recognition is transparent.

In GLOBE*trotter* 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 4.30, "lc_set_attr," on page 56

- Section 4.22, “lc_heartbeat,” on page 49
- Section E.1, “LM_A_ALLOW_SET_TRANSPORT,” on page 205
- Section 5.3, “LM_A_CHECK_INTERVAL,” on page 77
- Section 6.7, “Special Hostids,” on page 118

Chapter 12

UNIX and VMS Platform-Specific Notes

12.1 Data General

On DG systems the daemon lockfile MUST be on a local filesystem (not NFS-mounted).

On DG-Intel, the hostid is an ethernet address. To work, the /dev/net/* files must all be readable. For this reason we recommend that, as root, the following command be executed:

```
# chmod 644 /dev/net/*
```

We do not consider this a security problem, since any PC attached anywhere on the net has access to all the same network traffic.

12.2 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 version 2.4, */dev/lan0* must have read and write permissions for everyone. Ethernet and FDDI are known to be supported devices, although earlier versions of HPUX had a bug with FDDI as hostid.

12.3 IBM

On RS/6000, lmgrd cannot be started in */etc/rc*; this is 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 this 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.

12.4 NCR

Redundant servers are not supported on NCR systems, due to a bug in the *accept()* networking call on these platforms.

The permissions on the */dev/lan0* device must be readable to obtain a hostid.

12.5 SGI

SGI has a variety of CPUs, operating systems and compiler switches which 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(?).
[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 3 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.*

12.6 SCO

Part if 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_KEYs (obtained from GLOBE*trotter* Software) and ENCRYPTION_SEEDs (2 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 daemon name. Then type *make* in the sco_u3 directory.

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

12.7 SVR4 Systems

All SVR4 systems, including Solaris 2.x, require linking with "-lsocket -lnsl". Solaris 2.x also requires -lintl. (OSF/1 does NOT require these link flags, although it is otherwise similar to SVR4).

12.8 VMS

Redundant server hosts are not supported on VMS.

FLEX*lm* client routines continue to use SIGALRM and *setitimer()*, as they do on the UNIX version. The FLEX*lm* implementation of *setitimer()* uses the *alarm()* call on VMS.

On VMS, the SYSNAM and SYSLCK privileges are required to run the daemons.

Vendor daemons support up to 255 clients each.

12.8.1 Communications transport

The VMS version of FLEX*lm* uses DECnet as a transport, rather than TCP/IP. This means that VMS servers cannot serve UNIX clients, and UNIX servers cannot serve VMS clients.

12.8.2 Special AST Considerations

If you let FLEX*lm* do its checking automatically, the check routine is implemented with signals (i.e., it runs at AST level). Therefore, if you use any of the following handler routines:

- *LM_A_USER_RECONNECT*
- *LM_A_RECONNECT_DONE*
- *LM_A_USER_EXITCALL*
- *LM_A_PERIODIC_CALL*

you should be careful to code them such that they can work at AST level. Note that you can call any of the FLEX*lm* client routines (routines that start with "lc_") from AST level.

If you use your own authentication routine (the "crypt" member of the setup struct), this routine must be coded to work properly from AST level.

12.8.3 DECnet Logical Links

Each FLEX*lm* client requires a pair of DECnet logical links. A typical default for the maximum # of logical links is 32, therefore, you will need to increase this parameter on the node that runs your vendor daemon(s). Do this with the following NCP command sequence.

```
$ mcr ncp
NCP> set exec links 100
```

To show how many logical links your system is configured for, do the following:

```
$ mcr ncp
NCP> show exec char
```

The result will be displayed as "MAXIMUM LINKS".

Setting the number of links to 100 should allow for 30 FLEX*lm* clients and a sufficient number of links for other uses. (If your current number of links is greater than 32, increase it accordingly).

12.8.4 VMS Ethernet Device Support

VMS supported ethernet devices

_xqa0:	DELQA, DEQNA, DESQA
_xea0:	DEUNA, DELUA
_esa0:	DESVA
_eta0:	DEBNA
_ewa0:	Alpha
_eza0:	uVax 4000
_exa0:	Vax 9000

If your customer's device is not one of the supported devices, they can "add support" by doing an assign. The standard table includes the logical device names "flexlm_hostid0", "flexlm_hostid1", "flexlm_hostid2", and "flexlm_hostid3". Assuming that your customer has ethernet device qqa0:, the following assign statement would add device qqa0:

```
$ assign _qqa0: flexlm_hostid0
```

This assignment would be done in the process that runs the daemon (for floating licenses) or the process that runs your application (for nodelocked licenses).

12.8.5 lmswitch

The lmswitch utility switches the debug log file for the daemon serving the specified feature while the daemon is running.

Usage is:

`lmswitch` *feature new-file*

where:	**is the:**
feature	any feature this daemon supports.
new-file	New file path.

Of course, for this syntax to work, lmswitch needs to be installed as a foreign command.

The new logfile will be opened for write, rather than append, so it is possible to "switch" to the same filename in order to be able to view the old (previous version) log file.

Chapter 13

Windows 95/98, and Windows NT

FLEX*lm* supports the Windows platforms using two sets of libraries 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.

13.1 Supported Compilers

The FLEX*lm* client library on Windows/NT is implemented as a DLL or a static library. The DLL can interface with almost any compiler. However, to build your vendor daemon on a Windows NT or Windows 95 system, FLEX*lm* supports only the Microsoft Visual C/C++ compiler 4.2 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 and Windows NT systems.

13.2 lc_new_job() and lc_free_job() Must Be Matched

When using the FLEXible API, *lc_new_job()* creates a license job, and *lc_free_job()* marks the end of the license job. FLEX*lm* applications must match *lc_free_job()* with *lc_new_job()* perfectly to avoid unexpected system problems on Windows. The reason for this requirement is the following: FLEX*lm* issues a Windows Socket *WSAStartup()* call in *lc_new_job()*, and *WSACleanup()* is issued by *lc_free_job()*. Failure to match *lc_new_job()* with *lc_free_job()* will cause unmatched *WSAStartup()* and *WSACleanup()*, which violates the WinSock specification and often results in an unstable system. This is true to a lesser degree in NT.

With the Trivial API, CHECKIN must match CHECKOUT; lp_checkin() calls must match lp_checkout().

13.3 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)
```

13.4 FLEX*lm* exit() Callback

The default operation of FLEX*lm* when the connection to the server is lost is to try 5 times and then exit the program.

13.5 Default License File

The default location for the license file on Windows and Windows NT is *C:\FLEXLM\LICENSE.DAT.*

13.6 Time Zone Setting

An optional part of FLEX*lm*'s client/server handshake procedure is to check the difference between their system time settings. The allowed difference in minutes is set by the client program using *lc_set_attr(LM_A_MAX_TIMEDIFF, x)*. On Windows, the FLEX*lm* client library checks the time zone environment variable, *TZ,* to adjust local time (set by the DOS TIME command) to Universal Coordinated Time (UCT, the same as GMT) in order to communicate with the FLEX*lm* server which may be anywhere in the world. On Windows NT systems, the user has to use the control panel to set the time and time zone correctly.

Use the following DOS syntax to set the *TZ* environment variable:

```
set TZ=tzn[+|-]hh[:mm[:ss]][dzn]
```

Note: This is only required if you call *lc_set_attr(LM_A_MAX_TIMEDIFF, *)*.

The *tzn* must be a three-letter time-zone name, such as PST, followed by an optionally signed number, hh, giving the difference in hours between UCT and local time. To specify the exact local time, the hours can be followed by minutes (:mm), seconds (:ss), and a three-letter daylight savings time zone, dzn, such as PDT. Separate hours, minutes, and seconds with colons (:). If daylight savings time is never in effect, as is the case in certain states and localities, set TZ without a value for dzn. If the TZ value is not currently set, the default is PST8PDT, which corresponds to the Pacific time zone.

For example, to set the TZ environment variable to correspond to the current time zone in Germany, you can use either one of the following forms:

