revenera™

FlexNet Publisher 2023 R3 (11.19.5)

Licensing for Java Programming Reference Guide

Legal Information

Book Name: FlexNet Publisher 2023 R3 (11.19.5) Licensing for Java Programming Reference Guide

Part Number: FNP-11195-JPR00

Product Release Date: December 2023

Copyright Notice

Copyright © 2023 Flexera Software

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

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

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

Intellectual Property

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

Restricted Rights Legend

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

Contents

Licensing for Java Programming Reference Guide 5

Identifying the Toolkit Directories 6

Product Support Resources 7

1 Getting Started with FlexNet Publisher Licensing Toolkit for Java 9

Prerequisites 9

Evaluating the Licensing for Java 9

Incorporating Licensing for Java into Your Product 9

What's Installed 10

Licensing for Java and License Management 11

Licensing Toolkit Components 11

Unserved Licenses 12

Served Licenses 12

License Model Considerations 13

Licensing Toolkit for Java and Security 14

2 Evaluating the Toolkit for Java on UNIX 15

Prerequisites 15

Add flexlm.jar to CLASSPATH 16

Create the demoInfo Class File 16

Compile the Sample FlexEnabled Java Application 17

Create a Valid License File 18

Start the License Server 18

Run the Sample Java Application 19

Stop the License Server 19

3 Evaluating Licensing Toolkit for Java on Windows 21

Prerequisites 21

Add flexlm.jar to CLASSPATH 22

Create the demoInfo Class File 22

Compile the Sample FlexEnabled Java Application 23

Create a Valid License File 24

Start the License Server 24

Run the Sample Java Application 25

Stop the License Server 25

4 Incorporating Toolkit for Java Into Your Product 27

Configuring a Production Toolkit for Java 28

Define the CLASSPATH Environment Variable 28

Create the *vendor*Info.class File 29

Specifying the License File Location 29

Add Toolkit for Java API Function Calls to Your Product 30

A Basic Toolkit for Java Implementation 30

Testing Your Application 31

Create a License File 31

Start the License Server 32

Execute Your FlexEnabled Application 32

Deploying your Product 33

5 The Licensing Toolkit for Java API Programming Model 35

Licensing Toolkit for Java Framework 36

Feature Specifiers 36

License Sources 36

The VendorInfo class 37

Connection Data 38

Count 38

Checkout-Data Strings 38

Heartbeats 39

Automatic Heartbeats 40

Manual Heartbeats 41

Semi-Automatic Heartbeats 42

Hostids 42

Native Hostids 43

Vendor-Defined Hostids 44

FlexNet Publisher Licensing Toolkit for Java API 46

FlexNet Publisher Licensing Toolkit for Java Error Code Values 48

Glossary 53

Index 65

Licensing for Java Programming Reference Guide

This manual, *Licensing for Java Programming Reference Guide*, provides:

- Instructions for evaluating FlexNet Publisher Licensing Toolkit for Java
- Instructions for incorporating FlexNet Publisher Licensing Toolkit for Java into your product
- Considerations for upgrading your existing FlexNet Publisher Licensing Toolkit for Java to version 11.10

Once you are familiar with FlexNet Publisher Licensing Toolkit and how it operates, work through either Evaluating the Toolkit for Java on UNIX or Evaluating Licensing Toolkit for Java on Windows in this manual with the provided example FlexEnabled Java program. Then, learn how to incorporate FlexNet Publisher Licensing Toolkit into your Java application in Incorporating Toolkit for Java Into Your Product. This section gives you an overview for incorporating calls to the FlexNet Publisher Licensing Toolkit Java API into your code, testing your application with a valid license, and deploying the final FlexEnabled Java application.

For FlexNet Publisher Licensing Toolkit issues not specific to Java, see either the *Programming Reference for License File–Based Licensing* or *Programming Reference for Trusted Storage–Based Licensing*.

Table 1 ▪ *Licensing for Java Programming Reference Guide* Section Overview

Topic	Content
Licensing for Java Programming Reference Guide	Provides an overview of the *Licensing for Java Programming Reference Guide*.
Getting Started with FlexNet Publisher Licensing Toolkit for Java	Gives a brief introduction to FlexNet Publisher Licensing Toolkit for Java, and requirements for use, and a brief overview of FlexNet Publisher Licensing Toolkit and its components.
Evaluating the Toolkit for Java on UNIX	Walks you though the process of evaluating FlexNet Publisher Licensing Toolkit for Java behavior on UNIX.
Evaluating Licensing Toolkit for Java on Windows	Walks you though the process of evaluating FlexNet Publisher Licensing Toolkit for Java behavior on Windows.

Table 1 ▪ *Licensing for Java Programming Reference Guide* Section Overview

Topic	Content
Incorporating Toolkit for Java Into Your Product	This chapter gives you the steps necessary to: ● Configure FlexNet Publisher Licensing Toolkit for Java to work with your production FlexNet Publisher Licensing Toolkit. ● Incorporate the calls to the FlexNet Publisher Licensing Toolkit for Java API into your own Java product. ● Test your FlexEnabled Java product. ● Deploy your application to your end user.
The Licensing Toolkit for Java API Programming Model	Provides an in-depth overview of how to entitle a given feature of your FlexEnabled Java product, including how to successfully check out a license for that feature.
Glossary	Contains a collection of terms and their meanings.

Identifying the Toolkit Directories

Table 2 lists the main toolkit directories and the shortcut names that are used to refer to them in the documentation.

Table 2 ▪ Toolkit Directories

Shortcut Name	Default Location
`sdk_install_dir`	The directory where in install the toolkit. **Windows platforms:** No default, you define this path at installation time. **UNIX platforms:** No default, you define this path at installation time.

`sdk_platform_dir`	The directory within the toolkit that contains files specific to the operating-system platform of the toolkit. The folder label identifies this platform (such as x64_n6).
	Windows platforms:
	`<sdk_install_dir>\<sdk_platform_dir>`
	UNIX platforms:
	`<sdk_install_dir>/<sdk_platform_dir>`

Table 2 ▪ Toolkit Directories

Shortcut Name	Default Location

`machind`	Toolkit files used to license your applications.
	Windows platforms:
	`<sdk_install_dir>\machind`
	UNIX platforms:
	`<sdk_install_dir>/machind`

`examples`	Sample code you can use as templates.
	Windows platforms:
	`<sdk_install_dir>\examples`
	UNIX platforms:
	`<sdk_install_dir>/examples`

Product Support Resources

The following resources are available to assist you:

- Revenera Product Documentation
- Revenera Community
- Revenera Learning Center
- Revenera Support

Revenera Product Documentation

You can find documentation for all Revenera products on the Revenera Product Documentation site:

https://docs.revenera.com

Revenera Community

On the Revenera Community site, you can quickly find answers to your questions by searching content from other customers, product experts, and thought leaders. You can also post questions on discussion forums for experts to answer. For each of Revenera's product solutions, you can access forums, blog posts, and knowledge base articles.

https://community.revenera.com

Revenera Learning Center

The Revenera Learning Center offers free, self-guided, online videos to help you quickly get the most out of your Revenera products. You can find a complete list of these training videos in the Learning Center.

https://learning.revenera.com

Revenera Support

For customers who have purchased a maintenance contract for their product(s), you can submit a support case or check the status of an existing case by first logging into the Revenera Community and then making selections on the **Get Support** menu, including **Open New Case** and other options.

1

Getting Started with FlexNet Publisher Licensing Toolkit for Java

This section covers necessary information to help you get started in using FlexNet Publisher Licensing Toolkit for Java, including prerequisites, component installation location and description, and security.

Prerequisites

Prior to incorporating FlexNet Publisher Licensing Toolkit for Java into your product, you must evaluate and then build your production toolkit.

Evaluating the Licensing for Java

Before evaluating FlexNet Publisher Licensing Toolkit for Java, you are required to complete the following:

- Install the FlexNet Publisher Licensing Toolkit. This is a package that is separate from the FlexNet Publisher Licensing Toolkit for Java toolkit.
- Work through the sections in *Getting Started with License File–Based Licensing* corresponding to your development platform.
- Install the Sun Java Developers Kit on your development platform. See the FlexNet Publisher Licensing Toolkit Release Notes for information about the supported Sun JDK version.

Incorporating Licensing for Java into Your Product

In addition to the items in Evaluating the Licensing for Java, build your production toolkit customized with your private, vendor-specific information. Tamper-Resistant Licenses (TRL) must be enabled. See the *Development Environment Guide* for details. Do this build before incorporating FlexNet Publisher Licensing Toolkit for Java into your application.

Note ▪ *A C development environment is required to build the toolkit. Consult the Development Environment Guide and the Release Notes for your platform's C development environment requirements.*

What's Installed

The subdirectories and components that comprise FlexNet Publisher Licensing Toolkit for Java are listed in Table 1-1. Paths are relative to the installation directory.

Table 1-1 ▪ FlexNet Publisher Licensing Toolkit for Java Components

Component	Description
java/	
`EccpressoAll.jar`	FlexNet Publisher Licensing Toolkit for Java TRL classes. Needed at runtime.
`flexlm.jar`	FlexNet Publisher Licensing Toolkit for Java client classes. Needed at runtime.
`flexlmutil.jar`	FlexNet Publisher Licensing Toolkit classes used to generate your vendor-specific *vendor*`Info.java` file, where `vendor` is your vendor daemon name. Not needed at runtime.
`vendorInfo.bat (Windows)` `vendorInfo.sh (UNIX)`	Batch file to build *vendor*`Info.java` file if Tamper-Resistant Licenses are not enabled. (Where `vendor` is your vendor daemon name.)
`flexlmmgmt.jar`	System file only. Do not use.
examples/	
`BasicExample.java`	Sample FlexEnabled Java application. Used during the FlexNet Publisher Licensing Toolkit for Java evaluation.
`ConnDataExample.java`	Example showing how to override the display name, hostname, username or servertimeout value when connecting to the license server.
`FeatureDetails.java`	Example showing how to get a feature list from a license source.
`HeartbeatExample.java` `HeartbeatWorker.java`	Set of examples showing how to configure heartbeats into your product.
`IndependentCheckouts.java`	The point of the example is to show multiple uses of a single feature on a single license server connection (LicenseSource instance), with separate checkins of various quantities, without the use of checkout data.
`MyHostId.java`	Example showing how to configure a vendor-defined hostid.
`TSExample.java`	Example showing how to use licenses held in trusted storage on a FlexEnabled client. It lists all the features available in local trusted storage and checks out and checks in one of those features.

