
# *Programmers Guide*

**VERSION 7.1**

GLOBEtrotter Software, Inc.

September 2000

Printed in the USA.

September 2000

Table of

# Contents

Chapter 1

# Introduction

## 1.1 About This Manual

This manual, the *FLEXlm Programmers Guide*, is an introduction to FLEX*lm*® and includes a complete description of the simplest Application Programming Interfaces (APIs) that can be used to incorporate license management into your application. This manual describes the license administration tools that are bundled with FLEX*lm* and provides guidelines for integration of FLEX*lm* into your application. With this manual, you should be able to have an example license-managed application up and running within a few hours.

The *FLEXlm Reference Manual* provides a comprehensive description of all other aspects of FLEX*lm* from the software developer's perspective, including a complete description of the FLEXible API, the most complete API available for license management. Companies upgrading from a version older than v7.1 should refer to the *FLEXlm Reference Manual*, Appendix F, "Migrating to the Counterfeit Resistant Option."

The *FLEXlm End Users Guide* contains information for products that utilize FLEX*lm* as their licensing system. It describes the setup and administration of a FLEX*lm* licensing system, including setting up an options file and using FLEX*lm* license administration tools.

## 1.2 How to Use This Manual

If you are getting started with FLEX*lm* on UNIX, we suggest that you read Chapters 1, 2, 4, and 5 of this manual. If you are getting started with FLEX*lm* on Windows, we suggest that you read Chapters 1, 3, 4, and 6. After completing those chapters, you will have installed FLEX*lm*, run a sample application with license management in a number of different situations, and built a production FLEX*lm* SDK.

Once you are familiar with how FLEX*lm* operates, the remainder of this manual provides reference material for integrating FLEX*lm* into your application. It includes instruction in the use of the administration tools provided with FLEX*lm* and general guidelines on configuring your application and its licensing software.

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

## 1.3 Typographic Conventions

The following typographic conventions are used in this manual:

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

## 1.4 Introduction to FLEX*lm*

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

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

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

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

- Operation in a heterogeneous network of supported computer systems
- Transparent reconnection of applications when their license server process becomes unavailable, including conditions of license server node failure
- Simple configuration by using a single license file per network
- Configuration controls for system administrators
- Administration tools for system administrators
- Independent features from one or multiple vendors with independent vendor security codes

- License management on redundant server hosts for improved license availability due to hardware failure
- A wide variety of license policies and license styles, including:
  - Floating licenses
  - Node-locked licenses
  - Personal use licenses
  - Demo licenses
  - Counted and uncounted licenses
  - Optional license expiration dates
  - Several vendor-definable fields for each application feature

## 1.5 FLEX*lm* Terms and Definitions

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

| | |
|---|---|
| Feature | Any functionality that needs to be licensed. The meaning of a feature will depend entirely on how it is used by an application developer. For example, a feature could represent any of the following:<br>• An application software system consisting of hundreds of programs<br>• A single program (regardless of version)<br>• A specific version of a program<br>• A part of a program<br>• A piece of data (restricted via the access routines) |
| License | The legal right to use a feature. FLEX*lm* can restrict licenses for features by counting the number of licenses already in use for a feature when new requests are made by the application software (*client*). FLEX*lm* can also restrict software usage to particular nodes or user names. |
| Client | An application program requesting or receiving a license. |

| | |
|---|---|
| Daemon | A process that “serves” clients. Sometimes referred to as a *server.* |
| Vendor daemon | The server process that dispenses licenses for the requested features. This binary is built by an application’s vendor (from libraries supplied by GLOBEtrotter Software) and contains the vendor’s unique encryption seeds. |
| Vendor name | Name of the vendor as found in `lm_code.h`. Used as the name of the vendor daemon. |
| lmgrd | The daemon process, or license manager daemon, that sends client processes to the correct vendor daemon on the correct machine. The same license manager daemon process can be used by any application from any vendor because this daemon neither authenticates nor dispenses licenses. `lmgrd` processes no user requests on its own, but forwards these requests to the vendor daemons. |
| Server node | A computer system that runs the license server software. The server node will contain all site-specific information regarding all feature usage. Multiple server nodes used for redundancy can logically be considered the server node. |
| License file | A text file specific to an end-user site that contains descriptions of 1) license server node(s), 2) vendor daemons, and 3) licenses (features) for all supported products. |
| License file list | A list of license files separated with a colon “ : ” on UNIX and a semi-colon “ ; ” on Windows. A license file list can be accepted in most places where a license file is appropriate. When a directory is specified, all files matching `*.lic` in that directory are automatically used, as if specified as a list. |
| License key | See the *FLEXlm Reference Manual*, Appendix F, Migrating to the Counterfeit Resistant Option.” |

| | |
|---|---|
| Signature | A secure 12- to 120-character hexadecimal number which “authenticates” the readable license file text, ensuring that the license text has not been modified. |
| License server | An `lmgrd` and one or more vendor daemon processes. License server refers to the processes, not the computer on which they run. |

## 1.6 How FLEX*lm* Works

FLEX*lm* is a client-server application toolkit. The client (your application) requests a license from the license server and is either granted or denied the license.

The five main components of FLEX*lm* license management are:

- FLEX*lm* client library (embedded in the license-managed application)
- `lmgrd`, the license manager daemon
- Vendor daemon(s)
- Vendor and end-user license administration tools
- License file(s)

The end user installs `lmgrd`, the vendor daemon(s), and the license file(s) on the license server node. Once the license file(s) and the daemons are in place, the only requirement is to start `lmgrd`. The license manager daemon is typically started when the machine boots (in the machine startup file, or on Windows as a system service), but can also be started later by any user.

With the license file installed in an expected location and the FLEX*lm* daemons running, FLEX*lm* is transparent to the end user.

**SEE ALSO**

- Chapter 12, “The License File”
- Chapter 15, “End-User License Administration”

## 1.7 What's Installed

The following directories can be found in your FLEX*lm* installation:

| | |
|---|---|
| *platform* | Platform-specific executables and files necessary for evaluation, e.g., sun4_u5, i86_n3. |
| htmlman | Online documentation. |
| machind | Platform-independent files, include lm_code.h, manuals in PDF and/or PostScript format, and sample program source files. |
| examples | More example programs. |
| flexid7 | (Windows only) |
| flexid8 | (Windows only) |

The following files will be installed into each *platform* directory:

| | |
|---|---|
| demo/demo.exe | Evaluation vendor daemon |
| license.dat/ demo.lic | Sample license files |
| lmclient/ lmclient.exe | Sample FLEX*lm* command-line client program, uses Trivial API |
| lmcrypt/ lmcrypt.exe | FLEX*lm* license generation program |
| lmgrd/ lmgrd.exe | FLEX*lm* license manager daemon |
| lmutil/ lmutil.exe | Command-line utilities for managing a FLEX*lm* license server |
| makekey/ makekey.exe | Sample program for making license files |

In the *platform* directory on Windows, you will also find:

| | |
|---|---|
| `build.bat` | Batch file to build FLEX*lm* SDK |
| `flsetup.exe` | FLEX*lock* setup program |
| `genlic.exe` | Evaluation program for making license files |
| `installs.exe` | NT service installer for `lmgrd.exe` |
| `lmtools.exe` | GUI utilities for managing a FLEX*lm* license server |
| `lmwin.exe` | Sample FLEX*lm* GUI client application, uses Trivial API |

Chapter 2

# Evaluating FLEX*lm* on UNIX

This chapter will walk you though the process of evaluating FLEX*lm* behavior on UNIX. Before you start this chapter, please install the FLEX*lm* software as described in the FLEX*lm* QuickStart.

During the installation of FLEX*lm*, you built your demo FLEX*lm* Software Development Kit using the demo vendor keys you received from GLOBEtrotter. The *platform* directory (for example, `../sun4_u5/license.dat`) of the FLEX*lm* SDK contains the following files that you will use to evaluate FLEX*lm*:

- Sample license files (`license.dat`, `demo.lic`)
- Sample vendor daemon (`demo`)
- Sample client application (`lmclient`)
- License generation utility (`lmcrypt`)

GLOBEtrotter provides `lmutil`, a set of FLEX*lm* command-line utilities, to software vendors to distribute to their customers. You have copies of these utilities in the *platform* directory and will use some of them during your evaluation of FLEX*lm*. A complete description of `lmutil` commands can be found in the *FLEXlm End Users Guide*.

Note that FLEX*lm* demo vendor keys expire, and, therefore, the demo SDK that you built with the demo vendor keys expires. If your demo vendor keys expire before you complete this evaluation, contact your salesperson at GLOBEtrotter Software.

Now you are ready to begin your evaluation of FLEX*lm*.

## 2.1 Rekeying the Sample License File

A demo license file (`license.dat`) is shipped in the *`platform`* directory. This sample license will allow its license server to run on any machine (because of the server hostid "ANY" on the SERVER line) and is for a license count of 4 for feature "f1" served by the vendor daemon `demo`.

1. Open `license.dat` in a text editor. You will see something like:

   ```
   SERVER this_host ANY
   USE_SERVER
   VENDOR demo
   # a counted license
   FEATURE f1 demo 1.0 permanent 4 SIGN=E95FD936F845
   ```

2. You will be writing over `license.dat` during this demo, so copy it to a file called `licenseOrig.dat`.
3. The SIGN= attribute on the "f1" FEATURE line may not be up to date (the signature in this license file was not regenerated when you rebuilt the demo SDK). You will now regenerate the signature to ensure that it is valid. In the `platform` directory, type:

   ```
   % lmcrypt license.dat
   ```

## 2.2 Starting the Demo License Server

A FLEX*lm* license server is the combination of a running `lmgrd` process and one or more running vendor daemons (in this case, the `demo` vendor daemon). Starting the demo license server is necessary because the license for "f1" is *counted*, that is, its license count is non-zero. A license server is not needed for *uncounted* licenses (that is, where the license count is the numeral "0" or the keyword "uncounted").

1. To start the demo license server, in the *`platform`* directory, type:

   ```
   % lmgrd -c license.dat
   ```

   After `lmgrd` starts, it will start the `demo` vendor daemon. For the purposes of this demo, you have started the license server without specifying a debug log—to force the license server processes (`lmgrd` and `demo`) to write their debugging output to the window where you started `lmgrd`. However, in a production setting, you should start `lmgrd` with a flag to write its output to a debug log file, `lmgrd.dl`, in the current directory by typing:

   ```
   % lmgrd -c license.dat -l lmgrd.dl
   ```

   (That is the letter "`l`" for "log," not the number "`1`.")

2. After you start `lmgrd`, the output of the license server will look like:

```
11:11:49 (lmgrd) FLEXlm (v7.1b) started on myhost (Sun) (5/3/2000)
11:11:49 (lmgrd) FLEXlm Copyright 1988-2000, Globetrotter Software,
  Inc.
11:11:49 (lmgrd) US Patents 5,390,297 and 5,671,412.
11:11:49 (lmgrd) World Wide Web:  http://www.globetrotter.com
11:11:49 (lmgrd) License file(s): license.dat
11:11:49 (lmgrd) lmgrd tcp-port 27000
11:11:49 (lmgrd) Starting vendor daemons ...
11:11:49 (lmgrd) Started demo (internet tcp_port 34641 pid 9159)
11:11:49 (demo) FLEXlm version 7.1b
11:11:49 (demo) Server started on myhost for:    f1
```

3. Open another window and change to the *`platform`* directory (e.g., `sun4_u5`). Leave the first window open to monitor the output of `lmgrd` and the `demo` vendor daemon.

## 2.3 Checking the License Server Status

Now you will check the status of the license server to see whether it is up and who is using how many of the licenses that it is serving.

1. To check the license server status, in the *`platform`* directory, type:

```
% lmstat -a -c license.dat
```

2. You should see output like:

```
License server status: 27000@myhost
  License file(s) on myhost: /flexlm/v7.1/sun4_u5/license.dat:
  myhost: license server UP (MASTER) v7.1
Vendor daemon status (on myhost):
  demo: UP v7.1
Feature usage info:
Users of f1:  (Total of 4 licenses available)
```

Note that no users have any of the four licenses for “f1” checked out.

## 2.4 Running the Sample Application

You will start the sample client application, `lmclient`, and will use it to check out a license for the feature “f1.”

1. In the *`platform`* directory, type:

```
% lmclient
```

You will see:

```
Enter feature to checkout [default:"f1"]:
```

2. Press Enter to check out a license for feature “f1.”

3. Check the tail of the license server output to see the checkout of “f1.”

   ```
   11:22:35 (demo) OUT:”f1” daniel@myhost
   ```

4. In a third window, change to the *platform* directory.
5. To check the status of the license server with a license checked out, in the third window, again type:

   ```
   % lmstat -a -c license.dat
   ```

   You will see output with user, host, and display information about the “f1” license that you just checked out:

   ```
   Users of f1:  (Total of 4 licenses available)
     "f1" v1.0, vendor: demo
     floating license
       daniel myhost 19.16.18.26 (v1.0) (myhost/27000 102), start Fri
             5/3 7:29
   ```

6. Close the window where you just ran `lmstat`.
7. In the window where you are running `lmclient`, you will see:

   ```
   f1 checked out... press return to exit...
   ```

8. Press Enter to check the f1 license back in.
9. Check the tail of the license server output to see the checkin of “f1:”

   ```
   11:22:40 (demo) IN:”f1” daniel@myhost
   ```

## 2.5 Trying to Check Out an Unlicensed Feature

1. Run `lmclient` again, and instead of selecting the default feature “f1” for which you have a license, type `f3` as the feature to check out.
2. You will see that the checkout failed because “f3” is not supported by the license server:

   ```
   Checkout failed: License server does not support this feature
   Feature: f3
   License path:  @localhost:license.dat:./demo.lic
   FLEXlm error:  -18,147
   For further information, refer to the FLEXlm End User Manual,
   available at "www.globetrotter.com".
   ```

## 2.6 Editing the Sample License File

Now you will edit the sample license file to try to give yourself a license for feature “f3.”

1. Open `license.dat` in a text editor.
2. Duplicate the “f1” FEATURE line on the line below.

3. On the second FEATURE line, replace `f1` with `f3`.
4. Save and close `license.dat`.
5. Tell the license server to reread its edited license file, by typing:

   ```
   % lmreread -c license.dat
   ```

6. Run `lmclient` again and try to check out "f3."

   See that you still get an error because you changed a FEATURE line in a license file and did not regenerate its signature.

## 2.7 Rekeying the License File

You will now generate a license file with valid FEATURE lines for both "f1" and "f3" with `lmcrypt`.

1. In the *`platform`* directory, type:

   ```
   % lmcrypt license.dat
   ```

2. The license file should now resemble:

```
SERVER this_host ANY
USE_SERVER
VENDOR demo
# a counted license
FEATURE f1 demo 1.0 permanent 4 SIGN=E95FD936F845
FEATURE f3 demo 1.0 permanent 4 SIGN=E9A1D9763885
```

3. Reread the license file again by typing:

   ```
   % lmreread -c license.dat
   ```

4. Run `lmclient` again and try to check out "f3." Now that you have a valid license for "f3," you are able to check out a license for it.

## 2.8 Stopping the Demo License Server

When you are done experimenting with `lmclient` and your license file, stop the license server.

1. In the *`platform`* directory, type:

   ```
   % lmdown -c license.dat
   ```

   You will see a description of the license server and a request for confirmation:

```
    Port@Host          Vendors
1) 27000@myhost        demo
Are you sure (y/n)?
```

2. Confirm that you want to shut down the demo license server by typing `y`.
3. Look at the tail of the license server output:

```
11:35:46 (lmgrd) SHUTDOWN request from daniel at node myhost
11:35:46 (lmgrd) lmgrd will now shut down all the vendor daemons
11:35:46 (lmgrd) Shutting down demo
11:35:46 (demo) daemon shutdown requested - shutting down
```

## 2.9 Testing an Uncounted License

In addition to `license.dat`, you have a second demo license file in the *`platform`* directory named `demo.lic`:

1. Open `demo.lic` in a text editor.

```
# an uncounted license
FEATURE f2 demo 1.000 permanent uncounted HOSTID=ANY \
  SIGN=05E5BF3088AD
```

2. Close `demo.lic`.
3. In the *`platform`* directory, run `lmclient` and try to check out feature "f2."

   Because the license file for "f2" is in the same directory as `lmclient` and has an extension of `.lic`, its license is found and used. Because the license for "f2" is uncounted, it does not need a license server to allow it to run.

## 2.10 Evaluating the Counterfeit Resistant Option (CRO)

The Counterfeit Resistant Option uses public-key technology from Certicom to make the signatures on FEATURE/INCREMENT lines more difficult to counterfeit. It is a separately priced add-on to FLEX*lm* v7.1. However, during your evaluation of FLEX*lm* v7.1, you can edit one file, rebuild the demo SDK, and evaluate CRO.

### 2.10.1 Adding CRO to lm_code.h

1. In the `machind` directory, open `lm_code.h` in a text editor.
2. Find the definition of `LM_STRENGTH`. It is probably set to LM_STRENGTH_DEFAULT.
3. Redefine `LM_STRENGTH` to specify the length of your license signature.

```
#define LM_STRENGTH strength
```

where *strength* is one of:

| *strength* | Bits | Number of hex characters |
|---|---|---|
| LM_STRENGTH_113BIT | 113 | 58 |
| LM_STRENGTH_163BIT | 163 | 84 |
| LM_STRENGTH_239BIT | 239 | 120 |

To demo FLEX*lm*, you do not need CRO_KEYs, but for a production FLEX*lm* kit with CRO enabled, you will need to enter CRO_KEYs into lm_code.h (see Section 5.1, “Editing lm_code.h”).