```
set TZ=GST1GDT
set TZ=GST+1GDT
```

This uses the letters GST to indicate German Standard Time, though you can use any combination of three letters. This syntax assumes that Germany is one hour ahead of UCT, and that this timezone uses daylight savings time.

13.7 Node Lock and Hostid for Standalone PCs

FLEX*lm* for Windows and Windows NT also supports node locking. Several important issues should be considered when designing a node- lock licensing scheme for Windows and Windows NT systems. One or more of the following four options can be adopted to implement the node locking scheme that best fits your products.

1. Hostid implementation is essential to designing a node lock scheme, and the fact that PCs in general do not have a unique serial number is a problem. FLEX*lm* for Windows and Windows NT can use the volume serial number on the *C:* drive as the hostid for the PC. Your end-user may issue a DIR command on *C:* to get the serial number. The serial number is presented by DOS in hex form, and this number can be used in the hostid field of a license file. Please note that when a disk serial number is to be used as a hostid, you must use the format *DISK_SERIAL_NUM=xxxxxxxx* where x is a hexadecimal number.
2. There are two disadvantages to the volume serial number solution. First, disk serial number is only supported by DOS version 4.0 and later. Second, hard disk serial numbers can be changed by software with a single DOS file system IOCTL call. Therefore, FLEX*lm* offers a hardware solution using an external hardware key available from GLOBE*trotter* Software. When a license file has a hostid of the form *FLEXID=X-xxxxxxxx*, FLEX*lm* assumes that a hardware key solution for hostid is desired.
3. Another method of obtaining a hostid is to retrieve the Ethernet address as a hostid. When generating license files for this type of node lock scheme, simply use the Ethernet address without any prefix. Since there is not a generalized method of obtaining this value, special requirements apply.
4. If none of the above solutions is satisfactory, FLEX*lm* provides a set of callback routines for vendors to implement their private hostid. An application may use *lc_set_attr(LM_A_VENDOR_GETHOSTID, x*) to integrate a private hostid scheme into FLEX*lm*. The included sample program demonstrates how this can be done in *WINTEVW.CPP*.

SEE ALSO

- Section 6.7, “Special Hostids,” on page 118

13.8 System Requirements for obtaining Ethernet Address

There does not exist a uniform method of obtaining Ethernet address across platforms, so the requirements vary depending on the platform. The most common example of not configuring the system correctly is obtaining FFFFFFFF as the hostid.

13.8.1 Windows 95

Windows 95 requires the loading of either the NETBEUI Transport Protocol, or the NW Link (IPX/SPX) Transport Protocol to obtain the ethernet address.

13.9 Quick-Win/ 32 Bit Console Applications

On previous versions of FLEX*lm* it has been necessary to provide your own timers when running a 32 bit Console application. This is no longer necessary.

When running a Quick-Win application, the timers that are a part of Windows are not available. You will need to call *lc_timer()* on a periodic basis to ensure correct licensing operation.

13.10 Networking Requirements

Networking software is not required to use FLEX*lm* for nodelocked features. If you are running in a 32-bit mode, and you do not have WSOCK32.DLL in your path, and you are performing node-locked licensing, the correct operation will occur. If you are operating a laptop in portable mode, or have a dial up network connection that requires DNS, and you do not have DNS in your current mode of operation you can set the environment variable LM_NO_NETWORK to avoid lengthy DNS timeouts.

13.11 Hardware Hostids (Dongles)

13.11.1 General Information

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

13.11.2 FLEXID

This indicates a dongle hostid, and have a prefix like "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 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* is required to be installed.

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

13.12 Environment Variables (32-bit Platforms)

When running Windows applications, it is sometimes difficult to set environment variables. On NT, since 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. set command in autoexec.bat (windows), 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 a Service (on NT).

In FLEX*lm* v6 a new function was added to simplify this procedure. The function is lc_set_registry(job, char * EnvironVarName, char* EnvironValue);

This 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.

13.13 Special Syntax for Windows Version

When combining license files, use ‘;’ instead of ‘:’ for a list of license files in LM_LICENSE_FILE, e.g., file1;file2.

13.14 Minimum Files Required for Customer Installation

13.14.1 Client

- *lmgr327b.dll (Not necessary if using Statically linked library)*

13.14.2 Server

- *lmgr327b.dll (Not necessary if using Statically linked library)*
- *LMGRD.EXE*
- *XXXX.EXE (Your Vendor Daemon)*

13.15 Build Notes

The lmgr327b.dll library is built using the multi-threaded statically linked C Run-time library.

13.16 Linking to your Program

FLEX*lm* can be linked into your application in two ways. The two methods are using a static library, or using a DLL. To enhance security, it is recommended to use the static library if possible. The static library is compiled with Microsoft VC++ 4.2, 32 bit, with Multi-threading enabled, static C Runtime Library (/MT). The static library is named LMGR.LIB. GLOBE*trotter* Software also provides a library compiled with /MD, multi-threaded, using the C Runtime Library as a DLL, called LMGR_MD.LIB

Note there is no version information available in the static library at this time. You will need to use the date as your reference.

If it is necessary to use the DLL version of FLEX*lm*, please send email to support@globes.com, to get suggested enhancements to improve the security of your application. The DLL version is called LMGR327B.DLL with its associated import library LMGR327B.LIB.

13.17 If your application is a DLL

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:

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

where *dllname* is the name of your DLL.

13.18 Special Operating Conditions

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 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.

```
lc_cleanup();
```

13.19 Timers

Starting with FLEX*lm* v6.0, the default timers included in FLEX*lm* can be used for all applications, whether they are 32 bit console or WIN32 windows applications. You may still use your own timers by disabling the internal ones by calling:

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

13.20 Ethernet Address for WIN32 Platform

Over time changes have been made to improve the detection of ethernet addresses. It is necessary to load a Protocol stack other than TCP/IP to obtain the ethernet address because there is no mechanism in TCP/IP (WINSOCK) to obtain the ethernet address, nor is there any OS calls to obtain it. GLOBE*trotter* Software has spent considerable time with Microsoft to resolve this issue, but as of yet has not obtained the information necessary to perform this task without any additional protocol stacks.

First, if multiple ethernet cards are present, it will report all that it can detect. False Ethernet addresses (such as those generated from PPP stacks) have been filtered out. Do not issue a license for a ethernet address when the first four characters are “DEST” (44-45-53-54-00-00) and the rest are 0. Also certain cards will return a 0 or FFFFFFFF for a ethernet address, these are not valid addresses. Using the Nwlink (IPX/SPX) protocol to obtain the ethernet address will only result in one ethernet address being returned, instead of multiple ones if they are installed. This is due to problems inherent in the method of indirectly obtaining the ethernet address.

The process for detecting the ethernet address goes through the following series of steps. When it detects a valid address using the first method that succeeds, it stops any further processing and returns with that value.

First it checks if the LM_USE_SNMP environment variable/registry entry has been set. If so, it will attempt to obtain the ethernet address using SNMP protocol. This will only work if the SNMP service is installed on NT, and the service is running, and the customer has included the public trap in the Network SNMP control panel entries. SNMP on Windows 95 has not been tested.

If the LM_USE_SNMP has not been set, it checks for the ethernet address using the Netbios (NETBEUI) stack, unless the LM_NO_SNMP variable/registry value has been set.

Finally, if an ethernet address has not been detected up to this point, the code then checks for the ethernet address using the IPX/SPX (NWLINK) protocol stack. If this still doesn’t succeed, it will return a -1 which will print out as FFFFFFFF.

So to use the ethernet address as a hostid, you need to either load the SNMP service, the NETBIOS protocol, or the NWlink (IPX/SPX) protocol.

If you are having a problem with a specific protocol, and you have more than one loaded, you can set an environment variable to disable a particular attempt to get the ethernet address using a specific protocol.

Setting LM_USE_SNMP will enable using SNMP (If not set, FLEX*lm* will not use SNMP), setting LM_NO_NETBIOS will disable using Netbios for obtaining the ethernet address.

13.21 Operation on NEC NT machines

On certain models of NEC NT machines, the normal order of disk drives is reversed. In general on the NEC NT the boot hard-drive has the letter A as opposed to the normal C drive. When this is detected in FLEX*lm*, it automatically changes the default location of standard files such as the license file, vendor daemon lock file, etc. to drive A.

In previous versions, (4.0 and 5.0) the DISK VOLUME SERIAL NUMBER was always the one for disk C:. In later versions of FLEX*lm*, (v5.12 and v6.0)the following algorithm is used to get the DISK VOLUME SERIAL NUMBER: starting at drive A and going until drive C, the first fixed drive found that is not a CD drive is the drive for which the DISK VOLUME SERIAL NUMBER will be returned.

13.22 Networking

If FLEX*lm* is being used in a node-locked mode and node locking hostids which are not using networking functions are being used, then there are no networking software or hardware requirements. Instances of HOSTID's that do not require networking are the disk volume serial number, demo licenses, serial number licenses, or vendor defined hostid (if you are not using networking functions as a part of it).

If you are using floating (counted) licenses you will need to install TCP/IP and one of either NETBEUI, NWlink IPX/SPX, or SNMP.

13.23 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 Windows/NT, and this delay does not occur on other operating systems.

In FLEX*lm*, this is most notable when using 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 10 default ports, 27000-27009.

13.24 FLEX*lm* Utilities

In most of the other sections of the documentation, various FLEX*lm* utilities are referred to like lmstat, lmdiag, and lminstall. On the PC platforms, you need to either copy the lmutil.exe file to the name of the utility required, or just precede the lmstat, etc. with lmutil. For example:

```
lmutil lmstat -c mylicense.dat
```

On the WIN32 platforms, there is an additional program, LMTOOLS.EXE, which is a GUI front-end to lmutil with most of the same functionality.

13.25 Servers and Services

In order to use counted licenses, a license server must be running. This consists of a lmgrd running with the proper vendor daemon. These programs are WIN32 console applications, and can be run in a DOS window. They are started by lmgrd -app -c *license_file.*

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, it is possible to take advantage of a feature called Services. This is a interface that allows programs to be started and stopped through a common user interface, and run in the background. To get FLEX*lm* to run as a service, you need to "install" it. Two methods are available, the FLEX*lm* Control Panel, or a sample program, installs.c (located in the MACHIND directory).

13.26 Control Panel and Multiple Lmgrd's

Problems have been experienced using previous versions of FLEX*lm* and running multiple servers as services. On the NT operating system the preference is to run lmgrd as a service. Since there is a deficiency in allowing parameters to be passed to a service, it was necessary to set a global variable LM_LICENSE_FILE to pass the path to the license file. Since this was a single global, it was not possible to have multiple lmgrds run concurrently.

As of version 5.12 and continuing through 6.1, several changes have been made to enhance the operation of multiple services. These architectural changes will allow several version 5.12, and 6.1 FLEX*lm* lmgrds to run concurrently each inheriting a separate set of License files and other parameters.

First this information is stored in the registry in the following manner:

```
HKEY_LOCAL_MACHINE->SOFTWARE->FLEXlm License Manager->sname
LMGRD path-to-lmgrd.exe
License path-to-license
LMGRD_LOG_FILE path-to-log-file
```

where *sname* is the name of the service that shows up when you access the Services Control panel.

When the FLEX*lm* service starts up, it obtains its name, and then looks up the parameters in the registry. Lmgrd then uses this information to read the appropriate license file, launch the appropriate vendor daemon, and pass this information to the vendor daemon. When properly configured, all of this is invisible to the user.

The new version of install.c in version 6.1 (installs.c) will automatically write out these parameters in the registry, and also the new version of the FLEX*lm* Control Panel will similarly read and write these values.

13.27 WINDOWS 95 Multiple Servers

Windows 95 does not have the concept of Services as NT does. With version 5.12 and 6.1, GLOBE*trotter* Software has simulated partially the operation of services to allow FLEX*lm* to run multiple services, and not have them consume a console window on the display.

To do this we are saving the information in the registry as for NT, but when the computer powers up, we use the Microsoft\\Windows\\CurrentVersion\\RunServices function that is documented in the MSDN information. The FLEX*lm* Control panel when properly configured creates an entry in this registry location including a command line to be executed. This specifies the lmgrd95.exe program (which is assumed to be in the same directory as the lmgrd.exe that is going to be run) which inherits the values from the registry entries, which then specify the license file, log file, etc. This then starts the lmgrd in the background and starts the FLEX*lm* server.

As noted in the Microsoft documentation, after the server is started, if the user changes Users, or shuts down the system, the license manager will be stopped. This is why GLOBE*trotter* Software does not recommend using WIN95 for running license servers.

The installs.c program in the MACHIND directory has been updated to automatically support this feature when run on a Windows 95 machine. It automatically detects the operating system and then installs the Pseudo service.

13.28 Default Operation when Server connection is lost

The default operation of this version is now the same as UNIX. If you do not implement any reconnect callbacks, after trying to reconnect to the server for 5 times at 2 minute intervals, the program will exit. If this is not the desired operation of your product, either change the default parameters, register your own callbacks, or use the TRIVIAL or SIMPLE API and don’t use LM_RETRY_RESTRICTIVE.

13.29 Version Information

Starting in 5.12 and continuing in this version, in the lmgr327b.dll, GLOBE*trotter* Software has created a version resource. This will allow you to easily identify any current and future versions of the DLL. It is then possible to obtain the version by right clicking the mouse on the file and looking at the version tab. If your application uses the static library, lmgr.lib, run the lmutil lmver program to see which version of FLEX*lm* is bound to your executable.

13.30 Using languages other than “C”

There is a new API that is designed for non-C languages such as Visual Basic. It is designed to eliminate the usage of pointers. It is documented in the EXAMPLES/VB/VB4.0/VB.DOC file

13.31 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. The NOVELL server does not have any environment variables.

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 -> <environment_variable>

where <environment_variable> 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(job, char * env_var_name, char* environ_value);

This 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);
```

Chapter 14

Netware NLM

14.1 Introduction and Requirements

FLEX*lm* for Netware provides license server support on a Netware server which manages licensing via the IPX/SPX and TCP/IP protocol. This is accomplished by building a vendor-customized FLEX*lm* vendor daemon NLM. The client software development is the same as the normal FLEX*lm*, regardless of the transport protocol to be used.

FLEX*lm* for Netware NLM requires a WATCOM compiler version 10.0 or later and the Novell NLM SDK. Netware server version 3.x or 4.x is also required for running the vendor daemon NLM (Netware Loadable Module).

14.2 Installation

To install, you should use the same procedure as installing all other FLEX*lm* software and make sure to select the Novell Netware platform. Install your vendor daemon name, encryption seeds, and GLOBE*trotter*-provided vendor keys to include *lm_code.h* during the installation procedure. Build the vendor daemon by executing the batch file *flexlm_dir\i86_z3\build.bat.* When the batch file is completed, your FLEX*lm* vendor daemon DEMO.NLM is ready for use. You may customize your vendor daemon as described elsewhere. The customization procedure and mechanism is exactly the same as for FLEX*lm* on all other platforms. Copy the vendor daemon NLM to the Netware server SYS:\SYSTEM directory. Installation is now complete. Please refer to the next two sections on generating license files and starting the vendor daemon NLM.

14.3 Creating a License File For Netware

The create license programs MAKEKEY and LMCRYPT, which are included in FLEX*lm* for Win32, should be used to create the license file. To determine the hostname of the computer, type *CONFIG* on the console of the Netware server. Look for *File Server Name:* from the output. MAKEKEY queries for host id of the license server. For Netware servers, FLEX*lm* uses the Ethernet address of the first network interface card as its host id. To find the Ethernet address, type *CONFIG* on the console of your Netware server and look for *Node Address:* from the output. MAKEKEY also queries for the transport address. When using TCP as the transport protocol, simply enter the desired port number such as 7182 or TCP:7182. When using IPX/SPX as the

transport protocol, you must respond to the query with an SPX transport address, which consists of a socket number, a server node address, and an IPX network number. Enter the SPX transport address in the following format:

```
SPX:socket_number@node_address#network_number
```

For example,

```
1234@000000000001#2f14abf7.
```

The SPX socket number must be an unused decimal socket number on the server. If you use the FLEX*lm* vendor daemon NLM on a Netware server, you may use the following commands to find out the node address and network number of the Netware server: From a Netware client, execute the file *SYS:\system\netadmin.exe*. Make the following menu selection in sequence: *Manage Object -> View* or *Edit Property* of *Object -> View Server Information*. Look for *Node Address:* and *Network Address:* on screen. Please note that in many cases, the node address of your Netware server is 000000000001.

If you would like FLEX*lm* license server to support both IPX/SPX and TCP/IP clients concurrently, you must specify both transport addresses in the SERVER line of the license file. For example:

```
SERVER my_hostname my_hostid SPX:1234@000000000001#2f14abf7 TCP:7182
```

14.4 Starting the Vendor Daemon on a Netware Server

The FLEX*lm* license server on Netware is started with the following command at the Netware server console:

```
LOAD vendor_daemon_nlm [-l log_file_name] [-c license_file] [-x lmremove]
```

where:	**is:**
vendor_daemon_nlm	the file name of the vendor daemon NLM
log_file_name	the full path to the vendor daemon log file. The default log file is the system console.
license_file	the specified license file. Default is *SYS:\flexlm\license.dat*
-x lmremove	used to disable *lmremove* utility.

You may optionally enter your FLEX*lm* vendor daemon startup command in the Netware server *AUTOEXEC.CNF* file to auto-load the FLEX*lm* vendor daemon NLM during Netware server boot time.

14.5 Preparing the Client System to Use IPX/SPX for FLEX*lm*

In order to enable the FLEX*lm* client programs on Windows NT systems to use IPX/SPX, the following items should be checked:

The license file should specify the SPX transport address in the SERVER line as described in Section 14.3, “Creating a License File For Netware,” on page 173. You may manually change this field in the SERVER line of your license file. The Nwlink IPX/SPX Compatible Transport Driver must be properly installed and configured. You may use the Network icon from the Control Panel to verify this. FLEX*lm* supports Windows NT version 3.5 and later. You may use the *FLEXLM_COMM_TRANSPORT* environment variable to switch the FLEX*lm* transport protocol selection among TCP, UDP, and SPX. (You may use *lc_set_attr(lm_job, LM_A_COMM_TRANSPORT, LM_SPX)* to hard code your application to use IPX/SPX as the FLEX*lm* transport protocol. This usage is generally not encouraged since it limits end-user flexibility.)

In order to enable FLEX*lm* client programs on Windows systems to use IPX/SPX, the following items must be checked: The license file should specify the SPX transport address in the SERVER line as described in Section 14.3, “Creating a License File For Netware,” on page 173.

Netware client version 3.x or 4.x should be installed.

The following files must exist in the *WINDOWS* or *WINDOWS\SYSTEM* directory, or other directory in the Windows DLL search path: *NWCALLS.DLL NWIPXSPX.DLL TLI_SPX.DLl TLI_WIN.DLL* These files are part of the Netware client software.