Table 1-1 ▪ FlexNet Publisher Licensing Toolkit for Java Components

Component	Description
Documents/	
`api/index.html`	The FlexNet Publisher Licensing Toolkit for Java API documentation.

Conflicting Usage of Certicom Security Library (EccpressoAll.jar)

FlexNet Publisher Licensing Toolkit for Java utilizes the security library `EccpressoAll.jar` from Certicom. In the event that your product also utilizes a different version of this library, there could be a clash during the Java library load. In such a case, use the `FlexnetPublisher.properties` file to mention the path of `EccpressoAll.jar`.

1. If the FlexNet Publisher Certicom jar file `EccpressoAll.jar` is in `D:\Certicom\lib`, for example, then create a `FlexnetPublisher.properties` file and, in that file, add the path as shown here:

   ```
   CerticomJarPath=d:/Certicom/lib/
   ```

2. Save the file and set the location of the file in the classpath. For example, if you save the file in `C:/flexjava` then set the classpath as shown here:

   ```
   set CLASSPATH=C:\flexjava
   ```

Note ▪ *Use a forward slashes in the path of the jar file in properties file (all platforms).*

Licensing for Java and License Management

The FlexNet Publisher Licensing Toolkit for Java works in conjunction with the license server from the toolkit to provide a FlexEnabled Java application to your end user.

Licensing Toolkit Components

This section provides a basic overview of the components and their interaction with each other.

Components are organized based on your license model. License models are commonly classified according to their requirement for a license server:

- Licenses not served by a license server: they are available directly to the application. This is commonly referred to as an uncounted, or *unserved* license model.
- Licenses served by a license server: the licenses are available via a license server. The license server is either on the same system as the application or remote on the network. This is commonly referred to as a *served* license model. License files that include SERVER, VENDOR, and optionally, USE_SERVER lines require a license server. Optionally, uncounted licenses can be served in order to track usage.

  Security ▪ *Served uncounted INCREMENT line signatures are not linked to the hostid on the SERVER line, therefore using served uncounted licenses with HOSTID=ANY is an exploit risk. Instead, Revenera recommends served counted licenses with a large count.*

Unserved Licenses

License models that do not need a license server have three components:

- The FlexEnabled Java product, which references the `com.macrovision.flexlm` classes in the FlexNet Publisher Licensing Toolkit for Java component, `flexlm.jar`.
- The `flexlm.jar` component, itself, from FlexNet Publisher Licensing Toolkit for Java.
- The license file, which is created with the `lmcrypt` utility from your production toolkit. Figure 1-1 shows this model.

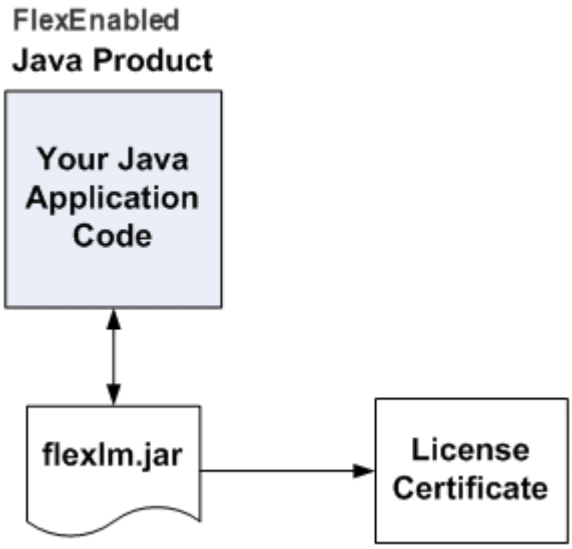

Figure 1-1: Unserved License Component Model

The FlexEnabled application and license file reside on the same machine. All counted licenses in the local license file are ignored.

Even though a local license file may exist, there are some circumstances in which license requests are forwarded to a license server:

- The license file contains a `USE_SERVER` line, in which case all local `FEATURE/INCREMENT` lines are ignored. The request is forwarded to the server specified in the accompanying `SERVER` line.
- The FlexEnabled application cannot find the desired license in the local license file and a `SERVER` line is present. The request is forwarded to the specified server.

Served Licenses

For served licenses, there are five required components:

- The FlexEnabled Java application, which references the `com.macrovision.flexlm` classes in the `flexlm.jar` file.
- The `flexlm.jar` component, itself, from Licensing Toolkit for Java.
- The license server manager, `lmadmin` or `lmgrd`, from your production toolkit.

- The vendor daemon from your production toolkit. Along with the license server manager, `lmadmin` or `lmgrd`, this is a core component of the license server.
- The license file, which is created with the `lmcrypt` utility from your production toolkit.

In addition to these four components, there are three optional components:

- Debug log File—created and written by `lmadmin` or `lmgrd`.
- Report log File—created and written by the vendor daemon for use by FlexNet Manager.
- Options file—file created and maintained by the license administrator.

Figure 1-2 shows the relationship these components have to one another. Note that a distinction is made between FlexNet Publisher Licensing Toolkit components and FlexNet Publisher Licensing Toolkit for Java components. The final deliverable to your end user includes components from both areas.

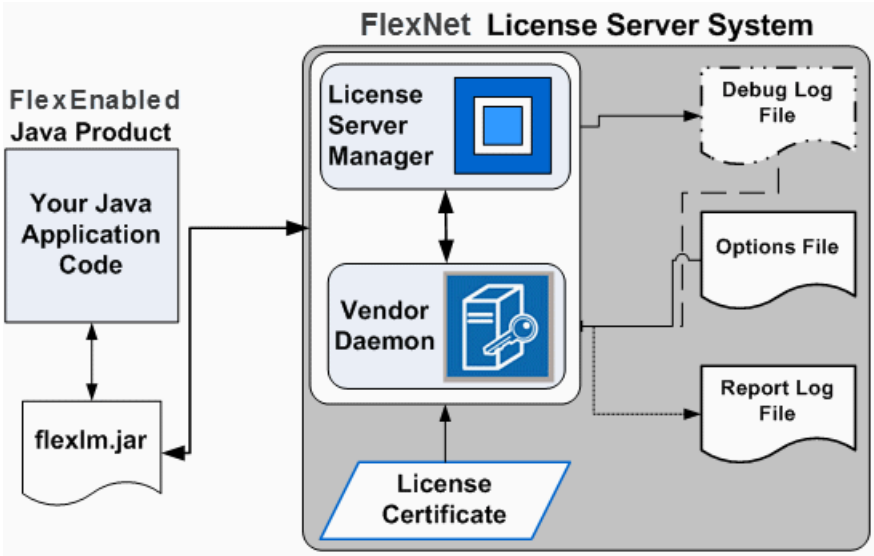

Figure 1-2: Served License Component Model

Typically, the license server components reside on a system in the network. However, it can optionally reside on the same system as the FlexEnabled application.

The three optional files, Debug Log, Administration Options, and Usage Log, are configured by the license administrator at the customer site.

Note ▪ *The environment variables,* `LM_LICENSE_FILE` *and* `VENDOR_LICENSE_FILE` *are not directly supported by FlexEnabled Java products.*

License Model Considerations

In order to control entitlement, the license file for an uncounted license model is node-locked to a specific hostid, which the FlexEnabled application then authenticates. However, because Java is so portable and generic, it is difficult to obtain a unique hostid directly from the Java product.

In a served model, the license server authenticates the hostid; many more hostid choices exist in this configuration. For this reason, it is recommended that your end user uses a license server. The FlexEnabled Java application requests a license from the license server and is either granted or denied the license, with all authentication taking place at the license server level.

For more information on available hostids, see Hostids.

Licensing Toolkit for Java and Security

There are special security considerations for companies using FlexNet Publisher Licensing Toolkit for Java because there aren't many possibilities for node-locked hostid types in the Java environment and Java applications can be relatively easily decompiled.

It is recommended that your end users use a license server rather than using node-locked uncounted licenses. In this case, the user application must point to the license server with `port@host`.

The license server can be used to serve uncounted licenses. However, served uncounted INCREMENT line signatures are not linked to the hostid on the SERVER line, therefore using served uncounted licenses with HOSTID=ANY is an exploit risk. Instead, Revenera recommends served counted licenses with a large count.

FlexNet Publisher Licensing Toolkit for Java supports Tamper-Resistant Licenses (TRL), which are used to reduce the possibility of counterfeit licenses. TRL utilizes Certicom's public-key system based upon elliptical curve cryptography (ECC), which is considered secure, fast, and efficient (see http://www.certicom.com). TRL is the default license signature mechanism; no other license signatures are supported. For more information about Tamper-Resistant Licenses, refer to the *Programming Reference for License File–Based Licensing*.

2

Evaluating the Toolkit for Java on UNIX

This demonstration walks you though the process of evaluating Toolkit for Java behavior on UNIX. You perform the following to test the toolkit:

- Add flexlm.jar to CLASSPATH
- Create the demoInfo Class File
- Compile the Sample FlexEnabled Java Application
- Create a Valid License File
- Start the License Server
- Run the Sample Java Application
- Stop the License Server

Prerequisites

As a prerequisite for evaluating the Toolkit for Java:

- Install the FlexNet Publisher Licensing Toolkit. This is a separately purchased package and is distinct from the Toolkit for Java.
- Work through the evaluation procedure for UNIX platforms in *Getting Started with License File–Based Licensing*.
- Install the Java Developers Kit on your development platform. See the *Release Notes* for information about the supported Sun JDK version.
- Install the Toolkit for Java.