4. Save and close lm_code.h.

### 2.10.2 Rebuilding the FLEX*lm* Demo SDK

Type make in the *platform* directory (e.g., sun4_u5) to rebuild the demo vendor daemon, lmcrypt, etc. (If you are doing this command-line demo on Windows, type nmake in the i86_n3 folder.)

### 2.10.3 Rekeying the Sample License File

1. In the *platform* directory (e.g., sun4_u5), type:

   ```
   % lmcrypt license.dat
   ```

2. Open license.dat in a text editor. Depending on the value of LM_STRENGTH that you chose (this example uses LM_STRENGTH_113BIT), you will see something like:

   ```
   SERVER this_host ANY
   USE_SERVER
   VENDOR demo
   # a counted license
   FEATURE f1 demo 1.0 permanent 4 SIGN="00F4 8B8B 8597 AB54 \
     C20E 21B4 E90B 1000 F6EE 67A1 86C8 B823 6D6C 6995 EC79"
   FEATURE f3 demo 1.0 permanent 4 SIGN="00E1 3742 FDE6 A9D0 \
     B835 A673 26A9 CD00 41BB EFE2 600A DDE3 F3D1 664F A2F6"
   ```

**Note:** Each time you run CRO-enabled `lmcrypt` on a license file, you will generate a different set of signatures for the FEATURE/INCREMENT lines in that license file. The signature field is not, therefore, used to determine whether two INCREMENT lines are identical.

3. Close `license.dat`.

Now, if you'd like, you can run through the same evaluation of FLEX*lm* with CRO enabled—FLEX*lm* runs the same with CRO enabled, the only difference is in the way the signature is generated and authenticated.

You have completed the demo of FLEX*lm* on UNIX. When you are ready to build your FLEX*lm* SDK with your production vendor keys, see Chapter 4, "Adding FLEXlm Function Calls Into Your Application." If you want to evaluate FLEX*lm* on Windows, see Chapter 3, "Evaluating FLEXlm on Windows." Remember to set `LM_STRENGTH` to an appropriate value before you build your production FLEX*lm* SDK.

Chapter 3

# Evaluating FLEX*lm* on Windows

This chapter will walk you though the process of evaluating FLEX*lm* behavior on Windows. Before you start this chapter, please install the FLEX*lm* software as described in the FLEX*lm* QuickStart.

During the installation of FLEX*lm*, the Windows registry was told about the sample license file, `license.dat` and a DEMO license server was configured. In addition to the sample license file, the FLEX*lm* SDK contains pre-built demo files to use to evaluate FLEX*lm*:

- Uncounted license (`demo.lic`)
- Vendor daemon (`demo.exe`)
- Sample client application (`lmwin.exe`)
- License generation utility (`genlic.exe`) — pre-built, but needs valid demo vendor keys to run

Note that the pre-built demo SDK and the demo vendor keys that ship with FLEX*lm* expire. If your pre-built files or your demo vendor keys expire before you complete this evaluation, contact your salesperson at GLOBEtrotter Software.

Now you are ready to begin your evaluation of FLEX*lm*.

## 3.1 Evaluating FLEX*lm* with a Pre-Built Client Application

### 3.1.1 Viewing the Sample License Files

A sample license file (`license.dat`) is shipped in the `i86_n3` folder and was used to set up a DEMO license server when you installed FLEX*lm*.

1. Open `license.dat` in a text editor. You will see something like:

```
SERVER this_host ANY
VENDOR demo
USE_SERVER
FEATURE f1 demo 1.0 permanent 4 SIGN=E95FD936F845
```

2. You will be writing over `license.dat` during this demo, so copy it to a file called `licenseOrig.dat`.

   This sample license will run on any machine (because on the SERVER line the server hostid is “ANY”) and contains one FEATURE line for “f1” that has a license count of 4 and is served by the vendor daemon `demo`.

3. Close `license.dat` and open `demo.lic` in a text editor. You will see something like:

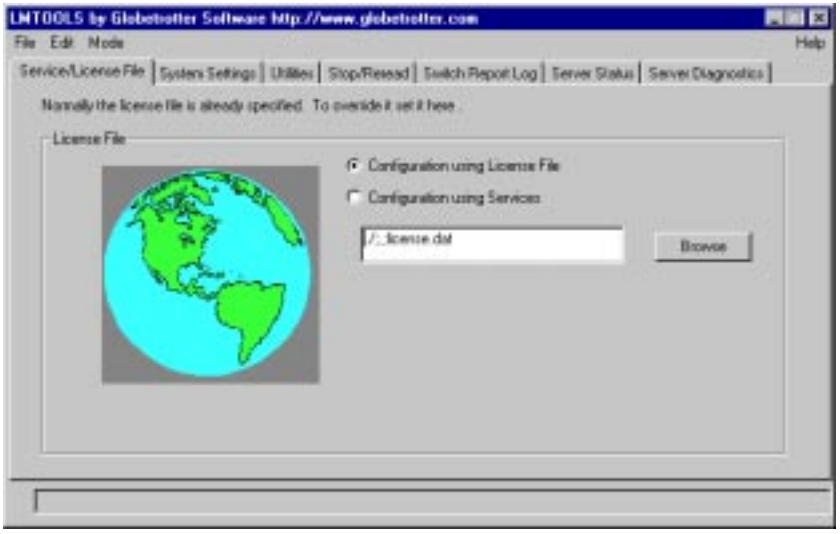
   ```
   # an uncounted license
   FEATURE f2 demo 1.000 permanent uncounted HOSTID=ANY \
     SIGN=05E5BD2088AD
   ```

   This license for “f2” is uncounted and does not need a license server to allow it to run.

4. Close `demo.lic`.

### 3.1.2 Starting lmtools

GLOBEtrotter provides tools for managing license servers to software vendors to distribute to their customers. On Windows, GLOBEtrotter provides `lmutil.exe`, a set of command-line utilities, and `lmtools`, a similar set of utilities with a graphical user interface. Start `lmtools` either by selecting Start→Programs→FLEXlm V7.1→FLEXlm Utilities or by double-clicking `lmtools.exe` in the `C:\Program Files\flexlm\v7.1\i86_n3` folder.

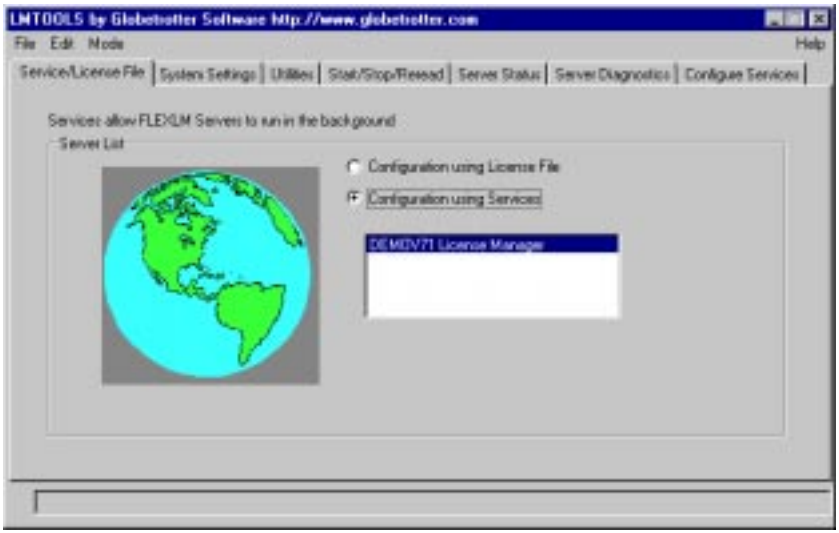


`lmtools` will appear with the Service/License tab open.

### 3.1.3 Starting the Demo License Server

A FLEX*lm* license server is the combination of a running `lmgrd` process and one or more running vendor daemons (in this case, the `demo.exe` vendor daemon). Starting the demo license server is necessary because the license for "f1" is *counted*, that is, its license count is non-zero. A license server is not needed for *uncounted* licenses (that is, where the license count is the numeral "0" or the keyword "uncounted").

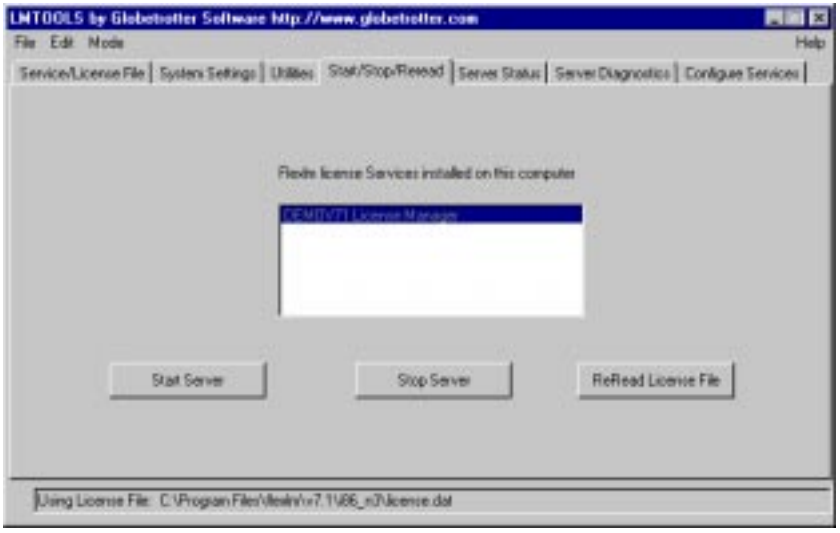
1. In the Service/License File tab, click the Configuration using Services radio button.




2. Click DEMOV71 License Manager.

   The DEMOV71 License Manager should have been installed during the demo installation of FLEX*lm*.

3. Click the Start/Stop/Reread tab.



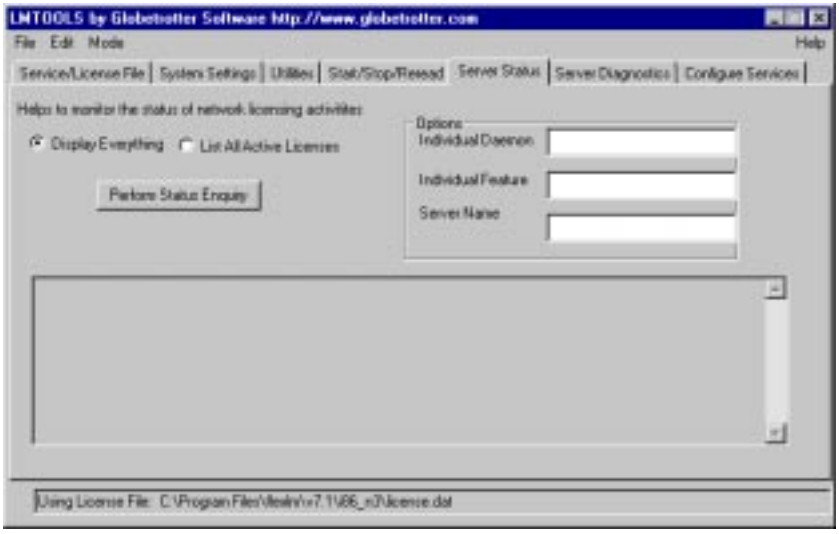

4. Click the Start Server button.

   The DEMOV71 License Manager license server starts and will write its debug log output to `C:\Program Files\flexlm\v7.1\i86_n3\lmgrd.dl`.

### 3.1.4 Checking the License Server Status

Now you will check the status of the license server to see whether it is up and who is using how many of the licenses that it is serving.

1. To check the license server status, click the Server Status tab.




2. Click the Perform Status Enquiry button.
3. Scroll down the display; you should see output like:

```
        demo: UP v7.1
Feature usage info:
Users of f1: (Total of 4 licenses available)
```

Note that no users have any of the licenses for “f1” checked out.

### 3.1.5 Running the Sample Application

You will start the sample client application, `lmwin.exe`, and will use it to check out a license for the feature “f1.”

1. Start `lmwin.exe` either by selecting Start→Programs→FLEXlm V7.1→FLEXlm Test Program or by double-clicking `lmwin.exe` in the `C:\Program Files\flexlm\v7.1\i86_n3` folder.


   By default, the feature that appears in the program’s display is “f1.” The feature “f1” has a *counted* license, that is, is has a non-zero license count. It is also a *floating* license, that is, “f1” can be run on any machine.

2. In `lmwin`, click the Checkout button. You will see that the checkout succeeded.
3. To check the status of the license server with a license checked out, click the Perform Status Enquiry button on the Server Status tab. Scroll down to see output with user, host, and display information about the “f1” license that you just checked out:

```
        demo: UP v7.1
Feature usage info:
Users of f1: (Total of 4 licenses available)
   “f1” v1.0, vendor: demo
   floating license
     daniel myhost myhost (v1.0) (myhost/27000 101), start...
```

4. If you could examine the tail of the debug log file, `C:\Program Files\flexlm\v7.1\i86_n3\lmgrd.dl`, you would see something like:

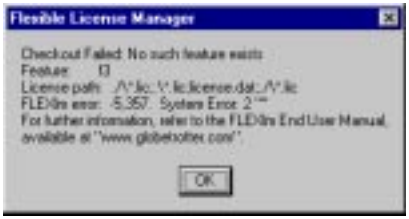
```
19:03:54 (demo) OUT:”f1” daniel@myhost
```

5. In `lmwin`, click the Checkin button to check the “f1” license back in.

6. Check the server status again to see that no licenses are checked out.

### 3.1.6 Trying to Check Out an Unlicensed Feature

1. In `lmwin`, replace “f1” with “f3” as the feature to check out.
2. Click the Checkout button.
3. You will see a dialog that informs you that the checkout failed because the license server doesn’t support this feature. Click OK.




### 3.1.7 Editing the Sample License File

Now you will edit the sample license file to try to give yourself a license for feature “f3.”

1. Open `license.dat` in a text editor.
2. Duplicate the “f1” FEATURE line at the bottom of the file.
3. On the second “f1” FEATURE line, replace `f1` with `f3`.
4. Save and close `license.dat`.
5. Click the Start/Stop/Reread tab in `lmtools`. Tell the license server to reread its edited license file, by clicking the ReRead License File button:
6. In `lmwin`, try to check out “f3” again.
7. You will see a dialog that tells you that you still have an error because you changed a FEATURE line in a license file and did not regenerate its signature. Click OK.

### 3.1.8 Rekeying the License File

You will now generate a license file with valid FEATURE lines for both “f1” and “f3” with `lmcrypt`.

1. Open a DOS command window and change to the `C:\Program Files\flexlm\v7.1\i86_n3` folder. In the this folder, type:

```
% lmcrypt license.dat
```

2. The license file should now resemble:

```
SERVER this_host ANY 27000
VENDOR demo
USE_SERVER
FEATURE f1 demo 1.0 permanent 4 SIGN=E95FD936F845
FEATURE f3 demo 1.0 permanent 4 SIGN=E9A1D9763885
```

3. In `lmtools`, reread the license file again:
4. In `lmwin`, try to check out “f3” again. Now that you have a valid license for “f3,” you are able to check out a license for it.
5. In `lmwin`, click the Checkin button to check in the “f3” license.

### 3.1.9 Stopping the Demo License Server

1. In `lmtools`, click the Start/Stop/Reread tab.
2. Click the Stop Server button.
3. Check the server status now that you have stopped the DEMOV71 license server. You will see:

```
lmgrd is not running: Cannot connect to license server
The server (lmgrd) has not been started yet, or
the wrong port@host or license file is being used, or the
port or hostname in the license file has been changed.
```

4. Look at the license server debug log file at `C:\Program Files\flexlm\v7.1\i86_n3\lmgrd.dl` to review the activity of the DEMOV71 license server.

### 3.1.10 Testing an Uncounted License

Remember that you started the DEMOV71 license server to be able to check out a counted “f1” license. Now you will see the difference in the behavior of “f1” and “f2” now that the DEMO license server is not running.

1. In `lmwin`, replace “lcm” with “f1” as the feature to check out.
2. Click the Checkout button to try to check out “f1.”
3. You will see that you can’t check out an “f1” license because the license server is down.
4. Now replace “f1” with “f2.” Remember that the license for “f2” is uncounted and node-locked and is located in `demo.lic`.

5. Click the Checkout button. Because the license file for “f2” is in the same directory as `lmwin.exe` and has an extension of `.lic`, its license is found and used. Because the license for “f2” is uncounted, it does not need a license server to allow it to run.
6. Click the Checkin button.
7. Close `lmwin`.
8. You will write over `license.dat` in the next section, so rename `license.dat` if you’d like to save its contents.

## 3.2 Adding Your Demo Vendor Keys to lm_code.h

For the demo version of `genlic.exe` to run correctly, you must enter your valid demo vendor keys into `lm_code.h`. You do not need to rebuild your demo FLEX*lm* SDK.

1. Change directories to `C:\Program Files\flexlm\v7.1\machind`.
2. Double-click `lm_code.h` to open it in a text editor or in the Microsoft Visual C++ editor. Find the lines similar to the following lines:

```
#define VENDOR_KEY1 0x0
#define VENDOR_KEY2 0x0
#define VENDOR_KEY3 0x0
#define VENDOR_KEY4 0x0
#define VENDOR_KEY5 0x0
```

3. Replace all five lines with the demo vendor key lines that you received from GLOBEtrotter. If your demo vendor keys have expired, contact your salesperson at GLOBEtrotter Software.
4. Save and close `lm_code.h`.