14.6 Accessing License Servers via TCP/IP and SPX/IPX Concurrently

There may be situations at the end user sites where a Windows or NT system needs to access multiple license servers through TCP/IP and SPX/IPX concurrently. For example, you may be using an accounting package and a word processor whose license servers reside on a Unix and a Netware server respectively. In order to check out licenses for both products, you should follow the check list below in setting up the Windows or NT configuration: As a prerequisite, the Windows or NT system must first have TCP/IP and SPX/IPX protocol stacks configured to co-exist. Set the *LM_LICENSE_FILE* environment variable to the list of license files for your servers. For example: set *LM_LICENSE_FILE=license1.dat; license2.dat; license3.dat.* On Windows systems, this command must be done before Windows is started. On Windows NT systems, you may use the control panel to set environment variables. Please note that these license file are separated by “;”, and the *SERVER* line in each license file specifies a license server and the communication protocol in order to access the server.

14.6.1 Notes

Callbacks in the Netware NLM are defined as standard “C” functions. This is different than on the NT platform where the PASCAL calling conventions are used. Since the NLM is one module, no calls need to be made outside of the library, standard calling conventions can be used.

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 five standard APIs:

- The FLEX*lm* Trivial API
- The FLEX*lm* Simple API
- The FLEX*lm* FLEXible API
- Software License Working Group API
- LSAPI (a proposed standard)

FLEX*lm* is the only licensing system available which supports all five APIs.

A.1 The FLEX*lm* FLEXible API

The FLEX*lm* 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 FLEX*lm* API is to give you your choice of licensing models in an easy to implement, robust package. The FLEX*lm* API is documented in Chapter 4, "FLEXible API."

A.2 The FLEX*lm* Trivial and Simple APIs

These APIs are suitable for most applications, and are robust and easy to implement. See the FLEX*lm* Programmers Guide for complete information on these.

A.3 Software License Working Group

The Software License Working Group was a sub-group of the Network Computing Forum, organized by Apollo Computer in 1988. This group published a licensing system API which GLOBEtrotter believes to be the most usable of all the "standard" licensing APIs in existence to date. FLEX*lm* has supported this interface since 1990. Even though this interface is vendor-neutral and well-designed, few software companies utilize this interface, and no other "standards" organizations adopted it.

Notes:

- You cannot mix Software License Working Group calls with either the native FLEX*lm* calls or the LSAPI calls.
- FLEX*lm* on Windows and Windows NT systems do not support this API.

DATA TYPES FOR ALL CALLS

*vendor_id (char *)*	Same as vendor daemon name
*vendor_key (VENDORCODE *)*	Vendor's encryption seeds
*job_id (LM_HANDLE *)*	Job handle
length (int)	Length of a string
*text (char *)*	String
status (int)	Status return from call
*product_id (char *)*	Feature name
*version (char *)*	Version of feature
amount (int)	Number of licenses
queue (int)	Enqueue request flag
check_period (int)	check_interval
*license_handle (char *)*	Handle for this license
queue_position (int)	Our position in the queue

GENERAL CALLS

ls_init (vendor_id, vendor_key, &job_id):	Same as *lc_init()*
ls_terminate (job_id):	Terminate licensing operations
ls_log_message (job_id, length, text, &status):	Similar to *lc_log()*

LICENSE BROKERING CALLS

lb_request (job_id, product_id, version, amount, queue, check_period, &license_handle)

Check out a license

lb_check_wait (license_handle, check_period, &queue_position)

Check queue position for a license

lb_wait_remove (license_handle)

Remove process from queue for this license

lb_confirm (license_handle, check_period)

Inform daemon that license is still in use

lb_release (license_handle)

Release a license (*lc_checkin()*)

lb_get_cur_users (job_id, product_id, version, next, maximum_entries, queued, total_licenses, number_of_entries, entries)

Get a list of users of "product_id" (like lc_userlist, only much harder to use).

A.4 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* API. 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* API 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 either the native FLEX*lm* calls or the Software License Working Group calls.

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.5 LSAPI General Calls

LIST PROVIDERS OF LICENSING SERVICE

*LS_STATUS_CODE LSEnumProviders (LS_ULONG Index, LS_STR *Buffer)*

GET MESSAGE TEXT FROM LICENSING SYSTEM

LS_STATUS_CODE LSGetMessage(LS_HANDLE Handle
LS_STATUS_CODE Value,
*LS_STR *Buffer, LS_ULONG BufferSize)*

QUERY LICENSE INFORMATION

LS_STATUS_CODE LSQuery(LS_HANDLE Handle, LS_ULONG Information,
*LS_VOID *InfoBuffer, LS_ULONG BufferSize,*
*LS_ULONG *ActualBufferSize)*

RELEASE LICENSE

LS_STATUS_CODE LSRelease(LS_HANDLE Handle, LS_ULONG
*TotUnitsConsumed, LS_STR *LogComment)*

REQUEST LICENSE

*LS_STATUS_CODE LS*Request(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)

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, ...);
```

UPDATE LICENSE STATUS

LS_STATUS_CODE LSUpdate(LS_HANDLE Handle,
LS_ULONG TotUnitsConsumed,
LS_ULONG TotUnitsReserved,
*LS_STR *LogComment,*
*LS_CHALLENGE *lpChallenge,*
*LS_ULONG *TotUnitsGranted)*

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

The Debug Log File

FLEX*lm* daemons all generate log files in the following format.