The `machind` and `<sdk_platform>` (for example, `sun4_u5`) directories of the FlexNet Publisher Licensing Toolkit contain files that you use to evaluate the FlexNet Publisher Licensing Toolkit for Java. Locate the following files:

- `machind/lm_code.h`
- `<sdk_platform>/lmpubkey.h`

Note ▪ *If this file does not exist, you need to create it. To do so, from the* `<sdk_platform>` *directory, enter* **`lmnewgen -pubkey`**.

- `<sdk_platform>/lmseeds.h`
- Sample license file (`<sdk_platform>/lmadmin/demo/demo.lic`)
- License generation utility (`<sdk_platform>/lmcrypt`)
- License server manager `lmadmin` (`<sdk_platform>/lmadmin/lmadmin`) or `lmgrd` (`sdk_platform/lmgrd`)
- Sample vendor daemon for `lmadmin` (`<sdk_platform>/lmadmin/demo/demo`) or for `lmgrd` (`<sdk_platform>/demo`)

Note ▪ *The license server* `lmadmin` *is used for this test. To use* `lmadmin`*, you must copy the vendor daemon* `demo` *and its associated library* `demo_libFNP.so` *from the* `<sdk_platform>` *directory to the* `<sdk_platform>/lmadmin/demo` *directory.*

Now you are ready to begin your evaluation of Toolkit for Java on UNIX.

Add flexlm.jar to CLASSPATH

To run a FlexEnabled Java application, you must set your `CLASSPATH` environment variable to include the current directory, the path to `flexlm.jar`, and the path to `EccpressoAll.jar`. If you installed the FlexNet Publisher Licensing Toolkit for Java into the default location, you would add to or set your `CLASSPATH` environment variable as follows:

```
setenv CLASSPATH .:<sdk_install_dir>/java/flexlm.jar: \
                 <sdk_install_dir>/java/EccpressoAll.jar
```

Make sure that the new `CLASSPATH` setting is active before proceeding.

Create the demoInfo Class File

Each software publisher must create a concrete subclass of `VendorInfo` containing vendor-specific information such as vendor name, LM_SEEDs, TRL keys, and vendor keys. An instance of a subclass of a `VendorInfo` must be associated with every `License` object. The name of the subclass is of the form `vendorInfo.class`, where `vendor` is the vendor name as registered with Revenera. The `demoInfo.class` is a subclass of `VendorInfo` for the vendor named `demo`.

Task ***To create the subclass demoInfo.class***

1. Copy `machind/lm_code.h`, `<sdk_platform>/lmpubkey.h`, and `<sdk_platform>/lmseeds.h` from the `<sdk_install_dir>` directory to the `<sdk_install_dir>/java` directory.

   Note ▪ *If* `lmpubkey.h` *does not exist, you need to create it. To do so, from the* `<sdk_platform>` *directory, enter* **`lmnewgen -pubkey`**.

2. Check that the following conditions are met:
   - `VENDOR_NAME` in `lm_code.h` is set to **`"demo"`**.
   - The `LM_STRENGTH` setting in `lm_code.h` is one of the tamper-resistant licenses (TRL) settings. It must have the same value as the `LM_STRENGTH` setting in `lm_code.h` that you used to build the production toolkit.
   - `lm_code.h` contains your demo vendor keys.

     This information is used to create a `demoInfo` class that is used to create a license object in your FlexEnabled application.
3. Change to the `<sdk_install_dir>/java` directory.

   In `vendorInfo.sh`, make sure the `-classname` switch specifies **`demoInfo`**. The contents of `vendorInfo.sh` should look like the example below:

```
java -classpath flexlmutil.jar;flexlm.jar; \
                EccpressoAll.jar \
      CreateVendorInfo -code lm_code.h -pubkey lmpubkey.h \
                       -seeds lmseeds.h \
                       -classname demoInfo
```

4. On the command line, execute `vendorInfo.sh` to create `demoInfo.java`.
5. Compile `demoInfo.java`:

   `javac -classpath flexlm.jar:EccpressoAll.jar demoInfo.java`
6. Copy `demoInfo.class` into the `<sdk_install_dir>/examples` directory.

Compile the Sample FlexEnabled Java Application

Compile the sample FlexEnabled application, `BasicExample.java`, so it references `demoInfo.class`, which was created in Create the demoInfo Class File.

Task ***To compile the sample FlexEnabled application***

1. Change to the `<sdk_install_dir>/examples` folder.
2. In `BasicExample.java`, check that the **`demoInfo`** constructor is properly referenced as in the excerpt below:

```
/*...*/
private static VendorInfo getVendorInfo()
{
    VendorInfo vInfo = null;
    try
    {
        vInfo = new demoInfo();
    }
/*...*/
    return vInfo;
}
```

3. Compile `BasicExample.java`:

```
javac -classpath \
      .:../java/flexlm.jar:../java/EccpressoAll.jar \
      BasicExample.java
```

Create a Valid License File

If not already generated during the Licensing toolkit evaluation, generate a counted license file, `counted.lic`, with `lmcrypt` from your Licensing toolkit. The contents of the license file must reflect the vendor and feature information that you put in the sample application.

Task ***To create a valid license file***

1. Open `<sdk_platform>/lmadmin/demo/demo.lic` in a text editor.
2. Edit the `FEATURE` line, making sure it is similar to the following:

   ```
   FEATURE f3 demo 3.0 permanent 4 SIGN=0
   ```

3. Save `demo.lic`.
4. Open a command window; and from the `<sdk_platform>` directory, enter the following:

   `lmcrypt ./lmadmin/demo/demo.lic`

5. Verify that a signature was generated. The `FEATURE` line should now look something like this:

   ```
   FEATURE f3 demo 3.0 permanent 4 SIGN="0008 0B78 63B1 1A00 DCE8 132C
      C70D 4A00 33E4 B817 F307 5041 8D0D 7A7A 2E47"
   ```

The number of hexadecimal characters in the license signature will correspond to the `LM_STRENGTH` setting you selected in Step 2 of Create the demoInfo Class File.

Start the License Server

In order to check out a counted **f3** license using the FlexEnabled application, you must start the license server. This process involves importing the vendor daemon `demo` and starting `lmadmin`. (When the license server starts, it uses the license file you just created in Create a Valid License File.)

Note ▪ *In general, when starting the license server for the first time, use the command-line process, as described next. This test uses* `lmadmin` *as the license server.*

Task ***To start the license server***

1. For purposes of this test, copy the vendor daemon `demo` and its associated library `demo_libFNP.so` from the `<sdk_platform>` directory to the `<sdk_platform>/lmadmin/demo` directory.
2. From the `<sdk_platform>/lmadmin` directory, enter the following to import the `demo` vendor daemon:

```
lmadmin -import demo/demo.lic -force
```

3. Enter the following to start the license server in the background:

```
lmadmin -adminOnly no -allowRemoteStopServer yes
```

Run the Sample Java Application

Task ***To run the sample FlexEnabled Java application***

From the `<sdk_install_dir>/examples` directory, enter the following:

```
java -classpath \
     .:../java/flexlm.jar:../java/EccpressoAll.jar \
     BasicExample f3 3.0 @host
```

where `host` is the name of the machine on which the license server was started in Start the License Server.

When you run the sample application, it will check out a license for version 3.0 of feature **f3**. Press **Enter** to check the license back in.

Stop the License Server

Task ***To stop the license server***

From the `<sdk_platform>` directory, enter the following:

```
lmdown -c ./lmadmin/demo/demo.lic
```

You have completed the demo of the Licensing for Java toolkit on UNIX. When you are ready to incorporate your production vendor keys into the Licensing for Java toolkit, see Incorporating Toolkit for Java Into Your Product. Remember to set `LM_STRENGTH` to an appropriate value before you build your production Licensing toolkit. If you want to evaluate the Licensing for Java toolkit on Windows, see Evaluating Licensing Toolkit for Java on Windows.

3

Evaluating Licensing Toolkit for Java on Windows

This demonstration walks you though the process of evaluating FlexNet Publisher Licensing Toolkit for Java behavior on Windows.You perform the following to test the toolkit:

- Add flexlm.jar to CLASSPATH
- Create the demoInfo Class File
- Compile the Sample FlexEnabled Java Application
- Create a Valid License File
- Start the License Server
- Run the Sample Java Application
- Stop the License Server

Prerequisites

As a prerequisite for evaluating FlexNet Publisher Licensing Toolkit for Java on Windows:

- Install the FlexNet Publisher Licensing Toolkit. This is a separately purchased package and is distinct from FlexNet Publisher Licensing Toolkit for Java.
- Work through the FlexNet Publisher Licensing Toolkit evaluation for Windows platforms in *Getting Started with License File-Based Licensing*.
- Install the Java Developers Kit on your development platform. See the FlexNet Publisher Licensing Toolkit Release Notes for information about the supported Sun JDK version.
- Install the FlexNet Publisher Licensing Toolkit for Java.

The `machind` and `<sdk_platform>` (for example, `i86_n3`) folders in the installation folder of the FlexNet Publisher Licensing Toolkit contain files that you use to evaluate FlexNet Publisher Licensing Toolkit for Java. Locate the following files:

- `machind\lm_code.h`

- `<sdk_platform>\lmpubkey.h`

  Note ▪ *If this file does not exist, you need to create it. To do so, from the* `<sdk_platform>` *directory, enter* **`lmnewgen -pubkey`**.

- `<sdk_platform>\lmseeds.h`
- Sample license file (`<sdk_platform>\lmadmin\demo\demo.lic`)
- License generation utility (`<sdk_platform>\lmcrypt.exe`)
- License server manager, which is either `lmadmin` (`<sdk_platform>\lmadmin\lmadmin.exe`) or `lmgrd` (`<sdk_platform>\lmgrd.exe`)
- Sample vendor daemon for `lmadmin` (`<sdk_platform>\lmadmin\demo\demo.exe`) or `lmgrd` (`<sdk_platform>\demo.exe`)

  Note ▪ *The license server* `lmadmin` *is used for this test. To use* `lmadmin`*, you must copy the vendor daemon* `demo.exe` *and its associated library* `demo_libFNP.dll` *from the* `<sdk_platform>` *directory to the* `<sdk_platform>\lmadmin\demo` *directory.*

Now you are ready to begin your evaluation of the FlexNet Publisher Licensing Toolkit for Java on Windows.