## 3.3 Generating and Testing Your Own Licenses

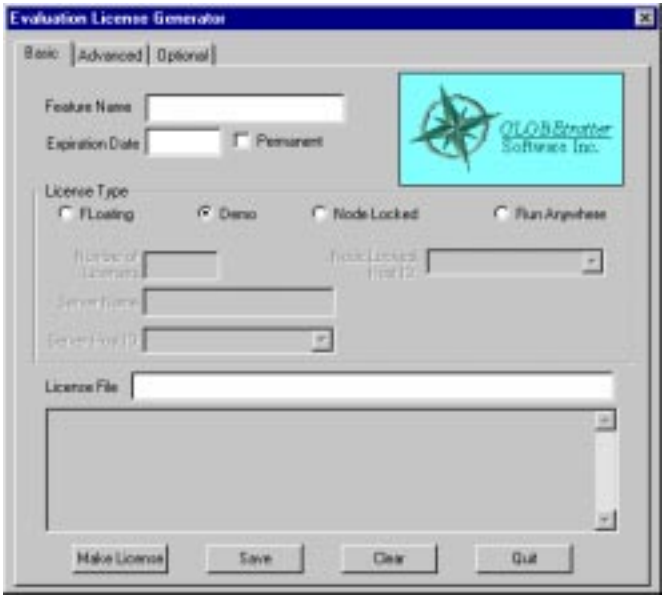
To generate your own licenses, you will use a pre-built demo version of a license generator interface, `genlic.exe`. Eventually, you’ll want to learn how to generate licenses with `lmcrypt`, but `genlic` is easier to get started with.

For more information, see Section 11.4, “genlic.exe (Windows Only).”

### 3.3.1 Test a Run Anywhere License

The *run anywhere* license that `genlic` generates is an uncounted license that will run on any computer. Choices of expiration date allow the license to be used for a limited or unlimited amount of time.

1. Select Start→Programs→FLEXlm V7.1→Evaluation License Generator or double-click `genlic.exe`.




2. Enter the feature name, “f1,” in the Feature Name field. Click the Permanent check box to make a non-expiring license. Choose the Run Anywhere radio button to set the License Type.
3. Click the Make License button. The text of the license file will appear in the display. Notice that this feature is uncounted. (To create multiple FEATURE lines, you would edit the form and click Make License again. Each FEATURE line will be appended to the display. For this example, however, you will add only one FEATURE line.)
4. Type `license.dat` in the License File field and click Save to create a license file called `license.dat`.
5. Run the program `lmwin.exe`, if it isn’t already running, located in the FLEX*lm* program group (FLEXlm Test Program). With feature “f1” selected, click the Checkout button. Note that the display shows that the license checkout has succeeded.

6. To release the license, click the Checkin button.

### 3.3.2 Test a Node-Locked License

The *node-locked* license that `genlic` generates is an uncounted license that runs only on a specific machine designated by a machine hostid on the FEATURE line.

1. Launch `genlic.exe` if it is not already running, or click the Clear button at the bottom if it is still running from the prior test.
2. Enter the feature name "f1" in the Feature Name field. Click the Permanent check box to make a non-expiring license. Choose the Node Locked radio button to set the License Type.
3. Open the Node Lock Host ID choice list and choose the DISK_SERIAL_NUM entry. This is the disk volume serial number for Drive C on your computer.
4. Click the Make License button. The license line will appear in the display. This feature is also uncounted.
5. Make sure that `license.dat` is in the License File field. Click Save to create a license file called `license.dat`. When the dialog warns that `license.dat` already exists, click Yes to overwrite the file.
6. Run `lmwin.exe` (if it's not already running).
7. Click the Checkout and Checkin in the `lmwin` window to test the license. This license will work only on this computer because you have specified the hostid of DISK_SERIAL_NUM with a value specific to this computer. If you were to run the test application with this license file on another computer, it would not run because that computer would not have this same hostid.

### 3.3.3 Test a Floating License

A *floating* license illustrates a type of licensing sometimes called *concurrent licensing* or *server-based licensing*. A floating license is counted and can be used by different people on a network as long as the license limit is not exceeded.

1. Launch `genlic` if it is not already running, or click the Clear button at the bottom if it is still running from the prior test.
2. Enter the feature name "f1" in the Feature Name field. Click the Permanent check box to make a non-expiring license. Choose the Floating radio button to set the License Type.

3. In the Number of Licenses field, enter `1`. In the Server Name field enter the host name of your computer. In the Server Host ID field, pull down the choice list and select Any.
4. Click the Make License button. Three lines (SERVER, VENDOR, and FEATURE) will appear in the display. Note that this license is counted.
5. Make sure that `license.dat` is in the License File field and then click the Save button to create a license file called `license.dat`. When the dialog warns that `license.dat` already exists, click Yes to overwrite the file.
6. Because this license is counted, you must start a license server. You will configure a new test license server (FLEX*lm* service) using `lmtools`. Start `lmtools` and select the Configuration using Services radio button. Click the Configure Services tab, click the existing Service Name, and press the Delete key. In the blank Service Name field, type:

   ```
   mytest
   ```

   Press the Tab key to clear the rest of the form. Enter the following paths in the form, assuming you installed FLEX*lm* in the default location:

   Path to the `lmgrd.exe` file:

   ```
   C:\Program Files\flexlm\v7.1\i86_n3\lmgrd.exe
   ```

   Path to the license file:

   ```
   C:\Program Files\flexlm\v7.1\i86_n3\license.dat
   ```

   Path to the debug log file:

   ```
   C:\Program Files\flexlm\v7.1\i86_n3\lmgrd.dl
   ```
7. Click the Save Service button to save the mytest service as a FLEX*lm* service. If you had checked the Use Services check box, mytest would also be an NT service. Confirm that you want to save the settings for the mytest service. In the Start/Stop/Reread tab, select the mytest service and click the Start Server button.
8. Run `lmwin` (if it's not already running). Checkout and checkin the license. To test that the server is actually counting the "f1" license, bring up another instance of `lmwin`. Check out "f1" from the first of the test applications and then try to check out "f1" from the second test application. The license server will deny the second license checkout request. Checkin "f1" from the first test application.
9. Finally, stop the mytest license server by clicking the Stop Server button in the Start/Stop/Reread tab of `lmtools`. Close `lmtools`, `genlic`, and the two instances of `lmwin`.

## 3.4 Evaluating the Counterfeit Resistant Option (CRO)

The Counterfeit Resistant Option uses public-key technology from Certicom to make the signatures on FEATURE/INCREMENT lines more difficult to counterfeit. It is a separately priced add-on to FLEX*lm* v7.1. However, during your evaluation of FLEX*lm* v7.1, you can edit one file, rebuild the demo SDK, and evaluate CRO.

### 3.4.1 Adding CRO to lm_code.h

1. In the `machind` directory, open `lm_code.h` in a text editor.
2. Find the definition of `LM_STRENGTH`. It is probably set to LM_STRENGTH_DEFAULT.
3. Redefine `LM_STRENGTH` to specify the length of your license signature.
   `#define LM_STRENGTH` *`strength`*
   where *`strength`* is one of:

| *`strength`* | **Bits** | **Number of hex characters** |
|---|---|---|
| LM_STRENGTH_113BIT | 113 | 58 |
| LM_STRENGTH_163BIT | 163 | 84 |
| LM_STRENGTH_239BIT | 239 | 120 |

   To demo FLEX*lm*, you do not need `CRO_KEY`s, but for a production FLEX*lm* kit with CRO enabled, you will need to enter `CRO_KEY`s into `lm_code.h` (see Section 6.1, “Editing lm_code.h”).
4. Save and close `lm_code.h`.

### 3.4.2 Rebuilding the FLEX*lm* Demo SDK

See Section 6.3, “Building with Microsoft Visual C++,” for instructions to rebuild your FLEX*lm* kit—however, do not change the vendor name from `demo` at this time.

### 3.4.3 Rekeying the Sample License File

1. Copy `licenseOrig.dat` into `license.dat`.
2. In the `i86_n3` directory, type:
   ```
   C:\> lmcrypt license.dat
   ```

3. Open `license.dat` in a text editor. Depending on the value of `LM_STRENGTH` that you chose (this example uses LM_STRENGTH_113BIT), you will see something like:

```
SERVER this_host ANY 27000
VENDOR demo
USE_SERVER
FEATURE f1 demo 1.0 permanent 4 SIGN="09F4 8B8B 8597 AB54 \
  C30E 21B4 E91B 1000 F6FE 67A1 86C8 B823 6D6C 6995 EC79"
```

---

**Note:** Each time you run CRO-enabled `lmcrypt` on a license file, you will generate a different set of signatures for the FEATURE/INCREMENT lines in that license file. The signature field cannot, therefore, be used to determine whether two INCREMENT lines are identical.

---

4. Close `license.dat`.

Now, if you'd like, you can run through the same evaluation of FLEX*lm* with CRO enabled—FLEX*lm* runs the same with CRO enabled, the only difference is in the way the signature is generated and authenticated.

You have completed the demo of FLEX*lm*'s pre-built programs on Windows. When you are ready to build your FLEX*lm* SDK with your production vendor keys, see Chapter 6, "Incorporating Production FLEXlm into a Windows Application." If you want to evaluate FLEX*lm* on UNIX, see Chapter 2, "Evaluating FLEXlm on UNIX." Remember to set `LM_STRENGTH` to an appropriate value before you build your production FLEX*lm* SDK.

Chapter 4

# Adding FLEX*lm* Function Calls Into Your Application

To incorporate FLEX*lm* licensing into your application, you will add FLEX*lm* function calls to your application program. This chapter will give you some introductory information about the Trivial and Simple APIs.

Chapter 5, “Incorporating Production FLEXlm into a UNIX Application,” and Chapter 6, “Incorporating Production FLEXlm into a Windows Application,” will give you an overview of building your production FLEX*lm* SDK containing your own vendor daemon and license generators, adding FLEX*lm* code to your application, building your application with the FLEX*lm* libraries, and testing your application with a new license file.

## 4.1 FLEX*lm* APIs

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

There are four APIs available to the developer:

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

GLOBEtrotter recommends using the Trivial API; if, however, the application requires FLEX*lm* functionality not provided in the Trivial API, use the Simple API. For complete flexibility, use the FLEXible API.

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

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

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

Most commonly, the FLEXible API is required:

- For asynchronous queuing, especially in GUI-based applications where queueing is required.
- To obtain a list of users of a given feature.
- For a vendor-defined hostid.

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

The Simple and Trivial APIs (as well as the Java API) are documented in this guide, while the FLEXible API is documented in detail in the *FLEXlm Reference Manual*. The Java API is not available for FLEX*lm* v7.1; the latest version is v7.0.

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.

## 4.2 FLEX*lm* Function Naming Conventions

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

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

## 4.3 Installation and Directory Naming for Java

The latest version of the Java API is for FLEX*lm* v7.0. No special installation of the FLEX*lm* class files is required. The classes are in the `java_01/flexlm` directory of the FLEX*lm* v7.0 main directory. Note that the FLEX*lm* class files must reside in a directory called `flexlm`, since they are in a Java package named flexlm.

If you are running Java applications, you must set your CLASSPATH environment variable to include a component reflecting the location of the FLEX*lm* classes. For example, if you installed the FLEX*lm* classes into */a/b/*lmgr/v7.0/java_01/flexlm, you would include the following component in your CLASSPATH:

```
"/a/b/lmgr/v7.0/java_01"
```

If you are setting up the FLEX*lm* class files for access by an http server, they must reside in a directory called flexlm, again, because the FLEX*lm* class files are in the flexlm package.

## 4.4 FLEX*lm* Example Applications

On both UNIX and Windows, the FLEX*lm* SDK contains the source for an example command-line client application program, lmclient, in the machind directory called lmclient.c. This is a small standalone licensed program using Trivial API macros and is a good place to start to learn how to integrate FLEX*lm* with your application.

For Windows systems, the SDK contains the source for lmwin, an example GUI application program, in machind\lmwin.c. lmwin uses Microsoft Visual C++ to build a slightly more complicated Trivial API example.

In the machind directory, you can also find lmsimple.c and lmflex.c which are the sources for application programs that behave like lmclient, but demonstrate function calls of the Simple and FLEXible APIs, respectively.

The lmcrypt, makekey, and genlic 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 signatures on all FLEX*lm*-supported platforms for all FLEX*lm* versions, allowing you to create licenses for any supported platform on any other supported platform.

The FLEX*lm* SDK also contains an examples directory at the top level of the hierarchy. The examples directory contains example programs, which have been put in the SDK to illustrate how to perform various operations with FLEX*lm*. These programs are **not supported** and GLOBEtrotter Software may not include them in future FLEX*lm* releases.

## 4.5 Client Heartbeats and License Server Failures

Your client application will need to communicate regularly with the license server via “heartbeat” calls to ensure that the license server is still running. Programming how the heartbeats occur and what action takes place when the license server is not running are the most important part of incorporating FLEX*lm* in an application. This is addressed in the following sections:

- Section 7.6, “HEARTBEAT()”
- Section 8.6, “lp_heartbeat()”
- *FLEXlm Reference Manual*, Section 3.21, “lc_heartbeat()”

## 4.6 License Policies

The Trivial and Simple APIs both require that you specify a license policy. A policy can be modified by ORing a list of optional modifiers. The following license policies are available:

- LM_RESTRICTIVE
- LM_QUEUE
- LM_FAILSAFE
- LM_LENIENT

### 4.6.1 LM_RESTRICTIVE

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

### 4.6.2 LM_QUEUE

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

### 4.6.3 LM_FAILSAFE

With this policy, the application will attempt a checkout, but no checkout failures of any kind will be reported to the calling application. This policy provides “optional” licensing to the user. If the user wants to use licensing, he

can, in which case the checkout will succeed. If the user doesn't want to use licensing, or if licensing is for some reason broken, applications will always continue to run.

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

### 4.6.4 LM_LENIENT

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

## 4.7 Policy Modifiers

These modifiers are binary ORed ("|") with the main policies, described above. The following policy modifiers are available:

- LM_MANUAL_HEARTBEAT
- LM_RETRY_RESTRICTIVE
- LM_CHECK_BADDATE
- LM_FLEXLOCK

### 4.7.1 LM_MANUAL_HEARTBEAT

If this policy modifier is not specified, heartbeats, via HEARTBEAT() or lp_heartbeat() are automatically sent every two minutes from the application to the server. On UNIX, SIGALRM is used to send the heartbeats.

For example:

```
LM_RESTRICTIVE | LM_MANUAL_HEARTBEAT
```

indicates that the main policy is LM_RESTRICTIVE, that no automatic heartbeats should be sent to the server, and the application will call HEARTBEAT() or lp_heartbeat() directly.

Most UNIX applications will require LM_MANUAL_HEARTBEAT, but some simple applications may prefer to have the heartbeats sent automatically. If your application does not send heartbeats to the license server, the application will not know if the license server has been shut down and

restarted. If the server is restarted, the old copies of the applications continue running and a (new) full complement of licenses becomes available, making license over-usage possible.

### 4.7.2 LM_RETRY_RESTRICTIVE

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

### 4.7.3 LM_CHECK_BADDATE

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

**SEE ALSO:**

- *FLEXlm Reference Manual*, Section 6.1.2, “Limited Functionality Demos”
- *FLEXlm Reference Manual*, Section 9.2.2, “ls_a_check_baddate”

### 4.7.4 LM_FLEXLOCK

If set, FLEX*lock* functionality is enabled.

**SEE ALSO:**

- Chapter 13, “FLEXlock and License Certificate Manager (Windows Only)

Chapter 5

# Incorporating Production FLEX*lm* into a UNIX Application

After you have become familiar with `lmgrd`, the sample vendor daemon (`demo`), `lmcrypt`, and the sample client program, you are ready to build your production FLEX*lm* SDK and to incorporate the FLEX*lm* client calls into your own software or into a copy of one of the example source files.

## 5.1 Editing lm_code.h

1. Open `lm_code.h` in a text editor.
2. Find the five VENDOR_KEY lines. Replace them with your production vendor key lines that you received from GLOBEtrotter.

   ```
   #define VENDOR_KEY1 0x0
   #define VENDOR_KEY2 0x0
   #define VENDOR_KEY3 0x0
   #define VENDOR_KEY4 0x0
   #define VENDOR_KEY5 0x0
   ```

3. If you are using CRO, add the two `CRO_KEY`s that you received from GLOBEtrotter. Otherwise, the `CRO_KEY`s are unused.
4. Change the `VENDOR_NAME` to your vendor name.
5. Change the `ENCRYPTION_SEED`s to four 32-bit numbers that you make up. Keep the `lm_code.h` file and the encryption seeds secret.

   `lm_code.h`, is used to build license generators (`lmcrypt`, `makekey`), the vendor daemon, and the `lm_new.o` file. You do not need the `lm_code.h` file to build your licensed applications once the `lm_new.o` file is built. So not all programmers in your company need access to `lm_code.h`, only the resulting `lm_new.o` file.