```
mm/dd hh:mm (VENDOR NAME) message
```

where:	is the:
mm/dd hh:mm	The time that the message was logged.
VENDOR NAME	Either lmgrd or the daemon name. In the case where a single copy of the daemon cannot handle all of the concurrent clients, an optional "_" followed by a number indicates that this message comes from a child daemon process.
message	The text of the message.

The log files can be used to:

- Diagnose configuration problems.
- Diagnose daemon software errors.

The content of these messages is described below. If you have a source kit, these messages are also found by searching for the "LOG" macros in the.*c* files in the *server*, *app*, and *master* directories.

Note: The debug log file is NOT suitable for usage reporting.

B.1 Informational Messages

Connected to node	This daemon is connected to its peer on "node".
CONNECTED, master is name	The license daemons log this message when a quorum is up and everyone has selected a master.
DEMO mode supports only one SERVER host!	An attempt was made to configure a demo version of the software for more than one server host.
DENIED: N feature to user (mm/dd/yy hh:mm)	"user" was denied access to "N" licenses of "feature".

EXITING DUE TO SIGNAL nnn	EXITING with code nnn. All daemons list the reason that the daemon has exited.
EXPIRED: feature	“feature” has passed its expiration date.
IN: feature by user (N licenses)	“user” has checked back in “N” licenses of “feature” at mm/dd/yy hh:mm.
IN server died: feature by user (number licenses)	“user” has checked in “N” licenses by virtue of the fact that his server exited.
License Manager server started	The license daemon was started.
Lost connection to host	A daemon can no longer communicate with its peer on node “host”, which can cause the clients to have to reconnect, or cause the number of daemons to go below the minimum number, in which case clients may start losing their licenses. If the license daemons lose the connection to the master, they will kill all the vendor daemons; vendor daemons will shut themselves down.
Lost quorum	The daemon lost quorum, so will process only connection requests from other deamons.
MASTER SERVER died due to signal nnn	The license daemon received fatal signal nnn.
MULTIPLE xxx servers running. Please kill, and restart license daemon	The license daemon has detected that multiple copies of vendor daemon “xxx” are running. The user should kill all “xxx” daemon processes and re-start the license daemon.
OUT: feature by user (N licenses)	“user” has checked out “N” licenses of “feature”
Removing clients of children	The top-level daemon logs this message when one of the child daemons exits.
RESERVE feature for HOST name RESERVE feature for USER name	A license of “feature” is reserved for either user “name” or host “name”.
REStarted xxx (internet port nnn)	Vendor daemon “xxx” was restarted at internet port “nnn”.

Retrying socket bind (address in use)
: The license servers try to bind their sockets for approximately 6 minutes if they detect “address in use” errors.

Selected (EXISTING) master node.
: This license daemon has selected an existing master (node) as the master.

SERVER shutdown requested.
: A daemon was requested to shut down via a user-generated kill command.

[NEW] Server started for: feature-list
: A (possibly new) server was started for the features listed.

Shutting down xxx
: The license daemon is shutting down the vendor daemon xxx.

SIGCHLD received. Killing child servers
: A vendor daemon logs this message when a shutdown was requested by the license daemon.

Started name
: The license daemon logs this message whenever it starts a new vendor daemon.

Trying connection to node
: The daemon is attempting a connection to “node”.

B.2 Configuration Problems

hostname: Not a valid server host, exiting
: This daemon was run on an invalid hostname.

hostname: Wrong hostid, exiting
: The hostid is wrong for “hostname”.

BAD CODE for feature-name
: The specified feature name has a bad license key.

CANNOT OPEN options file “file”
: The options file specified in the license file could not be opened.

Couldn’t find a master
: The daemons could not agree on a master.

license daemon: lost all connections
: This message is logged when all the connections to a server are lost, which often indicates a network problem.

lost lock, exiting

Error closing lock file

Unable to re-open lock file	The vendor daemon has a problem with its lock file, usually because of an attempt to run more than one copy of the daemon on a single node. Locate the other daemon that is running via a “ps” command, and kill it with “kill -9”.
NO DAEMON line for *daemon*	The license file does not contain a “VENDOR” line for “*daemon*”
No “FLEX*lm*” service found	The TCP “FLEX*lm*” service did not exist in /etc/services.
No features to serve!	A vendor daemon found no features to serve. This could be caused by bad data in the license file.
UNSUPPORTED FEATURE request: feature by user	The “user” has requested a feature that this vendor daemon does not support. This can happen for a number of reasons: the license file is bad, the feature has expired, or the daemon is accessing the wrong license file.
Unknown host: hostname	The hostname specified on a “SERVER” line in the license file does not exist in the network database (probably /etc/hosts).
lm_server: lost all connections	This message is logged when all the connections to a server are lost. This probably indicates a network problem.
NO DAEMON lines, exiting	The license daemon logs this message if there are no VENDOR lines in the license file. Since there are no vendor daemons to start, there is nothing to do.
NO DAEMON line for name	A vendor daemon logs this error if it cannot find its own VENDOR name in the license file.

B.3 Daemon Software Errors

accept: message	An error was detected in the “accept” system call.
ATTEMPT TO START VENDOR DAEMON xxx with NO MASTER	A vendor daemon was started with no master selected. This is an internal consistency error in the daemons.

BAD PID message from nnn: pid: xxx (msg)
: A top-level vendor daemon received an invalid PID message from one of its children (daemon number xxx).

BAD SCONNECT message: (message)
: An invalid “server connect” message was received.

Cannot create pipes for server communication
: The “pipe” call failed.

Can’t allocate server table space
: A malloc error. Check swap space.

Connection to node TIMED OUT
: The daemon could not connect to “node”

Error sending PID to master server
: The vendor daemon could not send its PID to the top-level server in the hierarchy.

f_do_notify called with no valid feature
: This is an internal consistency error.

Illegal connection request to DAEMON
: A connection request was made to “DAEMON”, but this vendor daemon is not “DAEMON”

Illegal server connection request
: A connection request came in from another server without a DAEMON name.

KILL of child failed, errno = nnn
: A daemon could not kill its child.

No internet port number specified
: A vendor daemon was started without an internet port.

Not enough descriptors to re-create pipes
: The “top-level” daemon detected that one of its sub-daemons exited. In trying to restart the chain of sub-daemons, it was unable to get the file descriptors to set up the pipes to communicate. This is a fatal error, and the daemons must be re-started.

read: error message
: An error in a “read” system call was detected.

recycle_control BUT WE DIDN’T HAVE CONTROL
: The hierarchy of vendor daemons has become confused over who holds the control token. This is an internal error.

return_reserved: can’t find feature listhead
: When a daemon is returning a reservation to the “free reservation” list, it could not find the listhead of features.

select: message
: An error in a select system call was detected.

Server exiting
: The server is exiting. This is normally due to an error.

SHELLO for wrong DAEMON
: This vendor daemon was sent a “server hello” message that was destined for a different “DAEMON”.

Unsolicited msg from parent!
: Normally, the top-level vendor daemon sends no unsolicited messages. If one arrives, this message is logged. This is a bug.

WARNING: CORRUPTED options list (o->next == 0)

Options list TERMINATED at bad entry
: An internal inconsistency was detected in the daemon’s option list.

Appendix C

FLEX*lm* Status Return Values

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. FLEX*lm* will attempt to open the standard file */usr/local/flexlm/licenses/license.dat*,(or *C:\FLEXLM\LICENSE.DAT* on Windows and Windows NT) or the file specified by the vendor (via *lc_set_attr()*), or the file specified by the user in *LM_LICENSE_FILE*.
`-2 LM_BADFILE`	`"invalid license file syntax"`
	A feature name is > MAX_FEATURE_LEN,
	A daemon name is > MAX_DAEMON_LEN.
	A server name is > MAX_SERVER_NAME.
	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 licenses 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 nodename 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 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 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, -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 GLOBE*trotter* 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, `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, `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 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 6.3.4, "FEATURE or INCREMENT Line," on page 100, 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 ormisspelling 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 2 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 hostname. Replace this_host with a real hostname 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 the section on Decimal Format 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, since 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 www.globetrotter.com or from your software vendor.

-111 LM_NORESLINK "lmgr.res, Windows Resource file, not linked"

When linking Windows binaries, you must link with lmgr.lib as well as lmgr.res

-112 LM_NODONGLEDRIVER“Missing Dongle Driver”

In order to read the dongle hostid, the correct driver must be installed. These drivers are available at www.globetrotter.com or from your software vendor.

-113 LM_FLEXLOCK2CKOUT“2 FLEXlock checkouts attempted”

Only 1 checkout is allowed with FLEXlock enabled apps.Subsequent checkout attempts will fail. They should be disabled if first checkout succeeded in FLEXlock mode.

Appendix D

FLEX*lm* Limits

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

D.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 Names	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
Latest start date	license key: 31-dec-2027 START=: 31-dec-9999
Number of FEATUREs	Unlimited (Same for INCREMENT/UPGRADE)
Number of VENDORs	Unlimited
Number of SERVERs	3 (Redundant server licenses are limited to 3 servers)
OVERDRAFT	32-bit integer
USER_BASED=*n*	32-bit integer (Same for HOST_BASED)
MINIMUM=*n*	32-bit integer
Other optional FEATURE attributes	Limited only by the total length of the FEATURE line.

Decimal Format

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

Convertible Licences
: Everything but PACKAGE and FEATURESET lines.

Convertible FEATURE names
: Only officially supported FEATURE names. In particular, '-' (hyphen) cannot be converted.

D.2 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.

D.3 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)

D.4 Other API limits

Vendor-defined hostid length
: 40 - including the "NAME=" prefix.

Vendor-defined FEATURE license key length
: 20

Number of licenses in one *lc_checkout()* request
: 9999

Long error message length
: 1024 characters (length of string returned from *lc_errstring()*)

D.5 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.

While file descriptors are exhausted, additional vendor daemons are spawned to support the extra file descriptors. the maximum number of spawned daemons is 255, which effectively limits the maximum number of clients to 256*25 -(2* # servers -1) or approximately 6,400 per vendor daemon.

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.

D.6 lmgrd

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

D.7 Sub-net, Domains, Wide-Area Networks

FLEX*lm* has no limitations regarding subnets (since FLEX*lm* does not use *broadcast* messages).

If the hostname in the license file is fully-qualified (*name.domain.suf*) or is an IP-Address (*nnn.nnn.nnn.nnn*), then there are no limitations with regard to internet domains.

There are no other limitations regarding wide-area networks.

D.8 LM_LICENSE_FILE

Number of licenses in path	Unlimited

Appendix E

Additional lc_set_attr() Attributes

The attributes listed in this section should be used with caution, as they are rarely needed, and can cause problems if inappropriately used.

E.1 LM_A_ALLOW_SET_TRANSPORT

Type: int

This option allows you to disable the end-user selection of communications transport, which they can do with either the daemon options file, or the *FLEXLM_COMM_TRANSPORT* environment variable.

Default: On(1), i.e., end-users can select communications transport.

SEE ALSO

- Section Chapter 11, “Communications Transport,” on page 151

E.2 LM_A_ALT_ENCRYPTION

Type: (VENDORCODE *)

This option allows a vendor to more easily migrate to a new set of encryption seeds while still supporting old license files in the field. This option would be used in new applications until all the old license files in the field were replaced.

Here’s an example of the use of *ALT_ENCRYPTION*:

Your company has 3 products, which are widely available, and you want eventually to make the old license files obsolete. You could do this by generating new license files with a new set of encryption seeds (the two hexadecimal numbers in lm_code.h that your company defines), but you might want to phase this in, since the 3 products will not all be re-released with the new encryption seeds for a year and you want the old versions to continue to work.

The solution is to make the new products support both old and new license files, shipping with new vendor daemons that will accept both old and new license files. At this point, license files would be shipped generated with the new encryption seeds.

Once all three new products become available, and all new license files are generated with the new encryption seeds; later versions of the products and vendor daemons can be released that will only accept the new encryption seeds.

Here is an excerpt from the interim client code:

```
lm_code.h:
        /* newcode */
        LM_CODE(newcode, ENCRYPTION_SEED1,
                        ENCRYPTION_SEED2, VENDOR_KEY1, VENDOR_KEY2,
                          VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
        /* oldcode */
        LM_CODE(oldcode, OLD_ENCRYPTION_SEED1,
                         OLD_ENCRYPTION_SEED2, VENDOR_KEY1, VENDOR_KEY2,
                         VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
application.c:
        rc = lc_new_job((LM_HANDLE *)0, VENDOR_NAME, &newcode, &lm_job);
        lc_set_attr(lm_job, LM_A_ALT_ENCRYPTION, &oldcode);

/* lc_checkout will try both newcode and oldcode */
        rc = lc_checkout(lm_job, feature, version, checkout_cnt,
                        LM_CO_NOWAIT, &newcode, LM_DUP_NONE);
```

Also, in the *lsvendor.c* file, the first version of the vendor daemon would need to support both encryption seeds, as follows:

```
        /*
         * Read license files using either vendorkeys and
         * accept connections from clients using either
         * vendorkeys
         */
        VENDORCODE vendorkeys[] = {
                  {VENDORCODE_4, ENCRYPTION_SEED1, ENCRYPTION_SEED2,
                       VENDOR_KEY1, VENDOR_KEY2, VENDOR_KEY3,
                       VENDOR_KEY4,FLEXLM_VERSION, FLEXLM_REVISION},
                 {VENDORCODE_4, OLD_ENCRYPTION_SEED1,
                      OLD_ENCRYPTION_SEED2, VENDOR_KEY1, VENDOR_KEY2,
                      VENDOR_KEY3, VENDOR_KEY4, FLEXLM_VERSION,
                      FLEXLM_REVISION}};
```

The final version of the vendor daemon, (the one that would no longer support old license files) would have the second vendor key (the one with OLD_ENCRYPT_CODE_*n*) removed, and the next release of the applications would have the *lc_set_attr(LM_A_ALT_ENCRYPTION,...)* call removed.

Default: None.

E.3 LM_A_COMM_TRANSPORT

Type: short

Communications transport to use, one of *LM_TCP* or *LM_UDP*. It is recommended that this be set by the end-user (license administrator), not set in the application.

Default: *LM_TCP*.

Note: Not supported on VMS.

SEE ALSO

- Section Chapter 11, “Communications Transport,” on page 151

E.4 LM_A_CONN_TIMEOUT

Type: int

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()*.

Default: 10 seconds.

E.5 LM_A_CRYPT_CASE_SENSITIVE

Type: short

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.

Default: Case-insensitive comparison.

E.6 LM_A_DIAGS_ENABLED

Type: short

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. GLOBE*trotter* 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.

Default: On (1).

E.7 LM_A_DISABLE_ENV

Type: short

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.

Default: *LM_LICENSE_FILE* environment variable enabled.

SEE ALSO

- Section E.10, "LM_A_LICENSE_FILE and LM_A_LICENSE_FILE_PTR," on page 210
- Section 6.5, "Locating the License File," on page 113

E.8 LM_A_EF_1, LM_A_EF_2, LM_A_EF_3, LM_A_EF_4, LM_A_EF_5

Type: int

These five event flags are used on VMS only. FLEX*lm* client routines assign these event flags dynamically. You can reassign the event flags used by FLEX*lm* by calling *lc_set_attr()* for the attributes *LM_A_EF_1*, *LM_A_EF_2*, *LM_A_EF_3*, *LM_A_EF_4*, or *LM_A_EF_5*. The only restriction is that flag #1 and flag #2 must be in the same event flag cluster. If you reset them, do it before calling any FLEX*lm* routines. You can use *lib$get_ef* and *lib$free_ef* to assign event flag numbers, if desired.

Default*:* Event flags assigned with *lib$get_ef()*

E.9 LM_A_ETHERNET_BOARDS

Type: (char **)

This table of ethernet devices, if specified, will be searched before the standard FLEX*lm* ethernet device table is searched. This can be used either to override the FLEX*lm* device order, or to add new devices that are not built into the FLEX*lm* table. This table is used for both ULTRIX and VMS systems but not for Windows and Windows NT systems. The table should be specified as a NULL-terminated array of character string pointers, as in the following example (this example is the standard ULTRIX table):

```
char *our_ethernet_boards[] =
        "qe0", "de0", "se0", "ni0", "ln0", "xna0", "ne0", 0};
lc_set_attr(LM_A_ETHERNET_BOARDS, our_ethernet_boards);
```

For VMS systems, be sure to include each interface in the table four times as xxa0:, xxa1:, xxa2:, and xxa3: (for device xx). The following example is the FLEX*lm* standard ethernet device table for VMS:

```
{"flexlm_hostid0", "flexlm_hostid1", "flexlm_hostid2", "flexlm_hostid3",
"_xqa0:", "_xqa1:", "_xqa2:", "_xqa3:",  /* DELQA, DEQNA, DESQA */
"_xea0:", "_xea1:", "_xea2:", "_xea3:",  /* DEUNA, DELUA */
"_esa0:", "_esa1:", "_esa2:", "_esa3:",  /* DESVA */
"_eta0:", "_eta1:", "_eta2:", "_eta3:",  /* DEBNA */
"_eza0:", "_eza1:", "_eza2:", "_eza3:",  /* uVAX 4000 */
"_exa0:", "_exa1:", "_exa2:", "_exa3:",  /* VAX 9000 */
"_ewa0:", "_ewa1:", "_ewa2:", "_ewa3:",  /* Alpha */
0 };
```

In order to update your daemon for a new set of ethernet addresses, you will need to put this call to *lc_set_attr()* into an initialization routine and set *ls_user_init1* to the name of your initialization routine, as in the following example:

In any source module:

```
void daemon_init()
{
        static char *new_board[] = {"new_board", 0};

                lc_set_attr(LM_A_ETHERNET_BOARDS, new_board);
}
```

In *lsvendor.c*:

```
        extern void daemon_init();
        void (*ls_user_init1)() = daemon_init;
```

Then, modify the makefile for the daemon to include your module with *daemon_init()*, or put this module into the *liblmgr_as.a* archive. On ULTRIX systems, *CONFIG_DAEMON* will automatically make the edit to *lsvendor.c* if you answer "daemon_init" to the daemon initialization (START) question.

Default: Only standard FLEX*lm* table searched.

E.10 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 5.15, "LM_A_LICENSE_DEFAULT," on page 82
- Section E.7, "LM_A_DISABLE_ENV," on page 208
- Section 6.5, "Locating the License File," on page 113

E.11 LM_A_MAX_TIMEDIFF

Obsolete. This check is now automatically performed when needed.

E.12 LM_A_NO_TRAFFIC_ENCRYPT

Type: short

This field allows you to defeat the authentication of traffic that happens between client and server programs. If you set this parameter to a non-zero value, client-server communications will be unencrypted, as they were in V1.x FLEX*lm*. DO NOT CHANGE THIS VALUE WITHOUT CONTACTING GLOBE*trotter* Software support.

Default: 0, Traffic is encrypted.

E.13 LM_A_PERIODIC_CALL

Type: Pointer to a function returning int. Return value not used.

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*.

Default: No periodic call.

SEE ALSO

- Section 5.3, "LM_A_CHECK_INTERVAL," on page 77
- Section 4.22, "lc_heartbeat," on page 49

E.14 LM_A_PERIODIC_COUNT

Type: int

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*.

Default: 0 (No *PERIODIC_CALL*).

SEE ALSO

- Section 5.3, “LM_A_CHECK_INTERVAL,” on page 77
- Section 4.22, “lc_heartbeat,” on page 49

E.15 LM_A_USE_START_DATE

Type: short

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.

Default: 1, Use start date.

E.16 LM_A_USER_CRYPT

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

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

Default: FLEX*lm* standard authentication routine.

The *crypt* routine is called as follows:

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

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

(VENDORCODE *) *key*	Pointer to the 1st 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 4.10, "lc_cryptstr," on page 39
- Section 8.3.28, "ls_user_crypt," on page 139

Symbols
/dev/lan0 156
@host
    contacting license server 115
A
AIX 155
ANY
    lc_gethostid 67
API
    Industry standards 177
B
BINARY_KIT 18
C
checkin callback
    in vendor daemon 137
checkin filter
    in vendor daemon 137
checkout filter
    in vendor daemon 138
chroot
    and ls_do_checkroot 136
client
    definition 14
COMM_TRANSPORT
    UDP vs TCP 152
Communications Transport (TCP vs. UDP) 151
COMPONENTS
    PACKAGE line syntax 108
COMPONENTS= 108
CONFIG 61, 211
    lc_auth_data 29
CONFIG_DAEMON 141
configuring FLEXlm
    number of server nodes 143
D
DAEMON

line syntax 98
daemon
definition 14
daemon death
automatic reconnection 87
detecting 77
Data General 155
Debug Log File
syntax 183
debug log file 130
Debugging hints 147
decimal format
lc_convert 38
Decimal Format Licenses 111
Decimal format licenses
limits 201
DEMO
Hostid on FEATURE line 102
lc_getid_type 67
DISK_SERIAL_NUM 118
DISPLAY
dup_group 33
DisplayString 79
DLL 161
Domains
limits 203
Dongles
windows 164
DUP_GROUP
and lingering licenses 21
Duplicate Grouping Mask 35
E
encryption routine
user 211
ENCRYPTION_SEED 140
END_LICENSE 115

End-User Options File
    limits 202
Ethernet Address
    Windows 164
examples 19
exinstal.c
    lc_check_key 31
    lc_convert 38
expiration date
    FEATURE line 101
    limits 201
F
FEATURE
    limits 201
    line syntax 100
feature
    defintion 13
feature name
    FEATURE line 101
FEATURE Name length 201
FEATURESET 60, 64, 65, 138, 149
FLEXible API 13
    example application 19
    function listing 27
    overview 15
FLEXID 119
FLEXlm API 177
FLEXlm.po 20
FLEXLM_COMM_TRANSPORT 152, 205
FLEXLM_DIAGNOSTICS
    LM_A_DIAGS_ENABLED 207
FORTRAN
    and SIGALRM 85
G
gethostname
    LM_A_DISPLAY_OVERRIDE 79

LM_A_HOST_OVERRIDE 81
H
Heartbeats
lc_timer 50
overview 20
Hewlett Packard 155
HOST
dup_group 33
Host ID
determining with *lmhostid* 116
on different machine types 116
HOSTID
lc_gethostid 66
LM_A_VENDOR_ID_DECLARE 90
hostid 66
lc_gethostid 66
LSAPI 179
HOSTID=
FEATURE attribute 102
HOSTID_DISK_SERIAL_NUM 67
HOSTID_DISPLAY 67
HOSTID_ETHER 66
HOSTID_FLEXID1_KEY 67
HOSTID_FLEXID2_KEY 67
HOSTID_FLEXID3_KEY 67
HOSTID_FLEXID4_KEY 67
HOSTID_HOSTNAME 67
HOSTID_INTEL 121
HOSTID_INTEL32 67
HOSTID_INTEL64 67
HOSTID_INTEL96 67
HOSTID_INTERNET 67
HOSTID_LONG 66
HOSTID_SERNUM_ID 67
HOSTID_STRING 67
HOSTID_UNAME_LONG 67

HOSTID_USER 66
HOSTID_VENDOR 67
HOSTTYPE 68
I
IBM 155
ID= 67
    FEATURE attribute 104
    Hostid type 119
ID_STRING
    Hostid type 119
INCREMENT
    line syntax 100
    ls_use_all_feature_lines 139
Installation
    Netware 173
Intel Pentium III+ Hostid 121
J
job
    lc_new_job 53
L
l_extract_date
    lc_crypt 62
l_n36_buf 53
l_new_hostid 29
LANG
    Internationalization for Solaris 20
lc_auth_data 29
lc_baddate 60
lc_check_key 30
    lc_convert 38
lc_checkin 31
    and lingering licenses 20
    LM_A_CHECKOUT_DATA—DUP_VENDOR 77
    LM_A_UDP_TIMEOUT 86
lc_checkout 31, 77
    LM_A_CHECKOUT_FILTER 78

lc_chk_conf 37
lc_ck_feats 60
lc_convert 37
LC_CONVERT_TO_DECIMAL 38
LC_CONVERT_TO_READABLE 38
lc_copy_hostid 61
lc_crypt 61
lc_cryptstr 39
lc_disconn 63
lc_display 63
lc_errstring 41, 42, 43
    lc_perror 55
lc_expire_days 43
lc_feat_list 44
lc_feat_set 64
lc_first_job 45
lc_free_hostid 45
lc_free_hostid()
    l_new_hostid 29
lc_free_job
    Windows 161
lc_free_mem 46
    lc_convert 38
    lc_cryptstr 40
lc_get_attr 47
    lc_set_attr 76
    multiple jobs 23
lc_get_config 48
lc_get_errno 64
lc_get_feats 65
lc_gethostid 65
    lc_free_hostid 45
lc_getid_type
    lc_free_hostid 45
lc_heartbeat 49
lc_hostid 50

lc_hostname 63
lc_hosttype 67
lc_idle 51
lc_init 52
    windows 161
lc_isadmin 68
lc_lic_where 69
lc_log 53
lc_master_list 69
lc_new_job 53
    lc_free_job 46
    multiple jobs 21
lc_next_conf 55
lc_next_job 45
lc_perror 55
    LM_A_USER_EXITCALL 86
lc_remove 70
lc_set_attr 56
    and lingering licenses 20
    building your application 28
    limits 202
    listing of attributes 75
lc_shutdown 71
lc_status 72
lc_timer 72
    LM_A_PERIODIC_COUNT 211
    LM_A_USER_EXITCALL 87
    LM_A_USER_RECONNECT 87
lc_timer()
    debugging 147
lc_userlist 58
    LM_A_CHECKOUT_DATA—DUP_VENDOR 78
lc_username 63
lc_vsend 59
    ls_vendor_msg 139
Lenient Licensing

REPORTLOG and OVERDRAFT 94
liblmgr.a
building your application 28
license
definition 14
license administrator 68
license daemon
automatic reconnection 87
definition 14
detecting daemon death 77
license file
combining license files from multiple vendors 149
definition 14
example 110
location 113
VMS default location 134
License File Limits 201
license file list
definition 14
License key
FEATURESET 110
LM_BADCODE 35
license key
definition 14
FEATURE 101
lc_crypt 61
lc_cryptstr 39
length and start-date 123
license file comments 110
license file format 97
LM_A_LKEY_LONG 83
LM_A_USE_START_DATE 211
LM_A_USER_CRYPT 211
LM_BADCODE error 185
ls_crypt_case_sensitive 135
port@host 116

SUPERSEDE 105
license server
definition 14
license-key
lc_check_key 30
limits 202
LM_LGEN_VER 82
start-date limits 201
LINGER 20
LM_A_LINGER 83
Linger
overview 20
LM_A_ALLOW_SET_TRANSPORT 151, 152
LM_A_ALT_ENCRYPTION 205
LM_A_CHECK_INTERVAL 210, 211
lc_timer 73
limits 202
LM_A_USER_RECONNECT 87
LM_A_CHECKOUT_DATA
limits 202
LM_A_CHECKOUT_DATA—DUP_VENDOR 77
LM_A_CHECKOUTFILTER 78
use with VENDOR_STRING 106
LM_A_COMM_TRANSPORT 152, 207
LM_A_CONN_TIMEOUT 207
LM_A_CRYPT_CASE_SENSITIVE 101, 207
LM_A_DIAGS_ENABLED 207
LM_A_DISPLAY_OVERRIDE 79
limits 202
LM_A_EF_1 208
LM_A_ETHERNET_BOARDS 209
LM_A_FLEXLOCK 80
LM_A_FLEXLOCK_INSTALL_ID 80
LM_A_HOST_OVERRIDE 80, 81
LM_A_HOSTNAME_OVERRIDE
limits 202

LM_A_LF_LIST 81
LM_A_LICENSE_CASE_SENSITIVE 82
LM_A_LICENSE_DEFAULT 82
LM_A_LICENSE_FMT_VER 82
    lc_convert 38
    lc_cryptstr 39
LM_A_LINGER 21, 82