Add flexlm.jar to CLASSPATH

To run a FlexEnabled Java application, you must set your `CLASSPATH` environment variable to include the current directory, the path to `flexlm.jar`, and the path to `EccpressoAll.jar`. If you installed FlexNet Publisher Licensing Toolkit for Java into the default location, you would add to or set the following components in the `CLASSPATH` environment variable as follows:

```
C:\Program Files\Flexera Software\FLEXnet Licensing Toolkit for JAVA <version>\flexlm.jar; \
C:\Program Files\Flexera Software\FLEXnet Licensing Toolkit for JAVA <version>\EccpressoAll.jar
```

Make sure that the new `CLASSPATH` setting is active before proceeding.

Create the demoInfo Class File

Each software publisher must create a concrete subclass of `VendorInfo` containing vendor-specific information such as vendor name, LM_SEEDs, TRL keys, and vendor keys. An instance of a `VendorInfo` subclass must be associated with every `License` object. The name of the subclass is of the form *vendor*`Info.class`, where `vendor` is the vendor name as registered with Revenera.

Specifically, `demoInfo.class` is a subclass of `VendorInfo` for the vendor named `demo`. Follow the steps below to create the subclass `demoInfo.class`.

Task ***To create the subclass demoInfo.class***

1. Copy `machind\lm_code.h`, `<sdk_platform>\lmpubkey.h`, and `<sdk_platform>\lmseeds.h` to the `<sdk_install_dir>\java` folder.

   Note ▪ *If* `lmpubkey.h` *does not exist, you need to create it. To do so, from the* `<sdk_platform>` *directory, enter* **`lmnewgen -pubkey`**.

2. Check the following:

   - `VENDOR_NAME` in `lm_code.h` is set to **`"demo"`**.
   - The `LM_STRENGTH` setting in `lm_code.h` is one of the tamper-resistant licenses (TRL) settings. It must have the same value as the `LM_STRENGTH` setting in `lm_code.h` that you used to build the production toolkit.
   - The file contains your demo vendor keys.

   This information is used to create a **`demoInfo`** class that is used to create a license object in your FlexEnabled application.

3. Change to the `<sdk_install_dir>\java` folder.

4. In `vendorInfo.bat`, make sure the `-classname` switch specifies **`demoInfo`**. The contents of `vendorInfo.bat` should look like the example given below.

```
java -classpath flexlmutil.jar;flexlm.jar; \
                EccpressoAll.jar \
      CreateVendorInfo -code lm_code.h -pubkey lmpubkey.h \
                       -seeds lmseeds.h \
                       -classname demoInfo
```

5. In a command window, execute `vendorInfo.bat` to create `demoInfo.java`.

6. Compile `demoInfo.java`:

   `javac demoInfo.java`

7. Copy `demoInfo.class` into the `<sdk_install_dir>\examples` folder.

Compile the Sample FlexEnabled Java Application

Compile the sample FlexEnabled application `BasicExample.java` so that it references `demoInfo.class` that was created in Create the demoInfo Class File.

Task ***To compile the sample FlexEnabled application***

1. Change to the `<sdk_install_dir>\examples` folder.

2. In `BasicExample.java`, check that the **`demoInfo`** constructor is properly referenced as in the excerpt below:

```
/*...*/
private static VendorInfo getVendorInfo()
{
    VendorInfo vInfo = null;
    try
    {
        vInfo = new demoInfo();
    }
/*...*/
return vInfo;
}
```

3. Compile `BasicExample.java`:

   `javac BasicExample.java`

Create a Valid License File

If not already generated during the FlexNet Publisher Licensing Toolkit evaluation, generate the license file, `demo.lic`, using `lmcrypt` from your FlexNet Publisher Licensing Toolkit. The contents of the license file must reflect the vendor and feature information that you put in the sample application.

Task ***To create a valid license file:***

1. Open `<sdk_platform>\lmadmin\demo\demo.lic` in a text editor.
2. Edit the `FEATURE` line, making sure it is similar to the following:

   ```
   FEATURE f3 demo 3.0 permanent 4 SIGN=0
   ```

3. Save `demo.lic`.
4. Open a command window, and from the `<sdk_platform>` directory, enter the following:

   `lmcrypt .\lmadmin\demo\demo.lic`

5. Verify that a signature was generated. The `FEATURE` line should now look something like this:

   ```
   FEATURE f3 demo 3.0 permanent 4 SIGN="0008 0B78 63B1 1A00 DCE8 132C
       C70D 4A00 33E4 B817 F307 5041 8D0D 7A7A 2E47"
   ```

The number of hexadecimal characters in the license signature will correspond to the `LM_STRENGTH` setting you selected in Step 2 of Create the demoInfo Class File.

Start the License Server

In order to check out a counted **f3** license using the FlexEnabled application, you must start the license server. This process involves importing the vendor daemon `demo` and starting `lmadmin`. (When the license server starts, it uses the license file you just created in Create a Valid License File.)

Note ▪ *In general, when starting the license server for the first time, use the command-line process, as described next. This test uses* `lmadmin` *as the license server.*

Task ***To start the license server***

1. For purposes of this test, copy the vendor daemon `demo.exe` and its associated library `demo_libFNP.dll` from the `<sdk_platform>` directory to the `<sdk_platform>\lmadmin\demo` directory.
2. From the `<sdk_platform_dir>\lmadmin` directory, enter the following to import the `demo` vendor daemon:

   ```
   lmadmin -import demo\demo.lic -force
   ```

3. Enter the following to start the license server in the background:

   ```
   lmadmin -adminOnly no -allowRemoteStopServer yes
   ```

Run the Sample Java Application

Task ***To run the sample FlexEnabled Java application***

From the `<sdk_install_dir>\examples` directory, enter the following:

```
java BasicExample f3 3.0 @host
```

where `host` is the name of the machine on which the license server was started in Chapter 3, "Start the License Server."

When you run the sample application, it will check out a license for version 3.0 of feature **f3**. Press **Enter** to check the license back in.

Stop the License Server

Task ***To stop the license server***

From the `<sdk_platform>` directory, enter the following:

```
lmdown -c .\lmadmin\demo\demo.lic
```

You have completed the demo of FlexNet Publisher Licensing Toolkit for Java on Windows. When you are ready to incorporate your production vendor keys into FlexNet Publisher Licensing Toolkit for Java, see Incorporating Toolkit for Java Into Your Product. Remember to set `LM_STRENGTH` to an appropriate value before you build your production FlexNet Publisher Licensing Toolkit. If you want to evaluate FlexNet Publisher Licensing Toolkit for Java on UNIX, see Evaluating the Toolkit for Java on UNIX.

4

Incorporating Toolkit for Java Into Your Product

This section gives the steps necessary to:

- Configure Toolkit for Java to work with your production Toolkit. This involves setting the `CLASSPATH` environment variable and creating the `vendorInfo` vendor-specific class based on settings from your production Toolkit.
- Incorporate the calls to the Toolkit for Java API into your own Java product.
- Test your FlexEnabled Java product.
- Deploy your application to your end user.

Throughout this chapter, reference is made to several default directory locations for both the FlexNet Publisher and the Toolkit for Java. Table 4-1 lists these locations and the shortcut terminology used in this chapter for each. Locate them in your installation and note their paths for reference when reading the rest of this chapter.

Table 4-1 ▪ Toolkit for Java Contents

Shortcut	UNIX Default Location	Windows Default Location
FlexNet Publisher Directories		
sdk_platform	`<sdk_install_dir>/<sdk_platform>`	`<sdk_install_dir>\<sdk_platform>`
sdk_machind	`<sdk_install_dir>/machind`	`<sdk_install_dir>\machind`
Toolkit for Java Directories		
flmjava_java	`<sdk_install_dir>/java`	`<sdk_install_dir>\java`
flmjava_examples	`<sdk_install_dir>/examples`	`<sdk_install_dir>\examples`

You are required to complete the following before proceeding with the instructions in this section:

- Build your production licensing toolkit customized with your private, vendor-specific information. See the document titled *Development Environment Guide* for details.

  Note ▪ *A C development environment is required to build the Licensing Toolkit for Java. See the document titled Development Environment Guide and the FlexNet Publisher Release Notes for your platform's C development environment requirements.*

  All toolkit components must be generated from the same set of vendor keys (VENDOR_KEYs in lm_code.h), encryption seeds (LM_SEEDs and any existing ENCRYPTION_SEEDs in lm_code.h), and TRL keys (TRL_KEYs in lm_code.h).

- Install the Java Developers Kit.
- Install the Toolkit for Java.
- Work through the Toolkit for Java evaluation. See Evaluating the Toolkit for Java on UNIX and Evaluating Licensing Toolkit for Java on Windows.

Before proceeding, you need the following files and information from your production toolkit:

- `sdk_machind/lm_code.h`
- `<sdk_platform>/lmpubkey.h`
- `<sdk_platform>/lmseeds.h`
- the value of `VENDOR_NAME` from `sdk_machind/lm_code.h`
- The `LM_STRENGTH` setting from `sdk_machind/lm_code.h`

Configuring a Production Toolkit for Java

In order to configure the production licensing toolkit for Java, you must first define the `CLASSPATH` environment variable and then create the `vendorInfo.class` file.

Define the CLASSPATH Environment Variable

To run a FlexEnabled Java application, verify that the following components are included in your `CLASSPATH` environment variable:

- *flmjava_java*/`flexlm.jar`
- *flmjava_java*/`EccpressoAll.jar`

where `flmjava_java` is determined from the Toolkit for Java Contents table.