6. If you are using CRO, define `LM_STRENGTH` to specify the length of your license signature.

   `#define LM_STRENGTH` *`strength`*

   where *`strength`* is one of:

| *`strength`* | **Bits** | **Number of hex characters** |
|---|---|---|
| LM_STRENGTH_113BIT | 113 | 58 |
| LM_STRENGTH_163BIT | 163 | 84 |
| LM_STRENGTH_239BIT | 239 | 120 |
| LM_STRENGTH_DEFAULT | Not public-key; SIGN=12 chars. | |
| LM_STRENGTH_LICENSE_KEY | Uses FLEX*lm* license key. | |

7. Save and close `lm_code.h`.

## 5.2 Editing the makefile and lsvendor.c

1. Open *`platform`*`/makefile` in a text editor.
2. Change the vendor daemon name from `demo` to your own vendor daemon name. Change:

   ```
   DAEMON = demo
   ```

   to:

   ```
   DAEMON = v<vendor>
   ```

   where *`vendor`* is the same as `VENDOR_NAME` in `lm_code.h`. Otherwise, after `demo` is built by `make`, you'll need to rename it to *`vendor`*, where *`vendor`* is your vendor daemon name.
3. Optional—If your vendor daemon requires some customization (not normally needed), make modifications in `machind/lsvendor.c`.

## 5.3 Building the Production FLEX*lm* SDK

Type `make` in the *`platform`* directory to build your own vendor daemon, `lmcrypt`, and `lm_new.o` with your own encryption seeds.

## 5.4 Adding FLEX*lm* Code to Your Application

Modify your application code to call the license manager client routines. If possible, use the Trivial API. Add calls like the following to your application code:

```
#include "lmpolicy.h"
/*...*/
CHECKOUT(LM_RESTRICTIVE, "myfeature", "1.0", “license.dat”);
/*...*/
CHECKIN();
```

See `lmclient.c` or Chapter 7, “Trivial API,” for a full description of the parameters).

## 5.5 Building Your Application

Build your application as usual, adding the following files to your link (`ld` or `cc`) command:

| | **FLEX*lm* Object File** | **FLEX*lm* Client Libraries** |
|---|---|---|
| Standard | `lm_new.o` | `liblmgr.a` |
| Add for CRO | | `libcrvs.a`<br>`libsb.a` |

If you get a link error complaining about missing `l_n36_buf`, the problem is that you forgot to add `lm_new.o` to your link list.

## 5.6 Creating a License File

1. Open *`platform`*`/license.dat` in a text editor.
2. Modify the SERVER, VENDOR, and FEATURE lines in `license.dat` to include server information and information for the features that your application checks out. In place of a signature on each FEATURE line, enter `SIGN=0`. See Chapter 12, “The License File,” for information about the allowed fields and syntax of a license file.
3. Save `license.dat`.
4. Run `lmcrypt` to calculate the signatures and create a valid license file.

## 5.7 Testing Your Application

1. Put your application's license file where your application expects to find it.
2. If you have counted licenses, start a license server that uses the application's license file.
3. Start your application and perform a task that checks out one of your licenses.

Chapter 6

# Incorporating Production FLEX*lm* into a Windows Application

After you have become familiar with lmgrd, the sample vendor daemon (demo), and the sample client program, you are ready to build your production FLEX*lm* SDK and to incorporate the FLEX*lm* client calls into your own software or into a copy of one of the example source files.

On Windows, integration with your C compilers can take some planning. The makefiles have been designed to work with Microsoft Visual C++ v5.0 or greater. Other compilers may work, but will take additional time and effort to get a running product. If you're using any other C compiler or a different language, see Section 6.6, "Alternative: Using the FLEXlm Shared Library (DLL)."

In this section, `lmwin.exe` is used as a sample application. Its source is in `machind\lmwin.c`.

## 6.1 Editing lm_code.h

1. Change directories to `C:\Program Files\flexlm\v7.1\machind`.
2. Double-click `lm_code.h` to open it in a text editor or the Microsoft Visual C++ editor. Find the five VENDOR_KEY lines similar to the following:

```
#define VENDOR_KEY1 0x00000000
#define VENDOR_KEY2 0x00000000
#define VENDOR_KEY3 0x00000000
#define VENDOR_KEY4 0x00000000
#define VENDOR_KEY5 0x00000000
```

3. Replace these lines with your production vendor key lines that you received from GLOBEtrotter.

4. If you are using CRO, add the two `CRO_KEY`s that you received from GLOBEtrotter. Otherwise, the `CRO_KEY`s are unused.
5. Change the `VENDOR_NAME` to your vendor name.
6. Change the `ENCRYPTION_SEED`s to four 32-bit numbers that you make up. Keep the `lm_code.h` file and the encryption seeds secret.

   `lm_code.h`, is used to build license generators (`lmcrypt.exe`, `makekey.exe`), the vendor daemon, and the `lm_new.obj` file. You do not need the `lm_code.h` file to build your licensed applications once the `lm_new.obj` file is built. So not all programmers in your company need access to `lm_code.h`, only the resulting `lm_new.obj` file.
7. If you are using CRO, define `LM_STRENGTH` to specify the length of your license signature.

   `#define LM_STRENGTH` *`strength`*

   where *`strength`* is one of:

| *`strength`* | **Bits** | **Number of hex characters** |
|---|---|---|
| LM_STRENGTH_113BIT | 113 | 58 |
| LM_STRENGTH_163BIT | 163 | 84 |
| LM_STRENGTH_239BIT | 239 | 120 |
| LM_STRENGTH_DEFAULT | Not public-key; SIGN=12 chars. | |
| LM_STRENGTH_LICENSE_KEY | Uses FLEX*lm* license key. | |

8. Save and close `lm_code.h`.

## 6.2 Editing the makefile and lsvendor.c

1. Open `i86_n3\makefile` in a text editor.
2. Change the vendor daemon name from `demo` to your own vendor daemon name. Change:

   ```
   DAEMON = demo
   ```

   to:

   ```
   DAEMON = vendor
   ```

where *vendor* is the same as VENDOR_NAME in lm_code.h. Otherwise, after demo is built by make, you'll need to rename it to *vendor*, where *vendor* is your vendor daemon name.

3. Optional—If your vendor daemon requires some customization (not normally needed), make modifications in machind/lsvendor.c.

## 6.3 Building with Microsoft Visual C++

### 6.3.1 Setting Up the C Development Environment

Make sure that you have your Microsoft Visual C++ development environment correctly configured. If you don't have this compiler, you'll have to use the FLEXlm DLL (see Section 6.6.1, "Define FLEXLM_DLL").

1. Check whether the MSVCDIR environment variable has been set to the main Microsoft Visual C++ folder and that your PATH variable contains the path to the Microsoft Visual C++ bin folder. For example, on Windows 95/98 you might add the following lines to your autoexec.bat file and on Windows NT you might add the following settings to your system environment via the Control Panel:

   ```
   SET MSVCDIR=c:\progra~1\micros~1\vc98
   PATH=%PATH%;c:\progra~1\micros~1\vc98\bin
   ```

2. If you are running on Windows 95/98, open a command window. Right-click in the upper left corner of the window and select Properties. Click the Memory tab. In the Initial Environment choice list, select 4096. Apply the changes. Close the command window.
3. Reboot your computer.
4. Open a command window.
5. To set your environment for using Microsoft Visual C++, type:

   ```
   vcvars32
   ```

### 6.3.2 Building the Production FLEX*lm* SDK

To build your FLEX*lm* SDK, in the command window, change to the C:\Program Files\flexlm\v7.1\i86_n3 folder and type:

```
build
```

A list of files being built is displayed in the command window. This builds lm_new.obj, which contains security information from lm_code.h and is used to build your application, your vendor daemon, as well as the sample applications, lmclient.exe and lmwin.exe. build.bat itself only calls

nmake. If you're familiar with nmake, you can run this directly. nmake uses the i86_n3\makefile to build all files which are out of date. If you've just edited lm_code.h, it will rebuild everything except your application.

If you've done this correctly, you should be able to generate a license file and use it with the sample applications, lmclient.exe or lmwin.exe.

### 6.3.3 Adding FLEX*lm* Calls to Your Application

Modify your application code to call the license manager client routines. Add calls like the following to your application code:

```
#include "lmpolicy.h"
/*...*/
if (CHECKOUT(LM_RESTRICTIVE, "myfeature", "1.0", "license.dat"))
{
        PERROR("Checkout failed");
        exit(1);
}
/*...*/
CHECKIN();
```

See lmclient.c or Chapter 7, "Trivial API," for a full description of the parameters.

### 6.3.4 Compiling Your Object File

#### ADDING THE FLEX*LM* INCLUDE DIRECTORY

Add "C:\Program files\FLEXlm\v7.1\machind" to your include path. If you use CL on the command line or in a batch file, add /I "C:\Program Files\FLEXlm\v7.1\machind" to your CL command line.

If you use the IDE (GUI development environment), add the include directory in Tools→Options→Directories. Pick include files from the choice list and add the full path to the machind directory.

#### SELECTING /MT OR /MD (MULTI-THREADED STATIC OR SHARED LIBRARIES)

FLEX*lm* requires multi-threaded libraries. The CL compiler /MT switch indicates the static multi-threaded C library (preferred) and the /MD switch indicates the shared multi-threaded C library. One or the other must be used when compiling. If you indicate /MT, then you must link with libcmt.lib and lmgr.lib. If you indicate /MD, then you must link with msvcrt.lib and lmgr_md.lib.

If you use the command line, make sure /MT (or /MD) is included in the CL command line.

If you use the IDE, in Project→Settings, pick the C/C++ tab. Then make sure that /MT (or /MD) is specified in the switches listed in the bottom window. In more current versions of Microsoft Visual C++, there's a choice list for multi-threaded.

#### COMPILING IN THE IDE

1. Make an empty project:

   File → New

   Project name: lmwin

   Select Win32 Application.

   Click OK.
2. Add source file.

   Project → Add to Project → Files...

   C:\Program Files\FLEXlm\v7.1\machind\lmwin.c
3. Add the include directory.

   Tools → Options → Directories → Include Directories

   C:\Program Files\FLEXlm\v7.1\machind
4. Specify multi-threaded.

   (Microsoft Visual C++ v6+): Specify Multi-threaded

   Otherwise: Change /ML or /MT to /MD in the options window.
5. Compile.

   File → Recent Files → lmwin.c

   Build → Compile lmwin.c

#### COMPILING ON THE COMMAND LINE

```
c:>cl /nologo /c /O1 /I../machind /MT  ../machind/lmwin.c
```

### 6.3.5 Linking the Client Application

#### IDE SETTINGS

Change the following settings:

1. Library path

   Tools → Options → Directories

   Pick the Library Files choice list, and add:

   C:\Program Files\FLEXlm\v7.1\i86_n3
2. Libraries and options

   Project → Settings → Link

Make sure that the `/NODEFAULTLIB` switch and one of the sets of FLEX*lm* and Microsoft Visual C++ libraries listed below are included in the list of object/library modules.

| | **FLEX*lm* Object File** | **FLEX*lm* Client Libraries** | **MSVC++ Libraries** |
|---|---|---|---|
| Standard | `lm_new.obj` | `lmgr.lib` | `libcmt.lib` (/MT)<br>`oldnames.lib`<br>`kernel32.lib`<br>`user32.lib`<br>`netapi32.lib`<br>`advapi32.lib`<br>`gdi32.lib`<br>`comdlg32.lib`<br>`comctl32.lib`<br>`wsock32.lib` |
| Add for CRO | | `libcrvs.lib`<br>`libsb.lib` | |
| Add for static FLEX*lock* | | `flock.lib` | |
| For /MD | Replace `lm_new.obj` with `lm_new_md.obj` | Replace Standard `lmgr.lib` with `lmgr_md.lib` | Replace Standard `libcmt.lib` with `msvcrt.lib` (/MD) |
| For CRO /MD | | Add to Standard:<br>`libcrvs_md.lib`<br>`libsb_md.lib` | Replace Standard `libcmt.lib` with `msvcrt.lib` (/MD) |

3. Link

   Build → Build lmwin.exe

#### LINKING ON THE COMMAND LINE

On the command line, the `lmwin` link line for a non-CRO, /MT example looks like:

```
C:> LINK /nologo /NODEFAULTLIB /out:lmwin.exe lmwin.obj lm_new.obj
lmwin.res lmgr.lib oldnames.lib libcmt.lib kernel32.lib user32.lib
netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib
wsock32.lib
```

If you get a link error complaining about missing `l_n36_buf`, the problem is that you need to add `lm_new.obj` to your link list.

## 6.4 Creating a License File

1. Open `i86_n3\license.dat` in a text editor.
2. Modify the SERVER, VENDOR, and FEATURE lines in `license.dat` to include server information and information for the features that your application checks out. In place of a signature on each FEATURE line, enter `SIGN=0`. See Chapter 12, “The License File,” for information about the allowed fields and syntax of a license file.
3. Save `license.dat`.
4. Run `lmcrypt.exe` to calculate the signatures and create a valid license file.

## 6.5 Testing Your Application

1. Put your application’s license file where your application expects to find it.
2. If you have counted licenses, start a license server that uses the application’s license file.
3. Start your application and perform a task that checks out one of your licenses.

## 6.6 Alternative: Using the FLEX*lm* Shared Library (DLL)

If your application does not use Microsoft Visual C++, the best alternative is to use Microsoft Visual C++ to make your own DLL, linking in the FLEX*lm* static library.

If you cannot make your own DLL, you can use the FLEX*lm* shared library (FLEXLM_DLL) for supporting alternate languages, like Visual Basic, or a different C compiler, like Borland. Any compiler that can use Windows DLLs can use the FLEXLM_DLL.

**Caution:** Use of the FLEX*lm* DLL on Windows remains strongly discouraged. However, for companies that choose to use the DLL, the security provided by `lm_new.obj` is now available with the DLL, and is therefore better at preventing counterfeiting than previous version. Note that this requires using the FLEXible API (lc_*xxx*()) and linking with a single `.obj` file.

Nearly all the steps for using the shared library are same as in Section 6, "Incorporating Production FLEXlm into a Windows Application." The following sections describe the differences.

### 6.6.1 Define FLEXLM_DLL

If you're using the Trivial or Simple API, when compiling your source code, you must make sure the FLEXLM_DLL is defined. You can do this by:

| | |
|---|---|
| Command line | `cl /DFLEXLM_DLL` |
| In the source code | Add `#define FLEXLM_DLL` before any FLEX*lm* headers are included. |
| In the IDE | Project→Settings→C/C++→<br>Preprocessor Definitions<br>Add FLEXLM_DLL |

To take advantage of the increased DLL security in v7.1, use the FLEXible API (see the *FLEXlm Reference Manual*). The only change from previous versions is that you must now add `lc_new_job_arg2` as the second argument to lc_new_job(), as in `machind/lmflex.c`:

```
if (lc_new_job(0, lc_new_job_arg2, &code, &lm_job))
{
     lc_perror(lm_job, "lc_new_job failed");
     exit(lc_get_errno(lm_job));
}
```

You can use `lc_new_job_arg2` in both DLL and non-DLL code. Then, you must link `lm_new.obj` into your application. If you cannot link an object file into your application, then you cannot take advantage of this security feature, and you must call lc_init(), as in the past.

### 6.6.2 Link with lmgr327b.lib

Link with `lmgr327b.lib` instead of `lmgr.lib`, as outlined in Section 6.3.5, “Linking the Client Application.”

### 6.6.3 Ship lmgr327b.dll

Ship `lmgr327b.dll` with your application. This must be in the same directory as your application or in the user’s `PATH`.

Chapter 7

# Trivial API

## 7.1 Overview of Trivial API Calls

The Trivial API consists of macros that call the Simple API. What makes this API trivial is the simplified arguments to the macros. The only required header file is `lmpolicy.h`, and no other macros or function calls are needed.

| | |
|---|---|
| CHECKIN() | Releases a license and frees all FLEX*lm* memory. |
| CHECKOUT() | Acquires a license. |
| ERRSTRING() | Returns a string describing the most recent error. |
| HEARTBEAT() | Sends a heartbeat to the server. |
| PERROR() | Presents current error message to user. |
| PWARN() | Presents current warning message to user. |
| WARNING() | Returns a string describing the most recent warning. |

To use the Trivial API, simply include `lmpolicy.h` at the top of your source file. With the Trivial API, only one feature can be checked out at a time from a single process.

Where possible, this is the preferred FLEX*lm* API to use in your application.

**Note:** You cannot mix Trivial API calls with either Simple or FLEXible API calls.

## 7.2 Trivial API Example Program

The following is a complete example of the FLEX*lm* calls required in an application which uses the Trivial API:

```
#include "lmpolicy.h"
       /*...*/
       if (CHECKOUT(LM_RESTRICTIVE, "myfeature", "1.0", "license.dat"))
       {
               PERROR("Checkout failed");
               exit(-1);
       }
/*
 *     Checkout succeeded. Actual application code here
 */
       /*...*/

       CHECKIN();       /* Done with "myfeature", check it back in. */
```

## 7.3 CHECKIN()

**SYNTAX**

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

**DESCRIPTION**

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

## 7.4 CHECKOUT()

**SYNTAX**

*status* = CHECKOUT(*policy*, *feature*, *version*, *license_file_path*)

**DESCRIPTION**

Acquires a license for a feature.