LM_A_LKEY_LONG 83
LM_A_NO_TRAFFIC_ENCRYPT 210
LM_A_PC_PROMPT_FOR_FILE 84
LM_A_PERIODIC_CALL 210
LM_A_PERIODIC_COUNT 210, 211
LM_A_RECONNECT_DONE
    lc_timer 50
LM_A_RETRY_CHECKOUT 84
LM_A_RETRY_COUNT 85
    lc_timer 50
LM_A_RETRY_INTERVAL 85
    lc_timer 73
LM_A_REVISION 91
LM_A_SETITIMER 85
LM_A_SIGNAL 85, 86
LM_A_TCP_TIMEOUT 86
    lc_timer 50, 73
LM_A_UDP_TIMEOUT 86, 152
    lc_checkin 31
    lc_idle 52
    lc_timer 50, 73
LM_A_USE_START_DATE 211
LM_A_USER_CRYPT 211
    ls_user_crypt 140
LM_A_USER_EXITCALL 86
LM_A_USER_OVERRIDE 87
LM_A_USER_RECONNECT
    lc_timer 50
LM_A_USER_RECONNECT_DONE 88

LM_A_USERNAME
limits 202
LM_A_VAL_TYPE 76
lc_set_attr 56
LM_A_VD_FEATURE_INFO 88
lc_get_attyr 48
LM_A_VD_GENERIC_INFO 88
lc_get_attr 48
LM_A_VENDOR_GETHOSTID 91
LM_A_VENDOR_ID_DECLARE 90
LM_A_WINDOWS_MODULE_HANDLE 91
LM_BAD_VERSION
lc_checkout 36
LM_BADCHECKSUM 194
LM_BADCODE 189
debugging hints 148
lc_check_key 30
lc_checkout 35
LM_BADCOMM 190
LM_BADDATE 190
LM_BADFEATPARAM 192
lc_checkout 35
LM_BADFEATSET 61, 194
LM_BADFILE 69, 189
LM_BADHANDSHAKE 192
lc_checkout 36
LM_BADHOST 190
LM_BADKEYDATA 54, 193
LM_BADPARAM 36, 193
lc_check_key 30
lc_convert 38
LM_BADPLATFORM 193
LM_BADSYSDATE 197
lc_checkout 36
LM_BADVENDORDATA 54, 195
LM_BUSYNEWSERV 36, 191

LM_CANT_DECIMAL 198
LM_CANTCOMPUTEFEATSET 61, 194
LM_CANTCONNECT 190
    lc_checkout 36
LM_CANTFINDETHER 192
LM_CANTMALLOC 193
LM_CANTREAD 191
LM_CANTWRITE 191
LM_CHECKINBAD 191
LM_CHECKOUTFILTERED 194
LM_CI_ALL_FEATURES 31
LM_CLOCKBAD 192
LM_CO_LOCALTEST 33
    lc_auth_data 30
LM_CO_NOWAIT 33
    lc_status 57
LM_CO_QUEUE 33
    lc_status 57
LM_CO_WAIT 32, 33
    lc_status 57
LM_CODE 212
lm_code.h 141
LM_CRYPT_DECIMAL
    lc_cryptstr 40
LM_CRYPT_FORCE 40
LM_CRYPT_IGNORE_FEATNAME_ERRS 40
LM_CRYPT_ONLY 40
LM_DATE_TOOBIG 194
LM_DEFAULT_SEEDS 54, 197
LM_DUP_DISP 34
LM_DUP_HOST 34
LM_DUP_NONE 34
LM_DUP_USER 34
LM_DUP_VENDOR 34
    LM_A_CHECKOUT_DATA 77
LM_ENDPATH 196

LM_EXPIRED_KEYS 54
LM_EXPIREDKEYS 194
LM_FEATEXCLUDE 193
LM_FEATNOTINCLUDE 193
LM_FEATQUEUE 36, 192
    lc_status 57
LM_FLEXLOCK2CKOUT 199
LM_FUNCNOTAVAIL 36, 61, 193
LM_FUTURE_FILE 197
    lc_check_key 30
LM_HOSTDOWN 198
lm_isres 59
LM_KEY_NO_DATA 54
LM_LGEN_VER 197
LM_LIBRARYMISMATCH 54, 195
LM_LICENSE_FILE
    lc_lic_where 69
    license-file list 150
    limits 203
    LM_A_DISABLE_ENV 208
    LM_A_LICENSE_DEFAULT 82
    lmgrd 130
    multiple jobs 21
    redundant servers via 144
LM_LOCALFILTER 36, 195
    LM_A_CHECKOUT_FILTER 78
LM_LOG_MAX_LEN 53
LM_LONGGONE 190
LM_MAXLIMIT 197
LM_MAXUSERS 36, 189
LM_NEVERCHECKOUT 58, 193
lm_new.o 53
lm_new.obj 53
LM_NO_SERVER_IN_FILE 69, 190
LM_NOADMINAPI 48, 196
    LM_A_VD_*_INFO 88

LM_NOCLOCKCHECK 193
LM_NOCONFFILE 189
LM_NODONGLE 198
LM_NODONGLEDRIVER 199
LM_NOFEATSET 60, 194
LM_NOFEATURE 36, 44, 189
    lc_auth_data 29
LM_NOFILEVSEND 60
LM_NOFLEXLMINIT 194
LM_NOKEYDATA 193
LM_NONETWORK 54, 195
LM_NOREADLIC 192
LM_NORESLINK 198
LM_NOSERVCAP 196
LM_NOSERVER 189
LM_NOSERVICE 189
LM_NOSERVRESP 194
LM_NOSERVSUPP 36, 191
LM_NOSOCKET 189
LM_NOSUCHATTR 56
    lc_get_attr 47
LM_NOT_THIS_HOST 198
LM_NOTLICADMIN 70, 71, 195
LM_NOTTHISHOST 190
LM_OBJECTUSED 197
LM_OLDVENDORDATA 54, 195
LM_OLDVER 36, 191
LM_PLATNOTLIC 197
LM_POOL 197
LM_REMOVE_LINGER 198
LM_REMOVETOOSOON 70, 195
LM_RESVFOROTHERS 198
LM_SELECTERR 191
LM_SERVBUSY 36, 191
LM_SERVER
    lc_master_list 69

LM_SERVER_MAXED_OUT 198
LM_SERVER_REMOVED 197
LM_SERVLONGGONE 192
LM_SERVNOREADLIC 195
LM_SERVOLDVER 196
LM_SETSOCKFAIL 194
LM_SOCKETFAIL 194
LM_TCP 207
LM_TOOMANY 192
LM_UDP 207
LM_USER_BASED 196
LM_USERSQUEUED 191
    lc_status 57
LM_VENDOR_DOWN 198
LM_VER_BEHAVIOR
    LM_A_LICENSE_CASE_SENSITIVE 82
lmadmin
    and lmgrd 129
    lc_remove 70
lmadmin group 68
    lmgrd 132
lmclient.c 19
lmclient.h
    building your application 28
lmdown
    lc_shutdown 71
    lmgrd -x lmdown 130
lmerrors.h
    internationalization 20
lmflex.c 19
lmgrd
    definition 14
    internationalization for solaris 20
    limits 203
    syntax and use 129
lmgrd log file 183

lmrandom 133
lmremove
    lmgrd -x lmremove 130
lmreread
    and lmgrd 129
lmsimple.c 19
lmstat
    lc_userlist 58
    LM_A_CHECKOUT_DATA—DUP_VENDOR 78
lmstrip
    overview 23
lmswitch
    VMS command 159
lmutil
    Internationalization for Solaris 20
Log File 183
logfile
    lmgrd debug logfile 130
ls_attr.h 141
ls_compare_vendor_on_increment 135
ls_compare_vendor_on_upgrade 135
ls_conn_timeout 135
ls_crypt_case_sensitive 135
ls_daemon_periodic 136
ls_do_checkroot 136
ls_dump_send_data 136
ls_enforce_startdate 136
ls_get_attr 141
ls_incallback 137
ls_infilter 137
ls_minimum_user_timeout
    lc_timer 50, 73
ls_outfilter 138
ls_read_wait 137
ls_show_vendor_def 78, 138
ls_tell_startdate 138

ls_use_all_feature_lines 138
ls_user_crypt 139
ls_user_init1 139
    LM_A_ETHERNET_BOARDS 209
ls_user_init2 139
ls_user_lockfile 140
ls_vendor_msg
    lc_vsend 59
LSAPI 177, 179
lsvendor.c 133, 134
    lc_timer 50
    lc_vsend 59
M
MAX_OVERDRAFT 95
MAX_VENDOR_CHECKOUT_DATA—DUP_VENDOR 78
Multiple Jobs 21
N
NCR 156
Netware
    lmgrd 130
    no redundant support 143
    starting vendor daemon 131
Netware NLM 173
Netware Server 174
number of licenses
    FEATURE line 101
O
OpenVMS
    hostid 118
OPTIONS=SUITE 108
OVERDRAFT
    and Lenient Licensing 94
    detecting with SUITEs 90
    limits 201
P
PACKAGE

and Demo licensing 93
pclose()
debugging 147
PLATFORMS= 197
FEATURE attribute 104
Platform-Specific Notes
SGI 156
VAX/VMS 157
port address
allowed license file changes 97
port@host
and redundant servers 115
contacting license server 115
Q
Quick-Win 164
R
redundant servers 143
connection timeout 135
multiple jobs 21
REPORTLOG
and Lenient Licensing 94
RS/6000 155
S
SCO 157
hostid 119
SDK standard callback routines 162
security guidelines 23
SERVER
allowed license file changes 97
line syntax 97
redundant servers 144
server node
definition 14
server nodes
deciding number of 143
definition 14

multiple 14
setitimer
LM_A_SETITIMER 85
SGI 156
sgi64 157
SIGALRM 77
debugging 147
SIGPIPE
debugging 147
Simple API 13
example application 19
overview 15
SITE LICENSE
dup_group 34
sleep()
debugging 147
SN=
FEATURE attribute 104
Software License Working Group 177
Software License Working Group API 177
Solaris 157
Source
Installation 17
SPX/IPX 175
START=
FEATURE attribute 104
START_LICENSE 115
start-date
limits 201
Sub-nets
limits 203
SUITE
detecting OVERDRAFT in 90
SUITE_DUP_GROUP
FEATURE attribute 105
SUPERSEDE

FEATURE attribute 105
SVR4 157
system()
debugging 147
T
TCP
communications transport 151
lc_checkin 31
LM_A_TCP_TIMEOUT 86
this_host
lc_convert 38
Time Zone (Windows) 162
TIMEOUT
lc_idle 51
lc_timer 50, 73
timeout value
setting 137
TRANSPORT
UDP vs TCP 151
Trivial API 13
example application 19
overview 15
TZ environment variable 162
U
UDP 86
communications transport 152
lc_checkin 31
lc_idle 52
lc_timer 73
Ultrix 209
Unix
lmstrip and Unix libraries 24
UPGRADE
ls_use_all_feature_lines 139
syntax 107
USER

dup_group 33
USER_BASED
FEATURE attribute 105
limits 201
V
VAX
ethernet hostid 66
VENDOR
debug log file 183
dup_group 33
line syntax 98
vendor
FEATURE line 101
Vendor Daemon
configuring 133
limits 202
vendor daemon
automatic reconnection 87
building 133
definition 14
detecting daemon death 77
internationalization for Solaris 20
vendor daemon log file 183
VENDOR_KEY
and ENCRYPTION_SEEDs 140
VENDOR_KEY5
lc_crypt 62
lc_cryptstr 39
VENDOR_STRING
FEATURE attribute 106
VENDORCODE 212
lc_crypt 62
Vendor-defined hostid 120
limits 202
Version
limits 201

version
    FEATURE line 101
VMS 157, 207, 208, 209
    and VENDOR line 99
    building vendor daemon 133
    communications transport 151
    Installation 17
    lc_isadmin 68
    lmgrd 130
    redundant servers 143
VMS FLEXlm 157
W
Wide-Area Networks
    limits 203
Windows
    FLEXlm differences 161
    hostids 118
    lc_perror 55
Windows NT 161
    and VENDOR line 99
    lmgrd 131
WinSock 161
WSACleanup 161
WSAStartup 161
X
X-Display name 79
XOpenDisplay 79
XtAppInitialize
    LM_A_DISPLAY_OVERRIDE 79