Make sure that the new `CLASSPATH` setting is active before proceeding.

Create the *vendor*Info.class File

Each software publisher must create a concrete subclass of `VendorInfo` containing vendor-specific information such as vendor name, encryption seeds, and vendor keys. An instance of a `VendorInfo` subclass must be associated with every `License` object. The name of the subclass is of the form *vendor*`Info.class`, where `vendor` is the vendor name as registered with Revenera.

Note ▪ *To create the subclass `vendorInfo.class`.:*

1. Copy `<sdk_machind/lm_code.h`, `<sdk_platform>/lmpubkey.h`, and `<sdk_platform>/lmseeds.h` from the `sdk_machind` directory to the `flmjava_java` directory.

   In `lm_code.h`, check that `VENDOR_NAME` is set to your vendor name, that the `LM_STRENGTH` setting has the desired value, and that the file contains your production vendor keys, encryption seeds, and TRL keys. This information is used to create a `vendorInfo` class that is used to create a license object in your FlexEnabled Java application.

2. In the `flmjava_java` directory, open `vendorInfo.bat` (Windows) or `vendorInfo.sh` (UNIX) in a text editor. Make sure the `-classname` switch specifies `vendorInfo`, where `vendor` is the name of your vendor daemon. The contents of `vendorInfo.bat` should look like:

```
java -classpath flexlmutil.jar;flexlm.jar; \
                EccpressoAll.jar \
      CreateVendorInfo -code lm_code.h -pubkey lmpubkey.h \
                       -seeds lmseeds.h \
                       -classname vendorInfo
```

3. In the `flmjava_java` directory, create the vendor-specific *vendor*`Info.java` file by executing `vendorInfo.bat` (Windows) or `vendorInfo.sh` (UNIX):

4. Compile *vendor*`Info.java` to create the class file `javac` *vendor*`Info.java`.

5. Copy *vendor*`Info.class` into the directory that contains your Java source file.

Specifying the License File Location

The most important part of integrating Toolkit for Java into your application is to correctly set the location of your the license file so it can be found at run time. This location should not be hard-coded, but should be determined at run time relative to your product's installation location. In the call to the **license** constructor, you must specify the path name of the license file. The following forms can be used to specify the license path:

- A single file, usually with the `.lic` extension. However, any extension can be used.
- *port@host*, where *port* is the port number and *host* is the host name of a license server. *port* can be omitted if it is within the default range of 27000–27009.

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

- A three-server redundancy configuration. Define it as *port@host1,port@host2,port@host3*

- A license file in a string. Precede the license certificate string with the string

  `FlexlmConstants.LICENSE_STRING_PREFIX`

  and end it with the string

  `FlexlmConstants.LICENSE_STRING_SUFFIX`

  Multiple license file locations can be concatenated together, separated by the “&” (ampersand) character.

Add Toolkit for Java API Function Calls to Your Product

This section describes how to modify your application code to call the FlexNet Publisher Licensing Toolkit for Java client API functions. This implementation is derived from `BasicExample.java` that is shipped with the Toolkit for Java and located in the `flmjava_examples` directory. This standalone example is a good place to start learning how to integrate the Toolkit for Java into your product.

Note ▪ *A complete reference to the Toolkit for Java API Reference documentation is included in the distribution at the following location:*

- *Windows platforms:* `<sdk_install_dir>\doc\api\index.html`
- *UNIX platforms:* `<sdk_install_dir>/doc/api/index.html`

A Basic Toolkit for Java Implementation

In the simplest implementation, incorporating Toolkit for Java into your application involves these three classes:

- **vendorInfo**—Encapsulates the software vendor's encryption seeds, vendor keys, and other vendor-specific information.
- **FeatureSpecifier**—Specifies the details of the license to be checked out.
- **License**—Checks out and checks in the license.

Assuming a few exceptions that can be thrown by the Toolkit for Java methods are caught at an enclosing level, a basic implementation of Toolkit for Java involves the following steps:

1. Import the necessary classes from the `com.macrovision.flexlm` package into your code:

   ```
   import vendorInfo;
   import com.macrovision.flexlm.*;
   ```

   vendor`Info`, where *vendor* is your vendor name as registered with Revenera, is created externally. See Create the *vendorInfo.class File* for further information.

2. Instantiate a Feature Specifier for the desired feature.

   ```
   FeatureSpecifier featSpec = new FeatureSpecifier("f1", "1.0");
   ```

   In this example, the minimum specification is used to specify version `1.0` of feature `f1`. See Feature Specifiers for more details.

3. Specify the license search path used to locate license rights that entitle usage of the feature specified in Step 2.

```
String licenseSourceString = "@localhost&c:\\MyDir\\licenses";
```

This code specifies, in search order, the following two license sources:

- `@localhost`—a license server running on the system `localhost`
- `c:\MyDir\licenses`—a directory containing license files.

Note the "&" used to separate these two components.

4. Instantiate a `License` object that fully qualifies the desired feature.

```
License lic = new License(featSpec,
                          licenseSourceString,
                          new vendorInfo(), null);
```

This `License` object is created from the feature specification, the license sources to use, and the vendor-specific information.

5. Now, check out a license and allow the FlexEnabled Java application code to execute.

```
lic.checkout(1);
```

6. After the license is no longer needed, check it back in.

```
lic.checkin();
```

This checks in all instances of the feature that were checked out.

7. Make sure all Toolkit for Java methods are enclosed in a `try{}` block followed by a `catch(FlexlmException ex){/***/}` block. Alternatively, they can be in a method that throws `FlexlmException`.
8. Save and compile your source file.
9. Now you are ready to test your FlexEnabled Java application with the license server environment. Proceed to Testing Your Application.

Testing Your Application

Testing your application entails first creating a license file and then executing your application.

Create a License File

Your application needs a license file in order to test it. Generate a counted license file, my`counted.lic`, using `lmcrypt` (available in the FlexNet Publisher Licensing Toolkit). The contents of the license file must reflect the vendor and feature information that you put in the sample application.

1. In the `<sdk_platform>` directory make a copy of the file `counted.lic` and rename it to `mycounted.lic`.
2. Modify the `SERVER`, `VENDOR`, and `FEATURE` lines in my`counted.lic` to include server information and information for the features that your application checks out. Place `SIGN=0` at the end of each `FEATURE` line. See the *Programming Reference for License File–Based Licensing* for information about the allowed fields and syntax of a license file.
3. Save `mycounted.lic`.

4. In the `<sdk_platform>` directory of your Toolkit, sign `mycounted.lic` by typing **`lmcrypt mycounted.lic`**. This creates a valid license file by calculating signatures for each `FEATURE` line.
5. Move `mycounted.lic` to the directory containing your vendor daemon under the `lmadmin` installation directory.

Start the License Server

Task ***To start the license server***

1. At a command prompt from the `lmadmin` installation directory, enter the following to start the license server:

   ```
   lmadmin -adminOnly no -allowRemoteStopServer yes
   ```

2. Close the command window.
3. To access the `lmadmin` user interface, open your Web browser, and enter the following URL:

   ```
   http://<server>:8090
   ```

   where

   - `<server>` is the system name where the license server is running.
   - `8090` is the port for the `lmadmin` user interface. If this is not the port you have assigned, change it to the correct one.

4. Select the Administration page, and sign in with your administrator credentials.
5. Go to the Vendor Daemon Configuration tab, and click **Import License**.
6. Click **Browse** to select the license file `mycounted.lic`.
7. Click **Import License** to import the vendor daemon.

Execute Your FlexEnabled Application

Task ***To execute your application and check out a licence***

1. Make sure the following components are included in your `CLASSPATH` environment variable:

   flmjava_java/flexlm.jar
   flmjava_java/EccpressoAll.jar

2. Start your application, and perform a task that checks out one of its features.
3. Stop your application and the license server.

Deploying your Product

To ensure that your end users can use your FlexEnabled Java application easily and successfully, use the information in this section as a guideline for distributing your FlexEnabled Java application to the end user. When you ship your FlexEnabled Java application, you need to include FlexNet Publisher Licensing Toolkit components in your software installation kit in addition to the files that you normally ship. FlexNet Publisher Licensing Toolkit Components for Distribution lists these additional components. Note that the components come from both the FlexNet Publisher and the Toolkit for Java.

Table 4-2 ▪ FlexNet Publisher Licensing Toolkit Components for Distribution

Component	Description
FlexNet Publisher Licensing Toolkit Components from the *sdk_platform* directory	
`lmadmin` or `lmgrd` (`lmadmin.exe` or `lmgrd.exe` on Windows)	The license server manager and vendor daemons. Include these if you distribute either counted or served-uncounted licenses.
vendor`.exe` (Windows) or *vendor* (UNIX)	
`lmutil.exe` (Windows) or `lmutil` (UNIX)	Used for license management and license server administration.
`lmtools.exe` (Windows only)	GUI front end to `lmutil.exe`.
`lmgr11.dll` (Windows) or `liblmgr11.so` (UNIX)	The client DLL or shared object used to support native hostid capabilities for FlexEnabled clients built with the Java toolkit.
FlexNet Publisher Licensing Toolkit for Java Components	
vendor`Info.class`	Subclass of the FlexNet Publisher Licensing Toolkit for Java `VendorInfo` superclass containing vendor-specific information.
`flexlm.jar`	Toolkit for Java classes.
`EccpressoAll.jar`	Toolkit TRL classes.

In addition to the components in Table 4-2, your customer eventually needs a license file.Depending on your internal policies, this can be shipped with your installation kit or shipped separately.

5

The Licensing Toolkit for Java API Programming Model

To entitle a given feature of your FlexEnabled Java application, the application needs to successfully check out a license for that feature. As such, the nucleus of the Licensing Toolkit for Java object model is represented by the `License` object. This object represents a specific usage of a feature. It encapsulates the following:

- The details of a feature (name, version, and so on) via a **FeatureSpecifier** object

  An application must specify the name of the feature to check out and the minimum acceptable version number of that feature. The application specifies this by creating a **FeatureSpecifier** object and passing that to one of the License constructors. See Feature Specifiers for more details.