**PARAMETERS**

| | |
|---|---|
| (int) *policy* | See Section 4.6, “License Policies.” Example: LM_RESTRICTIVE. |
| (char *) *feature* | The feature name to check out. |

| | |
|---|---|
| (char *) *version* | The version of the feature to check out. This is a string in floating-point format (e.g., 12345.123). If the license in the license file has the same version number or a higher version number, the checkout will succeed. GLOBEtrotter recommends that this version number be a license version level and *not* the application's version number. This version number should only be changed when you want old licenses to no longer work with a new version of the software. |
| (char *) *license_file_path* | The default location for the license file. If 0, this argument is unused. |

#### RETURN

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

The application will look in the following places for the license file:

- Location specified by the *VENDOR*_LICENSE_FILE and/or LM_LICENSE_FILE environment variable or registry settings.
- *license_file_path* specified here

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

#### SEE ALSO

- Section 4.6, "License Policies"
- Section 4.7, "Policy Modifiers"

## 7.5 ERRSTRING()

**SYNTAX**

```
string = ERRSTRING()
```

**DESCRIPTION**

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

**RETURN**

| | |
|---|---|
| `(char *) string` | An explanatory string. |

## 7.6 HEARTBEAT()

**SYNTAX**

```
status = HEARTBEAT()
```

**DESCRIPTION**

Exchanges heartbeat messages with the server. If the server goes down and later comes back up, HEARTBEAT() will automatically reconnect and check out the license. On failure, returns the number of failed attempts to reconnect to the server. On failure, applications should at a minimum notify the user of the failure. For a restrictive policy, applications may exit after a certain number of failures. In addition, applications may want to exit if reconnections succeed more than three or four times in a relatively short period (e.g., ten minutes), which may indicate a user restarting the license server in an attempt to acquire extra licenses. Do not call CHECKOUT() on failure from HEARBEAT()—this is not necessary and will cause problems if attempted.

HEARTBEAT() should not be called unless LM_MANUAL_HEARTBEAT is set in the CHECKOUT() call. If LM_MANUAL_HEARTBEAT is not set, then HEARTBEAT() is called automatically.

**RETURN**

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

**SEE ALSO**

- Section 4.7.1, “LM_MANUAL_HEARTBEAT”

## 7.7 PERROR()

**SYNTAX**

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

**DESCRIPTION**

Presents *string* and a description of the most recent error to the user. On Windows this appears in a dialog; on other systems, it prints to stderr.

**PARAMETERS**

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

## 7.8 PWARN()

**SYNTAX**

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

**DESCRIPTION**

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

**PARAMETERS**

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

## 7.9 WARNING()

**SYNTAX**

```
string = WARNING()
```

**DESCRIPTION**

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

### RETURN

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

Chapter 8

# Simple API

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

This API requires that you include the `lmpolicy.h` header file.

## 8.1 Simple API Library Routines

| | |
|---|---|
| lp_checkin() | Releases a license and frees all FLEX*lm* memory. |
| lp_checkout() | Acquires a license. |
| lp_errstring() | Returns a string describing the most recent error. |
| lp_heartbeat() | Sends a heartbeat to the server. |
| lp_perror() | Presents current error message to user. |
| lp_pwarn() | Presents current warning message to user. |
| lp_warning() | Returns a string describing the most recent warning. |

**Note:** You cannot mix Simple API calls with either Trivial or FLEXible API calls.

## 8.2 Simple API Example Program

The following is a complete example of the FLEX*lm* calls required in an application that uses the Simple API. The primary differences between this and the Trivial API example are:

- The setup is a bit more complicated.
- The args to lp_checkout() are more complicated than to CHECKOUT().
- You can check out more than one feature simultaneously, or more than one license of a feature (although neither of these are illustrated in the example).

```
#include "lmpolicy.h"
LP_HANDLE *lp_handle;
       /*...*/
       if (lp_checkout(LPCODE, LM_RESTRICTIVE|LM_MANUAL_HEARTBEAT,
                      "myfeature","1.0", 1, "license.dat", &lp_handle))
       {
              fprintf(stderr, "Checkout failed: %s",
                                    lp_errstring(lp_handle));
              exit(-1);
       }
/*
 *     Checkout succeeded. Actual application code here
 */
       /*...*/
/*
 *     Done with "myfeature", check it back in.
 */
       lp_checkin(lp_handle);
```

## 8.3 lp_checkin()

**SYNTAX**

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

**DESCRIPTION**

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

**PARAMETER**

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

## 8.4 lp_checkout()

**SYNTAX**

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

**DESCRIPTION**

Acquires a license for a feature.

**PARAMETERS**

| | |
|---|---|
| `(LPCODE_HANDLE *) LPCODE` | From the `lmpolicy.h` include file. Use the literal `LPCODE`. |
| `(int) policy` | See Section 4.6, “License Policies.” Example: LM_RESTRICTIVE. |
| `(char *) feature` | The desired feature name to check out. |

| | |
|---|---|
| `(char *) version` | The version of the feature to check out. This is a string in floating-point format (e.g., 12345.123). If the license in the license file has the same version number or a higher version number, the checkout will succeed.<br>GLOBEtrotter recommends that this version number be a license version level and *not* the application's version number. This version number should only be changed when you want old licenses to no longer work with a new version of the software |
| `(int) num_lic` | The number of licenses to check out. Usually this number is 1. |
| `(char *) license_file_path` | The expected location of the license file. The application will look in the following places for the license file: the location specified by the *VENDOR*`_LICENSE_FILE` and/or `LM_LICENSE_FILE` environment and/or registry variables, this default location, then the FLEX*lm* default (`/usr/local/flexlm/licenses/license.dat` for UNIX or `C:\flexlm\license.dat` for PCs). It is highly recommended that the expected location of the license file be set to a place in your product's installation hierarchy. The application may need to do some work to determine the exact path at runtime.<br>If 0, this argument is unused. |

| | |
|---|---|
| `pointer to (LP_HANDLE *) lp_handle` | This is the return handle, and is used for subsequent calls that apply to this checkout, e.g., lp_checkin(), lp_errstring(), etc. If lp_checkout() is called more than once, separate `lp_handle` variables must be declared and used, and the corresponding handle must be used with the other lp_*xxx*() (Simple API) calls. |

Upon success, the path to the license file used is set in *VENDOR*`_LICENSE_FILE` in the registry on Windows (`\HKEY_LOCAL_MACHINE\SOFTWARE\FLEXlm License Manager`) and `$HOME/.flexlmrc` on UNIX (v7+).

**RETURN**

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

To check out two features:

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

## 8.5 lp_errstring()

**SYNTAX**

*`string`* `= lp_errstring(lp_handle)`

**DESCRIPTION**

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

**PARAMETER**

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

**RETURN**

| | |
|---|---|
| `(char *) string` | Error description. |

## 8.6 lp_heartbeat()

**SYNTAX**

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

**DESCRIPTION**

Exchanges heartbeat messages with the server. If the server goes down and later comes back up, lp_heartbeat() will automatically reconnect and check the license out. On failure, returns the number of failed attempts to reconnect to the server. On failure, applications should at a minimum notify the user of the failure. For a restrictive policy, applications may exit after a certain number of failures. In addition, applications may want to exit if reconnections succeed more than three or four times in a relatively short period (e.g. ten minutes), which may indicate a user restarting the license server in an attempt to acquire extra licenses.

lp_heartbeat() should not be called unless LM_MANUAL_HEARTBEAT is set in the lp_checkout() call. If LM_MANUAL_HEARTBEAT is not set, then lp_heartbeat() is called automatically by the FLEX*lm* client library.

**PARAMETERS**

| | |
|---|---|
| `(LP_HANDLE *) lp_handle` | The handle from the lp_checkout() call. |
| `(int *) num_reconnects` | The number of reconnections in the last *num_minutes* minutes. This value is returned. If set to 0, no value is returned. |

| | |
|---|---|
| `(int) num_minutes` | Number of minutes for `num_reconnects`. If 0, `num_reconnects` is not returned. |

#### RETURN

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

#### SEE ALSO

- Section 4.7.1, “LM_MANUAL_HEARTBEAT”

## 8.7 lp_perror()

#### SYNTAX

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

#### DESCRIPTION

Presents `string` and a description of the most recent error to the user. On Windows this appears in a dialog; on other systems, it prints to stderr.

#### PARAMETERS

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

## 8.8 lp_pwarn()

**SYNTAX**

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

**DESCRIPTION**

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

**PARAMETERS**

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

## 8.9 lp_warning()

**SYNTAX**

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

**DESCRIPTION**

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

**PARAMETERS**

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

**RETURN**

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

Chapter 9

# Java API

The Java implementation of the FLEX*lm* client library allows applets and applications written in Java to use FLEX*lm* licensing. The Java API is similar to the Simple API. The Java API is not available for FLEX*lm* v7.1; the latest FLEX*lm* version that supports the Java API is v7.0.

---

**Note:** All FLEX*lm* class files are in the flexlm package. Any of your Java classes which invoke FLEX*lm* methods must use:

```
import flexlm.*;
```

---

## 9.1 license Class

The license class has the following constructor and instance methods:

| | |
|---|---|
| license () | license class constructor. |
| checkin() | Releases a license and frees all FLEX*lm* memory. |
| checkout() | Acquires a license. |
| get_errstring() | Returns a string describing the most recent error. |
| get_major_errcode() | Returns the FLEX*lm* major error code. |
| get_minor_errcode() | Returns the FLEX*lm* minor error code. |
| getDisplay() | Allows display name to sent to the license server in connection with a license checkout and checkin. |

| | |
|---|---|
| getHostname() | Overrides host name sent to the license server in connection with a license checkout and checkin. |
| getUsername() | Allows user name to sent to the license server in connection with a license checkout and checkin. |
| heartbeat() | Sends a heartbeat to the license server. |
| ok() | Checks whether major error code indicates a checkout failure. |
| warning() | Returns a string describing the most recent warning. |

### 9.1.1 license ()

**SYNTAX**

```
public license(vendor, d1, d2, k1, k2, k3, k4)
```

**DESCRIPTION**

Constructor that creates an instance of the license class, using the specified vendor name, encryption seeds, and vendor keys.

**PARAMETERS**

| | |
|---|---|
| `String vendor` | Vendor daemon name |
| `int d1` | XOR of *VENDOR_KEY5* and *ENCRYPTION_SEED1* |
| `int d2` | XOR of *VENDOR_KEY5* and *ENCRYPTION_SEED2* |
| `int k1` | *VENDOR_KEY1* |
| `int k2` | *VENDOR_KEY2* |
| `int k3` | *VENDOR_KEY3* |
| `int k4` | *VENDOR_KEY4* |

**Note:** Vendor keys are issued to you by GLOBEtrotter. The encryption seeds are 32-bit numbers you make up and keep secret. See the FLEX*lm* QuickStart for more information.

**Note:** It is wise not to store your encryption seeds in variables, but rather to mention them only in an expression where they are XORed with *VENDOR_KEY5*. Given that expressions are evaluated at compile time, this makes it harder for somebody to discover your encryption seeds by decompiling your classes.

### 9.1.2 checkin()

**SYNTAX**

```
public void checkin()
```

**DESCRIPTION**

Checks in (releases) the licenses acquired with checkout().

**PARAMETERS**

None.

**RETURN**

None.

### 9.1.3 checkout()

**SYNTAX**

```
public int checkout(policy, feature, version,
                    num_lic, license_file_path);
```

**DESCRIPTION**

Acquires a license for a feature.

**PARAMETERS**

| | |
|---|---|
| `int` *`policy`* | See Section 4.6, “License Policies,” for a list of valid policies. Example: LM.RESTRICTIVE. |
| `String` *`feature`* | The feature name to be checked out. |

| | |
|---|---|
| `String` *`version`* | The version of the feature to check out. This is a string in floating-point format (e.g., 12345.123). If the license in the license file has the same version number, or a higher version number, the checkout will succeed. GLOBEtrotter recommends that this version number be a license version level and not the application's version number. This version number should only be changed when you want old licenses to no longer work with a new version of the software. |
| `int` *`num_lic`* | The number of licenses to check out. Usually this is number is 1. |
| `String` *`license_file_path`* | The location of the license file. This may be a file name, a `[`*`port`*`]@`*`host`* specification, or a colon-separated list of file names and/or `[`*`port`*`]@`*`host`* specifications. Note that there is no Java equivalent for environment variables, so `LM_LICENSE_FILE` and *`VENDOR`*`_LICENSE_FILE` are not used in the Java version. |

**RETURN**

| | |
|---|---|
| `int` | 0 if successful; otherwise, use get_major_errcode() to get the FLEX*lm* major error number. |

### 9.1.4 get_errstring()

**SYNTAX**

```
public String get_errstring()
```

**DESCRIPTION**

Get the string corresponding to the most recent FLEX*lm* major and minor error codes. It is appropriate to use this string in an error message issued to the user.

**RETURN**

| | |
|---|---|
| `String` | String describing the most recent FLEX*lm* error. The returned string contains a text string describing the error, a major numeric error code corresponding to the error, and a minor numeric error code which indicates to GLOBEtrotter technical support exactly where in the source code the error occurred. |

### 9.1.5 get_major_errcode()

**SYNTAX**

```
public int get_major_errcode()
```

**DESCRIPTION**

Get major FLEX*lm* error code. The major and minor error codes appear in the string returned by the get_errstring() method. These codes correspond to the LM.* error codes (statically referenced in the LM class).

**RETURN**

| | |
|---|---|
| `int` | Major error code. |

### 9.1.6 get_minor_errcode()

**SYNTAX**

```
public int get_minor_errcode()
```

**DESCRIPTION**

Get minor FLEX*lm* error code. The major and minor error codes appear in the string returned by the get_errstring() method. These codes are meaningful only to GLOBEtrotter.

**RETURN**

| | |
|---|---|
| `int` | Minor error code. |

### 9.1.7 getDisplay()

**SYNTAX**

```
public String getDisplay()
```

**DESCRIPTION**

For overriding the display sent to the license server and used in the license server's log files in connection with checkouts and checkins associated with this instance.

The FLEX*lm* display name is by default a constant in FLEX*lm* for Java. It is "" (a null string). This value may be overridden by subclassing the FLEX*lm* license class and overriding this method.

**RETURN**

| | |
|---|---|
| `String` | New display value. |

### 9.1.8 getHostname()

**SYNTAX**

```
public String getHostname()
```

**DESCRIPTION**

For overriding the host name sent to the license server and used in the license server's log files in connection with checkouts and checkins associated with this instance.

The host name is determined at runtime via the getLocalHost() method in the java.net.InetAddress class. This value may be overridden by subclassing the FLEX*lm* license class and overriding this method.

**RETURN**

| | |
|---|---|
| `String` | New host name value. |

### 9.1.9 getUsername()

**SYNTAX**

```
public String getUsername()
```

**DESCRIPTION**

For overriding the user name sent to the license server and used in the license server's log files in connection with checkouts and checkins associated with this instance.

The FLEX*lm* user name is by default a constant in FLEX*lm* for Java. It is "JavaUser." This value may be overridden by subclassing the FLEX*lm* license class and overriding this method.

**RETURN**

| | |
|---|---|
| `String` | New user name value. |

### 9.1.10 heartbeat()

**SYNTAX**

```
public int heartbeat(num_reconnects, num_minutes)
```

**DESCRIPTION**

Exchanges heartbeat messages with the server. If the server goes down and later comes back up, heartbeat() will automatically reconnect and check the license out. On failure, returns the number of failed attempts to reconnect to the server. On failure, applications should at least notify the user of the failure. For a restrictive policy, applications may exit after a certain number of failures. In addition, applications may want to exit if reconnections succeed more than three or four times in a relatively short period (e.g., ten minutes), which may indicate a user restarting the license server in an attempt to acquire extra licenses.

---

**Note:** There is no automatic heartbeat in FLEX*lm* for Java; the heartbeat() method *must* be called periodically by your Java code.

---

#### PARAMETERS

| | |
|---|---|
| `int[] num_reconnects` | An array of length 1, or null. If non-null, the number of reconnects in the last `num_minutes` minutes is returned in the 0'th element of this array. |
| `int num_minutes` | Number of minutes over which to count reconnections attempts. |

#### RETURN

| | |
|---|---|
| `int` | 0 if successful; otherwise the number of failed reconnect attempts since the last successful heartbeat response was received. Note that if the license policy is LM.RETRY_RESTRICTIVE, the application exits after five unsuccessful heartbeat attempts. |

### 9.1.11 ok()

#### SYNTAX

```
public boolean ok()
```

#### DESCRIPTION

Checks whether major error code indicates a checkout failure. This can be used as an alternative to checking the integer values returned by other methods.

#### RETURN

| | |
|---|---|
| `boolean` | True if the major error code does not indicate a checkout failure; false if the major error code indicates a checkout failure. |

### 9.1.12 warning()

**SYNTAX**

```
public String warning()
```

**DESCRIPTION**

Returns a description of the most recent FLEX*lm* error, or null if no error has occurred. Similar to get_errstring(), but useful when the policy is LM.LENIENT or LM.FAILSAFE, and the result of get_errstring() is null. It is appropriate to use this string in an error message issued to the user.

**RETURN**

| | |
|---|---|
| `String` | Description of most recent FLEX*lm* error. |

## 9.2 LM Class

All constants in the LM class are declared `public static final int`.

### 9.2.1 License Policies

These values are used in the *`policy`* argument in the checkout() method. The meaning of these policies can be found in Section 4.6, “License Policies.”

The four policies available in the Java API are:

- LM.RESTRICTIVE
- LM.QUEUE
- LM.LENIENT
- LM.FAILSAFE

The one policy modifier, which can be ORed with the policy, is LM.RETRY_RESTRICTIVE. Note that the Java license policies substitute a “.” in place of the leading “_” that is found in the policies of the Trivial and Simple APIs.