- The count of the feature to check out

- The list of `LicenseSources` from which to try to check out

  Licenses can be checked out from license servers or from license files, which may refer to license servers. An array of `LicenseSource` objects is associated with each `License` instance by the constructor. One variant of the constructor accepts a `String` containing one or more license source specifiers, which are converted to an array of `LicenseSources` by the constructor. When checking out a license, the license source specified by each element of the array is tried, in order, to satisfy the checkout request. See License Sources for further details.

- The `LicenseSource` actually used for the checkout

  Over the course of its lifetime, a `License` may be checked out and back in many times, potentially from different `LicenseSource` objects. A `License` object can be associated with multiple `LicenseSource` objects over its lifetime, one at a time. The list of potential license sources with which it can be associated is established by the constructor. After a successful checkout, the actual `LicenseSource` instance that satisfied the checkout can be determined with the **getLicenseSource** method. Once a single `LicenseSource` is established in a `License` instance, it remains established as the sole `LicenseSource` for the life of the `License` instance.

- The checkout-data string associated with this `License`, if any

  A checkout-data string is used to label a feature checkout from a specific `License` object or to qualify a feature checkin request. The same checkout-data string can be associated with feature checkouts from multiple `License` objects. The checkout-data string can then be used to refer to all `License` objects using the same string, there by making it convenient to check in all licenses for the given feature with one invocation to the **checkin** method. See Checkout-Data Strings for more information.

Licensing Toolkit for Java Framework

This section covers the underlying framework for Licensing Toolkit for Java.

Feature Specifiers

In order to check out a license, the application must specify some details about the license to be checked out. At a minimum, an application must specify a feature name and the lowest acceptable version number of that feature. An application can specify additional details of the required license if it needs to. The application uses the `FeatureSpecifier` object to do this.

License Sources

A license is checked out from a license source, which can be any of the following:

- License file
- License server
- License in a string
- Local (client machine) trusted storage

In a basic implementation example, a list of license sources is specified from which to attempt a license checkout. This list is used to create an array of `LicenseSource` objects on behalf of the application.

An array of `LicenseSource` objects is associated with each License instance by the `License` constructor. One variant of the `License` constructor accepts a `String` containing one or more license source specifiers, which are converted to an array of `LicenseSource` objects by the constructor. When checking out a license, the license source specified by each element of the array is tried, in order, to satisfy the checkout request. The checkout may be satisfied by any one of the `LicenseSource` objects in the array, but only from one—not multiples.

After a successful checkout, the actual `LicenseSource` instance that satisfied the checkout can be determined with the **License.getLicenseSource** method. A `License` object can be associated with multiple `LicenseSource` objects over its lifetime, one at a time. The list of potential license sources with which it can be associated is established by the constructor.

Normally an application will leave the creation and management of `LicenseSource` objects to the `License` class. However, if the application wishes, it can manage `LicenseSource` objects itself. Variants of the `License` constructor take one, or an array of, existing `LicenseSource` objects as parameters. Alternatively, the **License.createLicenseSource** method can be used to create a single license source.

A `License` object created this way will try to check out a license from the supplied `LicenseSource` objects in the order specified. An application can create a `LicenseSource` object to be used in future `License` object instantiations by using the **LicenseSource.createLicenseSource** method. Also, an application can get the `LicenseSource` object associated with any given `License` object via the **License.getLicenseSource** method.

The following code demonstrates how to manage `LicenseSource` objects:

```
import vendorInfo;
import com.macrovision.flexlm.License;
import com.macrovision.flexlm.FeatureSpecifier;
import com.macrovision.flexlm.LicenseSource;
```

```
.
.
.
try
{
    // Set up specifications of features to check out
    FeatureSpecifier featSpec1 = new FeatureSpecifier("f1", "1.0");
    FeatureSpecifier featSpec2 = new FeatureSpecifier("f2", "1.0");

    // Set the license source string by hand.
    String licenseSourcesString =
        "7314@license.someserver.com&/opt/mypackage/licenses";

    // Create a VendorInfo instance for use with both
    // feature checkouts
    vendorInfo vInfo = new vendorInfo();

    // Create a license object for feature "f1" and check it out
    License lic1 = new License(featSpec1, licenseSourcesString, vInfo);
    lic1.checkout(1);

    // Get the license source used for the f1 checkout
    // and use it for checking out the f2 license.  Note that
    // we're invoking a License constructor that takes a
    // LicenseSource object this time (instead of a
    // String naming one or more LicenseSources).
    LicenseSource ls = lic1.getLicenseSource();
    License lic2 = new License(featSpec2, ls);
    lic2.checkout(1);
    .
    .
    .
    lic1.checkin();
    lic2.checkin();
{
catch (FlexlmException e)
{
System.out.println(e.getMessage());
System.exit(1);
}
```

The VendorInfo class

Each software publisher must create a concrete subclass of `VendorInfo` containing vendor-specific information such as vendor name, encryption seeds, and vendor keys. An instance of a subclass of a `VendorInfo` must be associated with every `License` object.

This class is created by a special utility program provided by Licensing Toolkit for Java that reads a data file containing the required information and writes a Java source file with the `VendorInfo` subclass. See Create the *vendorInfo.class File* for further information.

Connection Data

In some cases, an application may want to set connection-specific information that must be specified as part of the initialization of a connection to a license server, such as specifying a user name or heartbeat parameters. The application does this by creating an object that implements the `ConnectionData` interface and passing in that object when creating a `LicenseSource` or `License` object. The methods in the `ConnectionData` object are called on behalf of the application at the appropriate time during connection initialization.

IP Address Override

The FlexNet Publisher Licensing Toolkit enables you to override the IP address of the client requesting the license for a specific feature. To support this IP-address-override functionality in a Java-based application, add the following API to your Java application code:

```
lic.setipaddress("<IP_address>");
```

where **`<IP address>`** is the value used in the `FEATURE` line or options file.

For more information, see the description for the `LM_A_INTERNET_OVERRIDE` attribute in the *C/C++ Function Reference*.

Note ▪ *Add this API to the sample application,* `BasicExample.java`*, which ships with the Licensing toolkit for Java, if you want to test the IP-address-override functionality with the sample.*

Count

The **License.checkout** method checks out the specified count of the associated `Feature`. After the checkout method is called successfully, it may be called again any number of times with any count specified. The count is treated as the total complement of licenses required.

- If the count specified is the same as or lower than the count currently checked out, no additional licenses are checked out.
- If the count specified is higher than the number currently checked out, the `License` object attempts to check out just the additional count to meet the requested complement of licenses.

In this way, license server keeps a *high watermark* of the requested counts checked out.

Checkout-Data Strings

Up to 32 bytes of vendor-supplied data may be specified via **License.setCheckoutData** as a checkout-data string. As a label, this string is associated with checkouts and checkins of features from license servers. If no checkout-data string is specified, the `License` instance itself associates a unique checkout-data string with the checkout. This eliminates the need to write special code such as is necessary in non-Java applications to be able to check in a subset of the count checked out on a single license server connection.

Consider this example: an application wanting to check out a count of 10 of a given feature in two sets of five each, such that each set could be checked in independently on the same license server connection, can simply create two `License` instances that each check out a count of five. The `License` object's default management of checkout-data strings makes this possible.

Licensing Toolkit for Java does not allow checkouts in multiple `Feature` instances having any of the following:

- Same feature name in its `FeatureSpecifier`
- Same checkout-data string
- Same `LicenseSource`

Due to the way the license server behaves, allowing such checkouts could create ambiguities in the `License` instances. If a `License` instance attempts a checkout of the same feature, checkout data, and `LicenseSource`, it throws an exception.

A checkout-data string can also be used as a duplicate grouping criterion, in which case the application will want to manage checkout data strings itself.

Important ▪ *"`LM_A_MULTIPLE_CHECKOUT_DATA`" is not supported in the JAVA kit.*

See `LM_A_CHECKOUT_DATA` in the *C/C++ Function Reference* for more information about checkout-data strings.

Heartbeats

Heartbeats are messages exchanged between a FlexEnabled application and a license server for the purpose of assuring each that the other is still alive and healthy. By default, heartbeats are sent and received automatically by the FlexEnabled Java product. The only action necessary on the product's part is to provide methods to invoke when something in the heartbeat process goes wrong, for example, if the application can no longer communicate with the license server.

Heartbeats can be automatic, manual, or semi-automatic, the distinction being how the timing of heartbeat messages is managed and how reconnection attempts are recognized and handled. Table 5-1 summarizes these distinctions.

Table 5-1 ▪ Heartbeat Summary

Type	Heartbeat Timing	Reconnection Management	Considerations
Automatic	FlexNet Publisher Licensing Toolkit creates and manages a dedicated thread to send and receive heartbeat messages at a specified interval on behalf of the application. The application controls the interval.	Reconnection to license server is handled automatically and transparently.	• Requires the application be multithreaded. • Simplest method to implement. No setup required by the application. • Dedicated heartbeat thread, rather than the application, is automatically interrupted.

Table 5-1 ▪ Heartbeat Summary

Type	Heartbeat Timing	Reconnection Management	Considerations
Semi-automatic	Control of heartbeat timing regulated by the frequency with which the application invokes a particular method.	Reconnection to license server is handled automatically and transparently.	• No multithreaded requirement. • Requires application to setup heartbeat parameters. • Total control of interrupts to the application.
Manual	Control of heartbeat timing regulated by the frequency with which the application invokes a particular method.	Application detects reconnection attempts from the license server and determines how to respond.	• No multithreaded requirement. • Requires application to setup both heartbeat and reconnection parameters. • Total control of interrupts to the application.

An application may wish to temporarily suspend the sending of heartbeats so that the license server's timeout mechanism can reclaim a license that is not in active use. The `Heartbeat.setHeartbeatBlocked(boolean)` method provides a way for the application to specify the blocking and unblocking of heartbeats. Heartbeats are unblocked by default.

Automatic Heartbeats

By default automatic heartbeats are set up by FlexNet Publisher Licensing Toolkit for Java at license checkout time with the following default parameters:

- A heartbeat is sent to the license server every 120 seconds.
- Reconnecting to a lost license server tried every 60 seconds for up to five attempts.
- If connection is not re-established after five retries, **System.exit** is invoked to terminate the Java virtual machine.

If an application wishes to have different heartbeat and reconnect parameters, it can create a `LicenseSource` or `License` object using a customized **ConnectionData.initHeartbeat** method that calls the desired set methods in the `Heartbeat` object passed to it.

There are three possible outcomes with automatic heartbeats when reconnection is attempted after the connection to the license server is lost:

- Reconnection to the license server succeeded.
- Reconnection to the license server was attempted and failed, but more attempts will be made.
- Reconnection to the license server was unsuccessful after the retry limit was reached.

An application can supply code to execute at one or more of these junctures by providing an object implementing the `HeartbeatListener` interface and making it known via the **Heartbeat.addHeartbeatListener** method.

As an example, the code below implements **HeartbeatListener.reconnectFailed** method via a `HeartbeatAdapter` object to define the behavior in the event the reconnection retry limit is reached and reconnection is unsuccessful:

```
import com.macrovision.flexlm.Heartbeat;
import com.macrovision.flexlm.HeartbeatAdapter;
import com.macrovision.flexlm.HeartbeatEvent;
import com.macrovision.flexlm.LicenseSource;
import com.macrovision.flexlm.Feature;
.
.
.
ConnectionData connData = new ConnectionDataAdapter()
{
    public void initHeartbeat(Heartbeat hb)
    {
        hb.addHeartbeatListener(new HeartbeatAdapter()
        {
            public void reconnectFailed(HeartbeatEvent e)
            {
                // Get the name of the feature this event applies
                // to, and call the fictitious routine
                // "lostLicense()" to handle it from here.
            String featName = e.getHeartbeat().getFeature().
                              getName();
            lostLicense(featName);
            }
        });
    }
};
License lic = new License(featSpec,
                          licenseSourceString,
                          new vendorInfo(),
                          connData);
//See A Basic Toolkit for Java Implementation
//for definitions to featSpec, licenseSourceString,
//and vendorInfo.
```

Manual Heartbeats

Manual heartbeats are used instead of automatic heartbeats if the application:

- Wants to be in control of the precise timing of heartbeats.
- Cannot tolerate the extra thread introduced by automatic heartbeats.
- Cannot be multithreaded.

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

To set up manual heartbeats, the application needs to do the following.

Table 5-2 ▪ Manual Heartbeat Procedures

Step	Action	Description
Step 1	Call **Heartbeat.setHearbeatInterval(int)** with a value of zero from its **ConnectionData.InitHeartbeat** method.	It can store the `Heartbeat` object passed to that method, or later it can test its `LicenseSource` object to see if it is an instance of `Heartbeat` and cast to it.
Step 2	Invokes **Heartbeat.sendHeartbeat** at whatever interval is deemed appropriate.	The **sendHeartbeat** method returns the number of unacknowledged heartbeats since the last successful one.
Step 3	Subsequently, as needed, invoke **Heartbeat.getReconnects(int)** to determine the number of reconnections over an application-supplied time interval.	This value and the return value from **sendHeartbeat** are helpful to the application in determining the health, over time, of the license server and the probability that license servers are being restarted maliciously.

Semi-Automatic Heartbeats

Semi-automatic heartbeats combine aspects of automatic heartbeats and manual heartbeats. With semi-automatic heartbeats, the application is in control of precise heartbeat timing (as with manual heartbeats), but the scheme of HeartbeatListeners to notify the application of HeartbeatEvents (used with automatic heartbeats) applies.

To set up semi-automatic heartbeats, the application needs to do the following.

Table 5-3 ▪ Semi-Automatic Heartbeat Procedures

Step	Action
Step 1	Initialize the heartbeat interval to 0 as with manual heartbeats.
Step 2	Implement a `HeartbeatListener` interface, as with automatic heartbeats.
Step 3	Call **Heartbeat.doHeartbeat**.

The **Heartbeat.doHeartbeat** sends a heartbeat and checks for a response just like **Heartbeat.sendHeartbeat** does with manual heartbeats, but it also calls the `HeartbeatListener` methods as with automatic heartbeats. The **doHeartbeat** method returns the same value as **sendHeartbeat**, so you can use that value along with the **getReconnects** method in the same manner as for manual heartbeats.

Hostids

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

- **A license**—The license is bound to one or more hostids. They are used to define which systems are licensed to run your product. These licenses are said to be *node-locked* to a specific machine.
- **A license server**—A hostid used to define which system is authorized to run a license server that serves licenses to your FlexEnabled Java product.

In the context of a license server, any hostid type can be used as long as it is supported by FlexNet Publisher Licensing Toolkit. However, for node-locked licenses, the application is responsible for determining the hostid of the system on which it is running. Since FlexNet Publisher Licensing Toolkit for Java is a pure-Java package, only a few hostid types are supported—those not requiring a platform-dependent Java implementation on which the application is running. Available hostids include:

- `DEMO`
- `ANY`
- `HOSTNAME`
- `INTERNET`
- `USER`
- `PHY_HOSTNAME`
- `PHY_INTERNET`

In addition, hostid lists are supported, so you can node-lock a license to a list of hostids. For example,

```
HOSTID="INTERNET=1.2.3.4 INTERNET=1.2.3.5 HOSTNAME=foo"
```

Wildcards are supported in address octets. If you choose to issue uncounted, node-locked licenses, we recommend that they expire.

When using HOSTID in a feature line, you can use a wildcard character in the domain name. As an example, the following value allows the feature to be checked out by any hostname in the revenera domain.

```
HOSTID=HOSTNAME=*.revenera.com
```

FlexNet Publisher Licensing Toolkit for Java also provides indirect support for the full complement of native hostid types via the **VendorInfo.useNativeHostIds** method. Additionally, a hook is provided for vendor-defined hostids.

Native Hostids

Native hostids are provided transparently via Java Native Interface (JNI) hostid integration, and can be enabled with a single method invocation. By shipping platform-dependent native code, it is simple to use the non-Java FlexNet Publisher Licensing Toolkit code to provide hostid capability to FlexEnabled Java products. The **getNativeHostId** method fetches the native hostids. To fetch the native hostids, input the flex defined hostid types.

Note ▪ *This feature requires a FlexNet Publisher Licensing Toolkit. To build a toolkit with dongle support use the following command: `make -f makefile DONGLE=1`. For more details on building a kit, refer to the Development Environment Guide.*

The following sections outline how to implement native hostid support.

Microsoft Windows Platforms

Task ***To implement native hostid support on Windows platforms:***

1. Build the FlexNet Publisher Licensing Toolkit. Refer to the *Development Environment Guide* for details on building the kit.
2. Build `lmgr11.dll` in the FlexNet Publisher Licensing Toolkit. To do so, enter the following from the `<sdk_platform>` directory in your toolkit installation:

   `nmake -f makefile DLL=1` to build `lmgr11.dll` without dongle support

   or

   `nmake -f makefile DLL=1 DONGLE=1` to build `lmgr11.dll` with dongle support

   This client DLL contains the hostid-object code that you must ship with your product.
3. In the Java virtual machine's `$PATH` environment variable, include the path to the folder containing `lmgr11.dll`. This is how the Java Virtual Machine locates code invoked through JNI.
4. In the Java source code, invoke the **useNativeHostIds** method on any `VendorInfo` subclass instances that are created by the application code.

UNIX Platforms

Task ***To implement native hostid support on UNIX platforms***

1. Build the FlexNet Publisher Licensing Toolkit. Refer to the *Development Environment Guide* for details on building the kit.
2. Build `liblmgr11.so` in the FlexNet Publisher Licensing Toolkit. To do so, enter the following from the `<sdk_platform>` directory in your toolkit installation:

   `make shared_object`

   This shared object contains the C functions required to support Java native hostids.
3. Set the `LD_LIBRARY_PATH` system environment variable to include the path to `liblmgr11.so`.
4. In the Java source code, invoke the **useNativeHostIds** method on any `VendorInfo` subclass instances that are created by the application code. See the online API documentation for details on the **VendorInfo.useNativeHostids** method.

Vendor-Defined Hostids

A vendor-defined hostid hook is provided for applications that wish to use hostid types beyond those provided either directly by FlexNet Publisher Licensing Toolkit for Java or indirectly through the JNI interface. To use vendor-defined hostids, the application creates a subclass of the abstract `HostId` class. The subclass must implement several methods, including methods to do the following:

- Determine the value of the vendor-defined hostid of the current system
- Parse a hostid string and return a new object of this vendor-defined type
- Compare the hostid encapsulated in a `HostId` object to the hostid of the vendor-defined type of the current machine

A vendor-defined hostid type is registered at runtime via the **VendorInfo.registerVendorInfo** method.

Exceptions

The FlexNet Publisher Licensing Toolkit for Java API defines several exceptions. Each of the exceptions contains major and minor error numbers as in the FlexNet Publisher Licensing Toolkit, and the major error numbers are the same as those in the FlexNet Publisher Licensing Toolkit where possible. Table 5-5 lists the major error numbers that can be set by FlexNet Publisher Licensing Toolkit for Java API methods.

Various subclasses of the base exception class `FlexlmException` contain additional data describing the exception. The **FlexlmException.getMessage** method returns a localized error message according to the supplied `Locale` object and subject to the presence of a localized error message catalog file. An English version of this catalog is included in FlexNet Publisher Licensing Toolkit for Java as `FlexlmResources.properties`. This catalog is a sequence of `error_code=string values`. To localize the message catalog, create a separate properties file following the naming conventions outlined in the `java.util.ResourceBundle` class.

Task ***To implement vendor-defined hostids***

Ensure that you have started FlexJava Setup. This procedure is applicable for both counted and uncounted licenses.