---

**Note:** The LM_MANUAL_HEARTBEAT policy modifier from non-Java FLEX*lm* is not implemented in FLEX*lm* for Java; all heartbeats are manual in FLEX*lm* for Java.

---

### 9.2.2 Error Codes

See Appendix D of the *FLEXlm Reference Manual* for a list of FLEX*lm* error codes.

## 9.3 Java and Security

There are special security considerations for companies using FLEX*lm* for Java, since Java applications can be relatively easily decompiled. For this reason we recommend that licenses and license servers for Java applications be different than non-Java applications. Otherwise, compromising security through a Java application would imply compromised security for non-Java applications.

Chapter 10

# License Servers

A license server comprises a license manager daemon (`lmgrd`) process and one or more vendor daemon processes. License server refers to these processes, not the computer on which they run.

If you ship counted licenses, your customers will need to run a license server. Therefore, you will ship them a copy of the standard `lmgrd` available from GLOBEtrotter and your customized vendor daemon. This chapter also discusses the machine(s) on which your customers will be running license server(s).

## 10.1 lmgrd

The purpose of `lmgrd` is to:

- Start and maintain all the vendor daemons listed in the VENDOR lines of the license file(s)
- Refer application checkout (or other) requests to the correct vendor daemon
- Establish and maintain communications between redundant server machines

`lmgrd` is a standard component of FLEX*lm* that neither requires nor allows vendor customization. The license manager daemon allows the license file location and a few other parameters to be set by the end user. These options are set by command-line arguments when starting `lmgrd`.

### 10.1.1 Upgrading lmgrd

Please recommend to your customers that they always run `lmgrd` built with the newest version of FLEX*lm*, even if your vendor daemon and client application do not contain the newest version. The current version of `lmgrd` is available from http://www.globetrotter.com/lmgrd.htm.

### 10.1.2 Starting lmgrd

The most common command line for `lmgrd` is:

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

If *`license_file_list`* is more than once license file, it needs to be a list separated by colons on UNIX or semi-colons on Windows. If a directory is specified, `*.lic` in that directory is used.

A complete description of `lmgrd` options is contained in the *FLEXlm Reference Manual*.

### 10.1.3 Switching the Debug Log File on UNIX

The FLEX*lm* daemons write an ASCII debug log file on stdout. There are several processes in a parent-child hierarchy which are sharing the same file pointer, so this log file cannot be changed after the vendor daemons have been started, since each process has a copy of the current offset, etc.

There is another way to switch the log file output data; however, this involves piping the stdout of `lmgrd` to a shell script that appends each line to a file. This is done as follows:

Instead of the “normal” startup:

```
% lmgrd > LOG
```

Start `lmgrd` this way:

```
% lmgrd -z | sh -c 'while read line; do echo "$line" >> LOG ; done'
```

With this startup method, the output file `LOG` can be renamed and a new log file will be created. You could even make `LOG` a symbolic link and change the value of the link to “switch” the log file. The messages that are written by the daemons are described in the *FLEXlm End Users Guide.*

## 10.2 Configuring Your Vendor Daemon

You have configured and built a `demo` vendor daemon if you followed the QuickStart installation procedures and Chapter 2, “Evaluating FLEXlm on UNIX.” To build your own vendor daemon, you must supply the following information:

- Your encryption seeds
- Your FLEX*lm* production vendor keys
- The name of your vendor daemon — `VENDOR_NAME` in `lm_code.h`

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

The makefile will create `lm_new.o` on UNIX (`lm_new.obj` on Windows) and then build your vendor daemon, `lmclient` (the sample client application), and the license generators (`lmcrypt` and `makekey`). One of the functions of `lm_new.o` is to remove the vendor name and encryption seeds from the executables; they are constructed at runtime.

### 10.2.1 Building Your Vendor Daemon—UNIX Systems

To build your vendor daemon, edit `lm_code.h` (and `lsvendor.c`, if needed). Assuming that the FLEX*lm* SDK is in `/usr/gsi/flexlm/v7.1)`, type:

```
% cd /usr/gsi/flexlm/v7.1/platform
% make
```

### 10.2.2 Building Your Vendor Daemon—Windows Systems

To build your vendor daemon, edit `lm_code.h` (and `lsvendor.c`, if needed). Assuming that the FLEX*lm* SDK is in `C:\Program Files\flexlm\v7.1\`), type:

```
C:> cd \Program Files\flexlm\v7.1\i86_n3
C:> build
```

## 10.3 Upgrading Your Vendor Daemon

Improvements are made in each release of FLEX*lm*, therefore, we recommend distributing an upgraded vendor daemon to your customers soon after each version of FLEX*lm* is released. Upgrading your vendor daemon is a simple process—you need only rebuild your FLEX*lm* SDK, you do not have to rebuild your application.

To find the current version of your vendor daemon, vendor, you can type:

```
vendor -v
```

or

```
lmutil lmver vendor
```

These steps assume that the only change you are making is to upgrade your vendor daemon. To rebuild your vendor daemon:

1. Download and install the latest version of FLEX*lm* from GLOBEtrotter.
2. Open `lm_code.h` and edit it to match your previous `lm_code.h` settings: Replace the `VENDOR_KEY`s with the production vendor key lines that you received from GLOBEtrotter, enter `ENCRYPTION_SEED`s (four 32-bit numbers that you make up), and change the `VENDOR_NAME` to your vendor

name. If you are using CRO, add the two `CRO_KEY`s that you received from GLOBEtrotter and define `LM_STRENGTH` to specify the length of your license signature. Remember to keep the `lm_code.h` file and the encryption seeds secret.

3. Rebuild your FLEX*lm* SDK with the edited `lm_code.h`. This rebuilds your vendor daemon, as well as other files in the SDK.
4. Provide a convenient way for your customers to get your new vendor daemon.

## 10.4 Server Node Configuration

FLEX*lm* supports:

- Single license server nodes
- Redundancy via a license file list
- Three-server redundancy

If all the end user's data is on a single file server, then there is no need for redundant servers, and GLOBEtrotter Software recommends the use of a single server node for the FLEX*lm* daemons. If the end user's data is split among two or more server nodes and work is still possible when one of these nodes goes down or off the network, then multiple server nodes can be employed.

In all cases, an effort should be made to select stable systems as server nodes; in other words, do not pick systems that are frequently rebooted or shut down for one reason or another. Multiple server nodes can be any supported server nodes—it is not required that they be the same architecture or operating system.

### 10.4.1 Redundancy Via a License File List

Redundancy via a license file list is implemented by each end user setting an `LM_LICENSE_FILE` or *`VENDOR`*`_LICENSE_FILE` environment variable. This type of redundancy is best explained by example. If ten licenses are desired for both "f1" and "f2," you would issue your customer two license files with a count of 5 for each of "f1" and "f2," each keyed to a different license server node. The license server nodes can be physically distant.

The license files would look like:

License 1 for “chicago”

```
SERVER chicago 17007ea8 1700
DAEMON xyzd /etc/mydaemon
FEATURE f1 xyzd 1.000 01-jan-2005 5 SIGN=26C7DD9CD665
FEATURE f2 xyzd 1.000 01-jan-2005 5 SIGN=0739D2F78CE4
```

License 2 for “tokyo”

```
SERVER tokyo 17007ea8 1700
DAEMON xyzd /etc/mydaemon
FEATURE f1 xyzd 1.000 01-jan-2005 5 SIGN=16BE40E1DAEE
FEATURE f2 xyzd 1.000 01-jan-2005 5 SIGN=6DB6F3E40E61
```

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

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

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

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

The application attempts to check out licenses from the first license server in the list; if that fails for any reason, the second license server is tried. This type of redundancy can be used to balance the usage of licenses across several license server nodes.

### 10.4.2 Three-Server Redundant Servers

FLEX*lm* supports a set of three license servers to be used in a redundant manner. If any two of the three license servers are up and running (two out of three license servers is referred to as a *quorum*), the system is functional and hands out its total complement of licenses.

---

**Note:** Note that the use of two license servers is strongly discouraged, because a set of redundant server nodes with only two server nodes gives you two points of failure—you must always have both server nodes running.

---

When redundant servers are started, they elect a *master,* which performs all licensing operations. The other two servers are there to provide a secure licensing mechanism in the event of hardware failure or if the master 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.

These three-server redundant servers should have excellent communications and should be on the same subnet. Often this means that the three servers should be located physically close to each other. This form of redundancy requires that the servers exchange heartbeats periodically, and poor communications can cause poor performance. Redundant servers should never be configured with slow communications or dial-up links.

Three-server redundancy does not provide load-balancing of licenses because with three-server redundancy, only one of the three servers is master, capable of issuing licenses. Because all clients must contact the master, all clients must have reliable networking to a single node.

#### Generating a License File for Redundant Servers

To generate a license file that uses redundant servers, three servers must be specified when the license is created. Unlike independent servers, each SERVER line will require a port number, which can be any number from 1024 to 32000 which is unused at the end-user site.

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 the SERVER lines do not agree in all license files, FLEX*lm* uses alphabetical order to determine the master and the following messages are generated in the debug 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 license 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, have the customer ensure that the order of the server lines is consistent on all license server nodes.

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

### 10.4.3 Comparing License File List to Three-Server Redundancy

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

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

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

Finally, if one license server in the list 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 available to monitor license servers.
- When load-balancing is needed for clients located far apart, e.g., London and Tokyo. You can make servers available locally, with remote servers available as backup.
- License file list is more forgiving if one of the license server nodes goes down.
- It's not limited to three servers (any number will work). For wide-area networks, local servers can be specified first, with remote servers available as backup.
- Clients do not require reliable networking to a single node with a license file list, so this is recommended where networking itself requires redundancy.

Chapter 11

# Software Vendor Utility Programs

## 11.1 makekey

The FLEX*lm* SDK includes the `makekey` utility program used for the creation of license files. `makekey` is the easiest way to get started, because it asks a few questions and creates a correct license file. Once you have become familiar with the license file format, which is described in detail in the *FLEXlm Reference Manual*, you may want to use the `lmcrypt` utility to generate your signatures from a template file.

`makekey` is a standalone license file generator. `makekey` can be used as-is to generate license files for your customers or it can be used as an example for you to create your own customized license file generation program. If customizing, note that the essential function call is lc_cryptstr().

`makekey` asks a number of questions and then generates the license file for the specific end user.

`makekey` allows you to enter all data for a customer's license file from scratch or use an existing customer license file to update the feature lines.

The license file is left in the current directory with the name `license.dat`.

```
% makekey [-verfmt { 2 | 3 | 4 | 5 | 5_1 | 6 | 6_1 | 7 | 7_1 }] \
     [-maxlen n]
```

Licenses can be generated to be compatible with older versions (i.e., not using any of the features of the newer versions) by using the `-verfmt` argument.

Licenses can have shorter or longer lines with the `-maxlen` argument. This is normally used to generate files with short lines so they'll be less likely to have newlines inserted by mailers. For this a length of 50 is common.

## 11.2 lmcrypt

Once you know the format of the license file FEATURE or INCRMENT lines that you need to create, `lmcrypt` is an easier way to create your signatures than `makekey`. `lmcrypt` replaces the signature, which can be simply `SIGN=0`, in a license file with the correct signature, which is then ready to ship to a customer. To use `lmcrypt`, you need to either understand the FLEX*lm* license file syntax or use an example license file. Examples are available in the `examples/licenses` directory.

Usage:

```
lmcrypt [files][-i infile] [-o outfile] [-maxlen n] [-e errfile]
     [-verfmt { 2 | 3 | 4 | 5 | 5_1 | 6 | 6_1 | 7 | 7_1 }]
```

If no input file is specified, or if specified as "`-`" or stdin, standard input is used. If no output file is specified, or if specified as "`-`" or stdout, standard output is used. *`files`* are read and written back in place. If no file arguments are specified, `lmcrypt` reads stdin and writes stdout. All signatures are recomputed. `lmcrypt` will only work on lines that have a daemon name matching the vendor's daemon name.

If this is not possible, an error is produced, and the affected license line is left unaltered.

The maximum line length is controlled with the `-maxlen` argument. A value of 50 is commonly used to make shorter lines less subject to mailers inserting newlines. Licenses can be generated to be compatible with older versions by using the `-verfmt` argument.

The simplest way to use `lmcrypt` is:

1. Copy an existing good file to another name, say, `newlicense`.
2. Edit `newlicense`, and make any desired changes, such as changing the feature name, or number of licenses, or adding new features, and save the file.
3. Change the existing signature on each FEATURE line to `SIGN=0`.
4. Type the following:

   ```
   % lmcrypt newlicense
   ```

   `newlicense` is then filled with correct signatures and is usable by a customer. Comments are passed through unaltered.

See `lmcrypt.c` in the `machind` directory.

#### ERROR RETURNS

Errors are printed to stderr, or as specified with `-e`. Most errors will prevent generation of signatures and the text will be output unchanged from the input. An example of error reporting: If `-e` is not used, the error messages also appear at the top of the output file.

Input:

```
FEATURE f1 demo 1.a50 01-jan-2005 uncounted HOSTID=08002b32b161 \
     SIGN=0
```

Error reported:

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

#### SEE ALSO

- Chapter 12, “The License File”

## 11.3 makepkg

The `makepkg` utility is similar to `makekey` and is used to make PACKAGE lines for license files. The reason this utility is separate from `makekey` is that you will often want to ship PACKAGE lines with your product and issue enabling FEATURE or INCREMENT lines later when the product is sold to individual customers.

The source for `makepkg`, `makepkg.c`, is in the `machind` directory. You are encouraged to modify this source as needed.

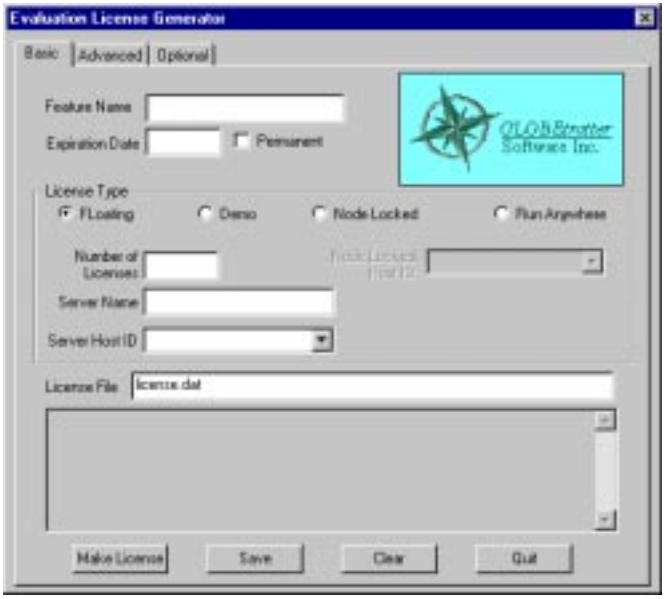
---

**Note:** PACKAGE lines can also be created with the `lmcrypt` utility, but not with `makekey`.

---

## 11.4 genlic.exe (Windows Only)

`genlic` is a visual license generation program provided with the SDK. After experimenting with the license files that come with the demo, you may wish to generate you own license files using `genlic.exe`. While `genlic` can generate most common license types, it cannot generate every license type which FLEX*lm* supports. The range of license types which `genlic` can generate has been limited in order to make it easier for you to get started. If you need a license which `genlic` cannot generate, you will need to use `lmcrypt` or the lc_cryptstr() function.




You must edit `lm_code.h` and insert your demo vendor codes to use `genlic`. If you don't, `genlic` will not run.

To use `genlic`, fill out the form for the type of license you want, then click the Make License button. The text of the license file will appear in the window. To create multiple features, edit the form and click Make License again. Each feature will be appended to the window. Click Save to save your work.

The main screen consists of the following:

**Feature Name** — Name of the feature to be licensed. Characters must be alphanumeric and/or "_".

**Expiration Date** — Date the license will expire. Valid date format is *dd-mmm-yyyy* (for example, 01-nov-2007 or 30-dec-2007). If the Permanent box is checked, the license will never expire and it has the date of 1-jan-0 (or the keyword "permanent") in the license file.

**License Type** — Any of the four following license types:

**Floating** — Anyone on the network can use the licensed software, up to the allowed number of licenses. The floating license type requires the following fields:

Number of Licenses — Total number of licenses that can be checked out at any given time.

Server Name — Name of the license server.

Server Host ID — Ethernet address or Disk Volume Serial Number of the license server. The information for your machine is available in the drop-down menu.

**Demo** — The licensed software will run on any system (it uses “DEMO” as the hostid). The licensed software is in DEMO mode.

**Node Locked** — The licensed software can only run on one particular computer. The node-locked license requires the following field:

Node Locked Host ID — Ethernet address, Dongle ID, or Disk Volume Serial Number of a particular system on the network (workstation). This information for your system is available in the drop-down menu.

**Run Anywhere** — The licensed software will run on any system (it uses ANY as the hostid). The licensed software is in normal mode. (The difference between the “ANY” and “DEMO” hostid is that the licensed software can check for demo feature lines and then alter the behavior as desired.)

**License File** — Name and path to the license file. To activate the browse feature, leave the field blank and click on the Save button.

**Make License** — Generate a FEATURE line without saving to a file.

**Save** — Write the licenses window to a license file.

**Clear** — Reset the content of the licenses window.

**Quit** — Exits the program.

**Advanced** — Settings concerning the vendor information and license server data.

**Version** — Feature’s version that is supported by the current license file.

**Start Date** — Date the current license file will take effect. This date will be authenticated.

**Use Decimal Format** — This allows you to generate the decimal format of the license file (easier to read numbers over the telephone).

**License Sharing** — This allows you to specify how multiple license requests share licenses.

None — Disable license sharing feature (default).

User — Allows multiple copies with the same user to share the same license.

Host — Allows multiple copies on the same computer to share the same license.

User and Host — Allows multiple copies of the same user on one computer to share the same license.

Site — Allows any system on the network to share the same feature (unlimited use at one site).

**Vendor Info (optional)** — Information that can be recorded on the license line.

**Optional** —

**TCP Port** — Optional port number (>1024 and <64000) to use on the SERVER line.

**SPX Address** — If using a FLEX*lm* Novell server, port number for IPX/SPX protocol to use on the SERVER line.

**Vendor Name** — "demo" by default. This is used in the VENDOR line. The name will change with modification to `lm_code.h` for `VENDOR_NAME` *vendor*.

**Vendor Daemon Path** — Path to the vendor daemon. This is used in the VENDOR line.

---

**Note:** After the advanced or optional features are saved, they will be used for all future license files until they are changed.

---

## 11.5 Integrating the License Certificate Manager

The License Certificate Manager supports automatic downloading and installation of license files over the Internet. This is enabled by default. It can be disabled using the FLEXible API, as described in the *FLEXlm Reference Manual*.

Chapter 12

# The License File

## 12.1 Format of the License File

A license file consists of the following sections:

SERVER/VENDOR lines

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

USE_SERVER line

A USE_SERVER line, if used, usually follows the SERVER line and indicates that a client application should not process the rest of the license file itself, but should check out the license directly from the license server. GLOBEtrotter recommends the use of a USE_SERVER line, particularly where performance is important.

FEATURE lines

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

Comment lines

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

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

Vendors and license administrators will read the license file to understand how the licensing will behave, e.g., what features are licensed, the number of licenses, and whether these licenses are node-locked. The options are very broad for what can be specified in a license file.

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

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

- Host names on SERVER lines
- (Optional) port numbers on SERVER or VENDOR lines
- (Optional) path names on VENDOR lines
- (Optional) options file path names on VENDOR lines
- (Optional) Lowercase *keyword=value* pairs on FEATURE lines

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

---

**Note:** The SERVER hostid(s) and everything on a FEATURE line (except the vendor daemon name and lowercase *keyword=value* pairs) are input to the authentication algorithm to generate the signature for that FEATURE line.

---

The license file format is documented in detail in the *FLEXlm Reference Manual*.

### 12.1.1 Example License File

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

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

## 12.2 Locating the License File

Client applications use the following rules for locating a license file:

1. If either the LM_LICENSE_FILE or *VENDOR*_LICENSE_FILE (where *VENDOR* is the ISV's vendor daemon name) environment variable is set, they are used. *VENDOR*_LICENSE_FILE is used first. *VENDOR*_LICENSE_FILE is used only by products from *VENDOR* and by lmutil and lmtools.
2. The environment variables LM_LICENSE_FILE and/or *VENDOR*_LICENSE_FILE can also be set in the Windows registry or, on UNIX, in $HOME/.flexlmrc. The Windows registry is in \HKEY_LOCAL_MACHINE\SOFTWARE\FLEXlm License Manager. These locations are also automatically set upon successful checkout.
3. The license location(s) can be set the CHECKOUT() or lp_checkout() calls. This is searched after LM_LICENSE_FILE and/or *VENDOR*_LICENSE_FILE.
4. FLEX*lm* utilities accept -c *license_file_path* argument, which specifies the license path(s). If this is set, the environment variables are ignored.
5. If none of the other locations are specified, then a "backup"/default location is used. Relying on this default location is not recommended.

   UNIX:

   /usr/local/flexlm/licenses/license.dat

   Windows:

   C:\flexlm\license.dat

---

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

---

### 12.2.1 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.

- A directory, where *dir*/*.lic are used in alphabetical order, as if specified like a license file list.
- @*host*, where *host* is the host name of the license server, when the SERVER has no port number, or the number is between 27000 and 27009. (New in v6—unsupported in older versions.)
- *port*@*host*, where *port* is the port number and *host* is the host name that come from the SERVER line.
- The actual license file text, with START_LICENSE\n as a prefix, and \nEND_LICENSE as suffix, where the embedded newlines are required. While awkward to specify in most shells, this is most useful inside a program.

Examples:

- Only one file is specified.

```
% setenv LM_LICENSE_FILE license.lic
```

- license.dat and */a/b/vendor/*licenses/*.lic are used.

```
% setenv LM_LICENSE_FILE license.dat:/a/b/vendor/licenses
```

- If the server is running on the same system on a default port, @localhost will find it. After that it looks for ./*.lic and */vendor/*licenses/*.lic.

```
% setenv LM_LICENSE_FILE @localhost:.:/vendor/licenses
```

---

**Note:** Before v5, both the client and server needed to read the *same* license file, since the client passes the signature from the FEATURE line to the vendor daemon. With FLEX*lm* v5+, *port*@*host* or USE_SERVER solves this problem, since the client never reads the license file FEATURE lines in these cases.

---

## 12.3 Hostids for FLEX*lm*-Supported Machines

A *hostid* is a means used to uniquely identify a specific machine. When you create a license with FLEX*lm*, you bind this license to a hostid. This binding allows only the authorized user(s) to run your software. The binding is created when you generate the *license file* for your customer. Section 12.4.1, “Simple Uncounted License,” and Section 12.4.3, “Simple Floating (Counted) License,” are two good examples of license files you might create to authorize your customer to run your software.

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. The program `lmhostid` will print the exact hostid that FLEX*lm* expects to use on any given machine. The *FLEXlm Reference Manual* lists alternate methods of determining the hostid for common machine architectures.

On the PC platforms, FLEX*lm* supports the following types of hostids:

- Intel CPU ID (Pentium III+ and FLEX*lm* v7.0d+ required). This hostid is probably preferable where available, but since this turned off by default, the user must enable it using BIOS Setup.
- Disk Volume Serial Number (`DISK_SERIAL_NUM=`)
- FLEXid hardware key, often called a *dongle* (`FLEXID=`)
- Ethernet address, default (twelve hex characters, e.g., 1234567890ab)

The decision to use a particular hostid is made in the license file. You do not need to do anything in your application to decide which hostid type to use. For more hostid information refer to the *FLEXlm Reference Manual*.

## 12.4 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.

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

### 12.4.1 Simple Uncounted License


- Uncounted licenses have unlimited use on the hostid specified. Uncounted licenses require no server.
- When the expiration date is “permanent” (or if a date is specified with a year of “0”), the license never expires.

- This license supports versions 0.0 through 2.0 (inclusive).

### 12.4.2 Expiring Demo License

```
FEATURE f0 demo 2.0 3-mar-2005 uncounted HOSTID=DEMO SIGN=AB0CC0C16807
                    |                  |
                    Expiration Date    Optional Attribute
```

- This license expires on 3 March, 2005.
- For a license which expires after 1999, use all four year digits, e.g. “1-jan-2001.”
- The “DEMO” hostid indicates that this license allows “f0” to run on any system. In addition, the client application can also detect that it is in demo mode, and could behave differently.

### 12.4.3 Simple Floating (Counted) License

```
SERVER speedy 08002b32b161
VENDOR demo
FEATURE f1 demo 2.0 permanent 9 SIGN=DBCC10416777
```

- SERVER and VENDOR lines required.
- Unexpiring.
- Floating — runs on any node. No hostid on FEATURE line.
- Limited to nine concurrent licenses.
- Server restricted to hostid “08002b32b161.” To remove this restriction, use hostid of “ANY” (e.g., SERVER speedy ANY 2837).

The breakdown of the SERVER and VENDOR lines is illustrated here:

- Host name can be changed by the end user. If host name is `this_host`, clients running on the same node as the server will work fine. Clients on other nodes will fail unless the host name is changed, or the clients use `@host` (or `port@host` if a port number is specified on the SERVER line) to find the server.
- `lmgrd` uses the `PATH` environment variable, or the current directory to find the vendor daemon binary.
- Nothing else can be changed on these two lines. Everything else is authenticated by the signature.

### 12.4.4 INCREMENT

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

- INCREMENT — the server adds up licenses for all lines for the same feature name. The concurrent usage limit is 3 (1 + 2).
- The first INCREMENT line could be a FEATURE line and the behavior would be the same.
- INCREMENT lines must differ in some way — otherwise only one will be used.

### 12.4.5 INCREMENT, Node-Locked

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

- One license is available on hostid “80029a3d.”
- Two licenses are available on “778da450.”
- The server tracks these licenses independently, in separate *pools.*
- This behavior *only* works with INCREMENT, not FEATURE, because with FEATURE, the server only recognizes the first FEATURE line for a given feature name.

### 12.4.6 Mixed Floating (Counted) and Uncounted

```
SERVER speedy 08002b32b161
VENDOR demo
FEATURE f1 demo 2.0 permanent 1 HOSTID=80029a3d SIGN=7B9F02AC0645
INCREMENT f1 demo 2.0 permanent 2 HOSTID=778da45 SIGN=6BAFD2BC1C3D
FEATURE f0 demo 2.0 permanent uncounted HOSTID=554066fa \
     SIGN=AB0CC0C16807
```

- Checkouts of "f0," since it is *uncounted*, may not communicate with the server — they only verify that the client is on node "554066fa" and that the version is <= 2.0. If USE_SERVER is specified, or either *VENDOR*_LICENSE_FILE or LM_LICENSE_FILE is set to @*host* (or *port*@*host* if a port number is specified on the SERVER line), then checkouts do require a server and their usage is logged.
- The "f0" line does not require the SERVER or VENDOR lines, and in fact could reside in another license file altogether.

### 12.4.7 Optional FEATURE Attributes

```
SERVER speedy 08002b32b161
VENDOR demo
INCREMENT f1 demo 2.0 permanent 4 SIGN=DBCC10416777
INCREMENT f1 demo 2.0 permanent 3 SIGN=BBDC1081492A
UPGRADE f1 demo 2.0 3.0 permanent 5 SIGN=3B8C60B10227
FEATURE pkg2 demo 1.0 permanent 1 HOSTID=12345678 \
     VENDOR_STRING=vd OVERDRAFT=1 \
     DUP_GROUP=UHD ISSUER=issuer NOTICE=notice\
     START=1-jan-2005 vendor_info=vi dist_info=di\
     user_info=ui asset_info=ai ck=161 SIGN=BB9C4071436D
```

- Most optional attributes are in *keyword*=*value* format.


- The values of the following keywords, which are printed in lowercase by the license generators, can be modified by the user and are *not* part of the signature: asset_info, dist_info, user_info, vendor_info, and ck.

#### ATTRIBUTES IN DETAIL

`VENDOR_STRING=vd`

The vendor-defined string is used for customization by the vendor, often to license subfeatures.

`HOSTID=FLEXID=8-12345678`

Use locked to hostid "FLEXID=8-12345678" (a node with the hardware key with id 8-12345678 attached).

`OVERDRAFT=1`

Usage is limited to license count (1) plus OVERDRAFT (1) = 2. The application can detect this state, and it is logged in the report log file.

`DUP_GROUP=UHD`

All usage by the same user on the same host and display are counted as a single use.

`START=1-jan-2000`

Optional start date.

```
ISSUER=issuer NOTICE=notice vendor_info=vi dist_info=di\
     user_info=ui asset_info=ai
```

Unused by FLEX*lm*. Can be used for customization by vendor or end user.

`ck=161`

A checksum, used by the `lmcksum` utility to validate the line.

See the *FLEXlm Reference Manual* for complete information about FEATURE line keywords.

### 12.4.8 PACKAGE

```
PACKAGE pkg demo 1.0 COMPONENTS="c1 c2 c3 c4 c5 c6 c7 c8" \
     SIGN=504091605DCF
FEATURE pkg demo 1.0 permanent uncounted HOSTID=778da450 \
     SIGN=DB5CC00101A7
```

The two lines above are a more efficient way of delivering:

```
FEATURE c1 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=D03F02432106
FEATURE c2 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=99375F40FD85
FEATURE c3 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=68FAC130DB90
FEATURE c4 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=D3D617E2075A
FEATURE c5 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=5A91D6EFB68C
FEATURE c6 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=8F75798EB975
FEATURE c7 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=790545E90575
FEATURE c8 demo 1.0 permanent uncounted HOSTID=778da450 SIGN=9EE9E788087F
```

- The FEATURE line *enables* the PACKAGE line.
- The components all inherit information from the enabling FEATURE line. In this example, they all inherit the expiration date, number of licenses, and hostid.
- The enabling FEATURE line must match the name, version, and vendor name of the PACKAGE line.
- The PACKAGE line is usually shipped with the product, since it contains no customer-specific fields.
- PACKAGE lines can be shipped in a separate file that never needs user editing, so long as the file is include in the license file list.

### 12.4.9 SUITE

```
PACKAGE office demo 1.0 OPTIONS=SUITE \
     COMPONENTS="write paint draw" SIGN=00504091605D
FEATURE office demo 1.0 permanent 3 SIGN=DB5CC00101A7
```

- There is no equivalent in FEATURE lines for this behavior.
- The client application checks out both a component feature, and, automatically, a copy of a feature called “office.” Without OPTIONS=SUITE, this additional checkout would not occur.
- A feature called “office” is created, in addition to all the components.
- This license file indicates that after *any* three licenses of any of the components are used, no further licenses are available for checkout. Without the OPTIONS=SUITE qualifier, there would be three licenses of *each* of the three components.

## 12.5 License in a Buffer

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

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

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

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

```
CHECKOUT(LM_RESTRICTIVE, "f1", "1.0",
     "path/to/license.dat:START_LICENSE\n\
     FEATURE f1 demo 1.0 permanent \
     uncounted HOSTID=ANY \
     VENDOR_STRING=\"Acme Inc\" SIGN=50A35101C0F3\nEND_LICENSE"
```

In this example, *path/to/license.dat* is first in the list, followed by the license in the string.

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

## 12.6 Decimal Format Licenses

Licenses can be represented in decimal format, to make license delivery easier for customers without access to email. Decimal has the advantage that it’s simpler to type in, and often the licenses are much shorter. There are notable exceptions, however, which are explained below.

To generate a decimal format license, use the `-decimal` 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. Please see the *FLEXlm Reference Manual* for a more complete description of the decimal format. Refer to the *FLEXlm End Users Guide* for more information on `lminstall`.

Chapter 13

# FLEX*lock* and License Certificate Manager (Windows Only)

FLEX*lock* makes it easy to add "try before you buy" licensing to a product that uses FLEX*lm*. The License Certificate Manager (LCM) is a web-based system for downloading and installing license files. A software vendor can use FLEX*lock* with LCM to provide a fully automated "try before you buy" program to its customers.

## 13.1 FLEX*lock*

When you enable FLEX*lock* in one of your FLEX*lm*-licensed products, FLEX*lock* automatically allows the user a trial period after the initial installation along with warnings and dialogs explaining the evaluation system. At the completion of the trial period, the user has the option to stop using or to purchase the product. After purchasing your product, the user is given a license file that will enable your product to run in a normal FLEX*lm* manner.

To use the FLEX*lock* functionality, you need to:

1. Determine the operation and parameters of the "try before you buy" functionality.
2. Run `flsetup.exe` to define these FLEX*lock* parameters and create the FLEX*lock* distribution file, `fldata.ini`.
3. Enable FLEX*lock* functionality in your program.
4. Link your application with a FLEX*lm* library and the FLEX*lock* library.
5. Ship `fldata.ini` and `flrsrc.dll` with your application.

### 13.1.1 FLEX*lock* Security

We've made every effort to make the FLEX*lock* feature secure. However, due to the type of security technology used for FLEX*lock*, it is less secure than the rest of FLEX*lm*. It is possible, but not ethical, for a user to "reset" FLEX*lock* to re-evaluate the product, but there are some built-in deterrents:

- Resetting FLEX*lock* requires tampering with the registry which many users will be reluctant or unable to do.
- If FLEX*lock* is "reset" for re-evaluation, the user will continually get popup FLEX*lock* reminders which are an annoyance.
- The popups also serve as a deterrent to companies that are usually law-abiding in that they make the theft somewhat public.

Using FLEX*lock* with the dynamic libraries (`lmgr327b.dll` and `flcflxa.dll`) is a significant security risk because these could be replaced by a hacker.

### 13.1.2 The FLEX*lock* Configuration Editor

To define the operation of FLEX*lock* features, you must run the `flsetup.exe` program and generate the `fldata.ini` file. `flsetup.exe` allows developers to quickly define:

The type of product trial including:

- Fixed number of days
- Fixed numbers of executions
- Expiration on a fixed date

How to behave when the trial expires:

- Don't run
- Run with a warning

Message displayed:

- When the product is first executed
- During the trial phase
- When the trial phase expires, including instructions on how to purchase the product

Product's Attributes:

- Product name
- Company name
- Copyright notice

After configuring these parameters and saving the file, run the File→Create Distribution Files. This generates the `fldata.ini` file that will need to be shipped with the product. A DLL file will need to be shipped with the application as well (see Section 13.1.4, "Linking Your Application with FLEXlm Libraries").

### 13.1.3 Enabling FLEX*lock* Functionality in Your Code

To enable the functionality of the FLEX*lock* you will need to specify to FLEX*lm* that you will be permitting this functionality.

If you are using either the Trivial or Simple API, specify an additional license policy of LM_FLEXLOCK to your checkout call, for example:

```
LM_USE_FLEXLOCK();
CHECKOUT(LM_RESTRICTIVE | LM_FLEXLOCK, "myfeature", "1.0",
     "license.dat");
```

The LM_USE_FLEXLOCK() macro can be used with the Trivial, Simple, or FLEXible API.

For additional security, use LM_A_FLEXLOCK_INSTALL_ID and LM_A_FLEXLOCK in the FLEXible API.

If your application checks out more than one feature and you want to use FLEX*lock*, or if you want your application to check whether it is licensed with FLEX*lock*, you will need to use the FLEXible API (see Section 2.7, "FLEXlock," in the *FLEXlm Reference Manual*.

### 13.1.4 Linking Your Application with FLEX*lm* Libraries

#### LINKING WITH THE FLEX*LM* STATIC LIBRARY

If you link your FLEX*lock*-enabled application with `lmgr.lib`, you will also need to link with `flock.lib`. You will ship `fldata.ini` and `flrsrc.dll` to your customers with your application. These files should be placed in the same directory as your program executable.

#### LINKING WITH THE FLEX*LM* DYNAMIC LIBRARY

Using the dynamic FLEX*lm* library with FLEX*lock* is not recommended because it is a security risk.

If you link your FLEX*lock*-enabled application dynamically with FLEX*lm*, do not use the LM_USE_FLEXLOCK() macro before your FLEX*lock* checkout call. You will link with `lmgr327b.lib` and will ship `fldata.ini` and `flcflxa.dll` in addition to `lmgr327b.dll` to your customers with your application. These files should be placed in the same directory as your program executable.

## 13.2 License Certificate Manager (LCM)

LCM uses GTweb*Licensing* to obtain a license file over the Internet and have it automatically installed on a user's system (without the need for copying and pasting). An application automatically supports LCM if `lcmflxa.dll` is shipped with the application.

When a license is unavailable for an application, a dialog appears. One of the options presented in the dialog is to download a license from the Internet. If a user selects this option, the user is prompted for a URL and key.

The default URL for the LCM is `www.globetrotter.com/`*`vendor`*, where *`vendor`* is your vendor daemon name. This default URL can be changed with the FLEXible API (see LM_A_LCM_URL in the *FLEXlm Reference Manual*).

To demo the LCM, use the `lmclient` sample program with feature "lcm."

Chapter 14

# Integration Guidelines

The following sections describe some things to consider when you integrate FLEX*lm* into your software application.

## 14.1 Single Server vs. Redundant Servers

You will have to help your end user decide how many license server machines to run, as the signatures are partially derived from the list of license server machine hostids in a license file. See Section 10.4, “Server Node Configuration,” for guidelines about setting up license servers.

## 14.2 Where to Install FLEX*lm* Components

When your software’s installation procedure installs FLEX*lm* components at an end user’s site, there are some things you should keep in mind:

- All license server executables (`lmgrd` and vendor daemons) should be *local* on the machine(s) that will run them. A corollary of this is that you should not run license servers on diskless nodes.
- There should be a *local copy* of the license file on each server node. It is fine to NFS-mount the license file for client access, but each license server node should have a local copy. A better approach is to have each user set either the *VENDOR*`_LICENSE_FILE` or `LM_LICENSE_FILE` environment variable to `@`*`host`* (or *`port`*`@`*`host`* if a port number is specified on the SERVER line). In fact, for large license files, `@`*`host`* is more efficient, because it doesn’t need to read the license file. For more information, see Section 12.2, “Locating the License File.”
- If a user configures a three-server redundant server, a copy of `lmgrd`, your vendor daemon, and the license file(s) should be placed on the disk of *each* license server node.

## 14.3 Keeping Your Software Secure

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

### 14.3.1 Counterfeit Resistant Option (CRO)

FLEX*lm* offers a Counterfeit Resistant Option (CRO), which is a separately priced add-on. Without CRO, FLEX*lm* utilizes the standard FLEX*lm* license key, which uses a proprietary, non-public-key digital signature method. CRO offers a standard public-key system which is recognized by the security community, and recommended for US government work (with US government export approval). The system comes from Certicom (http://www.certicom.com) and uses Elliptical Curve Cryptography.

What is the difference between the CRO SIGN= signature and the standard FLEX*lm* license key? The standard FLEX*lm* license key is easier to crack and it is more likely that a license file might some day be counterfeited. To date, the current version of the standard FLEX*lm* license key has not been compromised. However, with the CRO signature, the possibility of counterfeiting becomes more remote.

First you must decide whether you want to use CRO. The evaluation vendor keys include CRO, so you can test CRO with the demo SDK, although by default the demo does not enable CRO. You have to redefine `LM_STRENGTH` in `machind/lm_code.h` for CRO to be tested.

If you decide that you want to use CRO, then you have to decide what length signature (SIGN=) attribute you want by setting the `LM_STRENGTH` value in `machind/lm_code.h`:

- LM_STRENGTH_DEFAULT—12 characters (non-public-key)
- LM_STRENGTH_113BIT—58 characters
- LM_STRENGTH_163BIT—84 characters
- LM_STRENGTH_239BIT—120 characters

If you set `LM_STRENGTH_113` to `_239` bit, you must rebuild the FLEX*lm* SDK, using `make` in the *`platform`* directory (e.g., `sun4_u5`), or `nmake` on Windows.

Information about migrating to CRO from a previous version of FLEX*lm* can be found in Appendix F, “Migrating to the Counterfeit Resistant Option,” in the *FLEXlm Reference Manual.* Information about implementing CRO for new

FLEX*lm* customers can be found in Chapter 5, “Incorporating Production FLEXlm into a UNIX Application,” and Chapter 6, “Incorporating Production FLEXlm into a Windows Application.”

### 14.3.2 FLEX*lm* Points of Vulnerability

The following list shows known points of vulnerability in FLEX*lm* in increasing order of difficulty to break. GLOBEtrotter Software also maintains a list of techniques for making your implementation more secure—please contact GLOBEtrotter technical support (support@globes.com) for a description of these techniques.

#### EASY TO BREAK

- Running a debugger on application code if it is released with unstripped executables (on UNIX) or as a debug version (on Windows).

#### DIFFICULT TO BREAK, DEPENDING ON LICENSE POLICY IN APPLICATION

- Killing the daemons, because a dead daemon is detected by the client heartbeat. If, however, you do not use one of the built-in timers and you do not call a heartbeat function, then your software protection could be bypassed by someone who kills the daemons each time that the application reaches the maximum license limit, because the application would never detect that the daemon went down.

  To reduce the potential for theft by killing and restarting daemons:

  - Call a heartbeat function at least every 120 seconds (but not more often than every 30 seconds).
  - If reconnection is attempted, notify the user and take whatever action is appropriate.

#### VERY DIFFICULT TO BREAK

- Guessing the signatures that belong in the license file. FLEX*lm*’s authentication algorithm takes the user-visible data fields (number of licenses, expiration date, version number, vendor-defined string, feature name, hostids of all servers, plus any optional authenticated fields) and combines them with the vendor’s private encryption seeds to produce a signature.
- Writing a new daemon that emulates your vendor daemon. FLEX*lm* encrypts the traffic between client and vendor daemon to make this point of attack much more difficult.
- Running the debugger on a stripped (UNIX) or a non-debug (Windows) executable. This requires someone to find the FLEX*lm* calls without any symbol table knowledge.

Chapter 15

# End-User License Administration

## 15.1 End-User Options File

For complete syntax and descriptions of end-user options, see the *FLEXlm End Users Guide*.

End users can customize software usage via a vendor daemon options file. An options file allows the end user to reserve licenses for specified users or groups of users, to allow or disallow software usage to certain people, to set software timeouts, and to start a report log that can be read by SAM*report*.

The vendor daemon options file can be specified on the VENDOR line in the license file as follows:

```
VENDOR vendor [vendor_daemon_path] [options_file_path]
```

If the path to the vendor daemon is not specified on the VENDOR line, the path to the options file must be preceded by `options=`:

```
VENDOR vendor [options=options_file_path]
```

If the vendor daemon is v6+, the options file does not need to be specified if the options file is both:

1. Named *vendor*`.opt`
2. Located in the same directory as the license file

The following options keywords are available:

- EXCLUDE
- EXCLUDEALL
- GROUP
- HOST_GROUP
- INCLUDE

- INCLUDEALL
- LINGER
- MAX
- MAX_OVERDRAFT
- NOLOG
- REPORTLOG
- RESERVE
- TIMEOUT
- TIMEOUTALL

The TIMEOUT option allows an idle license to return to the free pool for use by another user. TIMEOUT requires that the application not send heartbeats when it is idle. Therefore, for the TIMEOUT option to work, the application must support it. If FLEX*lm* timers are disabled, then the application must regularly call a heartbeat function when active, and not call a heartbeat function when inactive. If a heartbeat function is not called regularly when active, the TIMEOUT option can cause the client to lose its license. If the application uses FLEX*lm* timers (LM_MANUAL_HEARTBEAT not set), a TIMEOUT specification will be ineffective.

## 15.2 License Administration Tools

Syntax and detailed descriptions of `lmutil` commands can be found in the *FLEXlm End Users Guide*.

On UNIX, all license administration tools are contained in the single executable `lmutil`. `lmutil` behavior is determined by its first argument, or its `argv[0]` name. `lmutil` renamed to `lmstat` will behave the same as `lmutil lmstat`. When you installed your FLEX*lm* SDK, the installation created hard links from `lmutil` to each `lmutil` program names listed. FLEX*lm*. You should also create these hard links when your software is installed on your customer's system.

On Windows, `lmutil.exe`, a command-line program similar to `lmutil` on UNIX, is provided. A utility is invoked with `lmutil.exe` *`function`*, where *`function`* is `lmstat`, `lmdiag`, etc. A tool interface called `lmtools` is also provided on Windows with the same functionality as `lmutil.exe`.

`lmutil` and `lmutil.exe` contain the following utility programs:

- `lmcksum` — Prints license checksums.
- `lmdiag` — Diagnoses license checkout problems.
- `lmdown` — Gracefully shuts down all license daemons (both `lmgrd` and all vendor daemons) on the license server node (or on all three nodes in the case of three-server redundant servers).
- `lmhostid` — Reports the hostid of a system.
- `lminstall` — Converts license files between different formats.
- `lmnewlog` — Moves existing report log information to a new file name and starts a new report log file with existing file name.
- `lmremove` — Releases a hung license to the pool of free licenses.
- `lmreread` — Causes the license daemons to reread the license file and start any new vendor daemons.
- `lmstat` — Displays the status of a license server.
- `lmswitchr` — Switches the report log to a new file name.
- `lmver` — Reports the FLEX*lm* version of a library or binary file. If no file name is specified, `lmver` looks for the `liblmgr.a` library file to detect its version.

All FLEX*lm* utilities take the following arguments:

| | |
|---|---|
| `-c` *`license_file_path`* | Operate on this license file. |
| `-e` *`filename`* | Redirects error message to a file. |
| `-v` | Print version and exit. |
| `-verbose` | Prints longer description of all errors found. The output from the utilities may be harder to read with this option, but is useful for diagnostics. |

**Note:** If an application is active when a license is removed with `lmremove`, it will simply checkout the license again (assuming the applications FLEX*lm* timers are enabled or a heartbeat function is called). Therefore, `lmremove` cannot be used to “steal” licenses.

Chapter 16

# End-User Installation Instruction Template

To ensure that your customers can use your FLEX*lm*-managed product easily and successfully, use the information in this chapter as a guideline about what files and information they need.

## 16.1 FLEX*lm* Files Your Customers Will Require

When your application software is built with FLEX*lm* calls, you will need to ship the following files in addition to the files that you normally ship in your software installation kit:

| | |
|---|---|
| *vendor*`.lic` | The license file, customized for your customer. |
| `lmgrd` (or `lmgrd.exe`) | The license manager daemon. |
| `lmutil` (or `lmutil.exe` and `lmtools.exe`) | FLEX*lm* utility program. |
| `xyzd` (or `xyzd.exe`) | Your vendor daemon. |
| `lcmflxa.dll` | If you're using the LCM feature. |
| `fldata.ini` and `flrsrc.dll` | If you're using the FLEX*lock* feature. |

## 16.2 Information Every Customer Needs to Know

In addition to installing your software, your customer will need to do the following steps.

### 16.2.1 Install the License File

We recommend that the application specify a default directory for a license file in the installed product hierarchy, for example, *`install_path`*`/licenses`, where *`install_path`* is determined by the end user during installation and `licenses` is a directory). At run time, *`install_path`* is determined (many applications do this with an environment variable or global internal character string), and the default license directory is specified in CHECKOUT(), lp_checkout(), or lc_set_attr(*`job`*`, LM_A_LICENSE_DEFAULT, ...`).

The license file should be installed with a `.lic` suffix in the default license directory so the application can find it. The application does not need to know the exact name of the file, only that it ends with `.lic` and exists in the specified directory. A useful practice is to name the file the date of installation, in *`yyyymmdd`* format. For example, if the license is generated 10 January, 2005, it can be installed as:

```
install_path/licenses/20050110.lic
```

We recommend that the license file be delivered by email. If delivered by email, the entire email message can be saved in the specified location; email headers and extraneous text are automatically ignored by FLEX*lm*.

If the license file is delivered in decimal format, the end user should use `lminstall` to generate a readable license file. `lminstall` will request the name of its output license file and will default to a license file called *`yyyymmdd`*`.lic`. This license file should then be moved to the *`install_path`*`/licenses` directory.

As an alternative to emailing a license file, you can use the License Certificate Manager (LCM) on Windows. LCM works with GT*licensing* to provide Internet-based license fulfillment.

### 16.2.2 And If Licenses are Counted...

#### INSTALL LMGRD AND VENDOR DAEMON

These can be installed wherever the end user prefers. `lmgrd` will automatically find the vendor daemon if they are in the same directory or in `lmgrd`'s `PATH`. Otherwise, the user will need to edit the license file to add the path to the vendor daemon to the VENDOR line.

#### INSTALL LMGRD TO START AT BOOT (OPTIONAL)

On UNIX, edit the appropriate boot script, which may be `/etc/rc.boot`, `/etc/rc.local`, `/etc/rc2.d/S`*xxx*, `/sbin/rc2.d/S`*xxxx*, etc. Remember that these scripts are run in `/bin/sh`, so do not use the `csh` "`>&`" redirection syntax.

Each UNIX operating system can have some quirks in doing this, but the following script has been successfully tested for HP700 systems. See the notes following the script listing for a full explanation.

```
/bin/su daniel -c 'echo "starting lmgrd" > \
        /home/flexlm/v7.1/hp700_u9/boot.log'


/bin/nohup /bin/su daniel -c "umask 022; \
        /home/flexlm/v7.1/hp700_u9/lmgrd -c \
        /home/flexlm/v7.1/hp700_u9/license.dat >>& \
        /home/flexlm/v7.1/hp700_u9/boot.log"


/bin/su daniel -c 'echo "sleep 5" >> \
        /home/flexlm/v7.1/hp700_u9/boot.log'
/bin/sleep 5


/bin/su daniel -c 'echo "lmdiag" >>\
        /home/flexlm/v7.1/hp700_u9/boot.log'


/bin/su daniel -c '/home/flexlm/v5.12/hp700_u9/lmdiag -n -c\
        /home/flexlm/v7.1/hp700_u9/license.dat >> \
        /home/flexlm/v7.1/hp700_u9/boot.log'


/bin/su daniel -c 'echo "exiting" >>\
        /home/flexlm/v7.1/hp700_u9/boot.log'
```

Please note the following about how this script was written:

- All paths are specified in full, since no paths can be assumed at boot time.
- Since no paths are assumed, the vendor daemon must be in the same directory as `lmgrd`, or the VENDOR line must be edited to include the full path to the vendor daemon binary.
- The `su` command is used to run `lmgrd` as a non-root user, "daniel." We recommend that `lmgrd` not be run as "root," since it can be a security risk to run any program as "root" that does not require root permissions, and `lmgrd` does not require root permissions.

- Daniel has a `csh` login, so all commands executed as “daniel” must be in `csh` syntax. All commands not executed as “daniel” must be in `/bin/sh` syntax, since that’s what’s used by the boot scripts.
- The use of `nohup` and `sleep` are required on some operating systems, notably HP-UX and Digital UNIX, for obscure technical reasons. These are not needed on Solaris and some other operating systems, but are safe to use on all.
- `lmdiag` is used as a diagnostic tool to verify that the server is running and serving licenses.

---

**Note:** On IBM RS6000 systems, `/etc/rc` cannot be used, because TCP/IP is not installed when this script is run. Instead, `/etc/inittab` must be used. Add a line like this to `/etc/inittab` after the lines which start networking:

```
rclocal:2:wait:/etc/rc.local > /dev/console 2>&1
```

---

#### START LMGRD MANUALLY (IF NOT STARTED ON BOOT)

First, make sure the vendor daemon is in the same directory as `lmgrd`, or in `lmgrd`’s `PATH`, or that the user has edited the license file to include the path to the vendor daemon on the VENDOR line.

Start the license manager daemon as follows:

```
% lmgrd -c license_file_list -l debug_log_path (UNIX)
C:\> lmgrd -c license_file_list -l debug_log_path (Windows)
```

where *`license_file_list`* is the full path to the license file or a delimited license file list and *`debug_log_path`* is the path to the debug log file. See the *FLEXlm End Users Guide* for descriptions of the messages in the debug log file.

# Index