1. Open the `BasicExample.java` file and uncomment the following lines: (for trusted storage–based licensing edit `TSExample.java`)

   ```
   try {
   vInfo.registerVendorHostId(new MyHostId("VDH=123"));
   }
   catch (FlexlmException e) {
   System.out.println(e.getMessage());
   System.exit(1);
   }
   ```

   Note ▪ *If there exists a line "`vInfo.useNativeHostIds(true, null);`" in BasicExample.java file and TSExample.java file, then comment the line, as this will lock the HOSTID to lmhostid value and not VDH value.*

2. Open `MyHostid.java` file and change the return value of the last line to "123" and save the file.
3. Compile `BasicExample.java` file for license file–based licensing or use `TSExample.java` file for trusted storage–based licensing.
4. Now encrypt the license file.

   ```
   SERVER this_host VDH=12345678 //Value present in vendor_hostid.c
   VENDOR <vendor_name>
   INCREMENT vd2 <vendor_name> 1.0 permanent 1 HOSTID=VDH=123 SIGN=0
   ```

5. Start the license server.

6. Checkout vd2 using *javaclient(BasicExample* or *TSExample)*.
7. Shut down the license server.

FlexNet Publisher Licensing Toolkit for Java API

The FlexNet Publisher Licensing Toolkit for Java API is defined by the `com.macrovision.flexlm` package. Figure 5-1 shows the major classes and interfaces comprising the `com.macrovision.flexlm` package (shaded background) and their relationship to each other and other Java components.

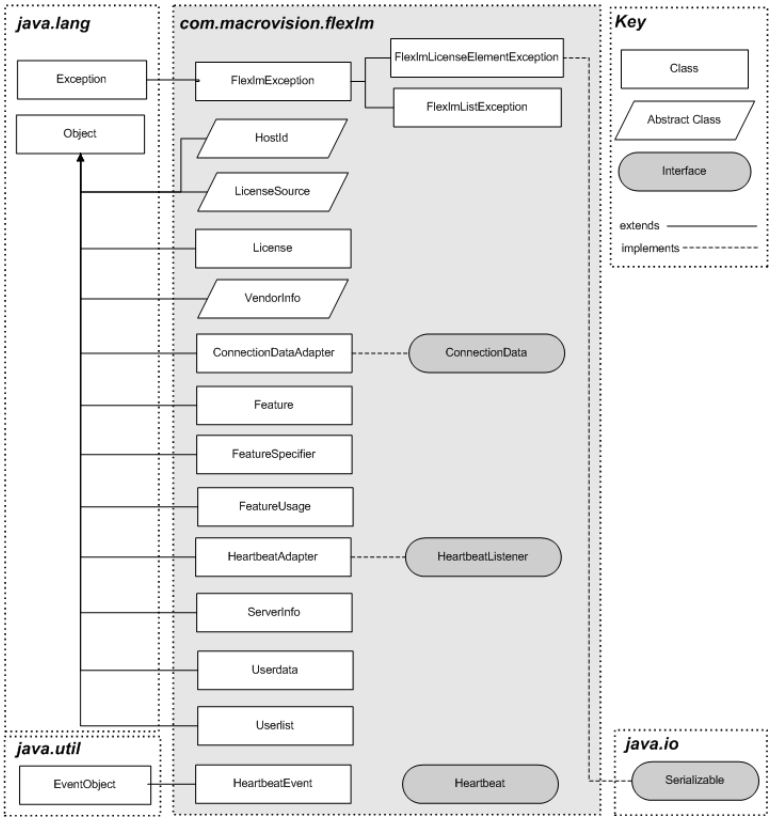

Figure 5-1: The `com.macrovision.flexlm` Package Hierarchy

Table 5-4 lists components contained in the `com.macrovision.flexlm` package. Definitions for each interface, class, and exception are located in the FlexNet Publisher Licensing Toolkit for Java distribution at the following location.

- Windows platforms: `<sdk_install_dir>\Documents\api\index.html`
- UNIX platforms: `<sdk_install_dir>/Documents/api/index.html`

Table 5-4 ▪ The `com.macrovision.flexlm` Package

Name	Description
INTERFACES	
ConnectionData	Provides license server connection data.
FlexlmConstants	Constants used throughout the `com.macrovision.flexlm` package.
Heartbeat	Controls the heartbeats for a license source
HeartbeatListener	Provides callbacks for automatic heartbeats.
CLASSES	
ConnectionDataAdapter	Provides a default implementation of the `ConnectionData` interface.
Feature	Describes a FEATURE or INCREMENT line.
FeatureSpecifier	Specifies the details of the feature to be checked out.
FeatureUsage	Provides feature usage data items in response to **LicenseSource.getFeatureUsage**.
HeartbeatAdapter	Provides a default implementation for the `Heartbeat` interface.
HeartbeatEvent	Encapsulates the data passed to a `HeartbeatListener` object.
HostId	The superclass for providing vendor-defined hostids.
License	Represents the specific usage of a feature.
LicenseSource	The superclass of all classes from which `Feature` objects can be checked out.
ServerInfo	Information describing a license server's characteristics, primarily those expressed in the vendor daemon configuration file, `lsvendor.c`.
Userinfo	Encapsulates all the items known about a specific use of a feature checked out from a license server.
Userlist	Encapsulates data enumerating information known about each specific current use of a specified feature on a license server.
VendorInfo	The superclass for vendor-specific information.
EXCEPTIONS	

Table 5-4 ▪ The `com.macrovision.flexlm` Package (cont.)

Name	Description
FlexlmException	Encapsulates the FlexNet Publisher Licensing Toolkit major and minor error numbers. See Table 5-5 for a list of major error numbers.
FlexlmLicenseElementException	Encapsulates exceptions that occur with a syntactically invalid license.
FlexlmListException	Encapsulates multiple `FlexlmException` objects into a list.

FlexNet Publisher Licensing Toolkit for Java Error Code Values

This section lists the error code values that are contained in the FlexlmException object; they are defined in `com.macrovision.flexlm.FlexlmResources.properties`. Values starting at -500 are specific to FlexNet Publisher Licensing Toolkit for Java. Values between -1 and -100 are defined in the FlexNet Publisher Licensing Toolkit C implementation and are included in this table for your convenience.

Table 5-5 ▪ FlexNet Publisher Licensing Toolkit for Java Error Codes

Value	Symbolic Name	Description
-1	`LM_NOCONFFILE`	Can't Find License File
-3	`LM_NOSERVER`	No License Server System For This Operation
-4	`LM_MAXUSERS`	All Licenses In Use
-5	`LM_NOFEATURE`	No Such Feature Exists
-8	`LM_BADCODE`	Authentication Failed
-9	`LM_NOTTHISHOST`	Feature Is Locked To A Different Host
-10	`LM_LONGGONE`	Feature Is Expired
-12	`LM_BADCOMM`	Invalid Message Received From License Server System
-14	`LM_BADHOST`	Unknown Host
-15	`LM_CANTCONNECT`	Can't Connect to License Server System
-16	`LM_CANTRECEIVE`	Can't Receive Data From License Server System
-17	`LM_CANTSEND`	Can't Send Data To License Server System
-18	`LM_NOSERVSUPP`	License Server System Does Not Serve This Feature

Table 5-5 ▪ FlexNet Publisher Licensing Toolkit for Java Error Codes

Value	Symbolic Name	Description
-20	LM_SERVBUSY	License Server System Busy (No Majority)
-22	LM_CHECKINBAD	Checkin Failed
-23	LM_BUSYNEWSERV	License Server System Temporarily Busy (New Server Connecting)
-24	LM_USERSQUEUED	Users Are Already Queued For This Feature
-25	LM_SERVLONGGONE	Server Does Not Support Feature Version Requested
-30	LM_NOREADLIC	Cannot Read License File
-31	LM_TOOEARLY	Start Date Not Yet Reached
-34	LM_CLOCKBAD	Clock difference too large between client and license server system
-35	LM_FEATQUEUE	Queue Entered
-38	LM_FEATEXCLUDE	User or Host is On The EXCLUDE List For This Feature
-39	LM_FEATNOTINCLUDE	User or Host is Not on the Options File INCLUDE List
-41	LM_BADPARAM	Invalid Parameter Passed To Method
-44	LM_BADKEYDATA	Invalid FlexNet Publisher Licensing Toolkit key data supplied
-48	LM_BADPLATFORM	FlexNet Publisher Licensing Toolkit platform not enabled
-50	LM_EXPIREDKEYS	FlexNet Publisher Licensing Toolkit key data has expired
-63	LM_NOTLICADMIN	License Administrator Privilege Required
-64	LM_REMOVETOOSOON	Attempt to Force Checkin Before Minimum Interval Elapsed
-77	LM_BAD_VERSION	Invalid Version Number Syntax
-82	LM_BADPKG	Invalid PACKAGE Line Syntax
-83	LM_SERVOLDVER	FlexNet Publisher Licensing Toolkit version of the Licensing client library newer than server
-84	LM_USER_BASED	USER_BASED or HOST_BASED License, But No Users or Hosts Included in Options File
-88	LM_BADSYSDATE	System Date Has Been Set Back
-89	LM_PLATNOTLIC	This Platform Is Not Enabled By the License

Table 5-5 ▪ FlexNet Publisher Licensing Toolkit for Java Error Codes

Value	Symbolic Name	Description
-92	LM_SERVER_REMOVED	Server Removed During Re-Read, or Server HostId Mismatch With License
-101	LM_RESVFOROTHERS	All licenses are reserved for other users
-500	LM_BADPORT	Invalid Server Port Number
-501	LM_BADINTEGER	Invalid Value In License Where An Integer Was Expected
-502	LM_BADCOUNT	Invalid Value Supplied for Count
-503	LM_BADHOSTID	Invalid Hostid Supplied in License
-504	LM_BADHOSTIDTYPE	Invalid HostId Type Supplied
-505	LM_BADFEATURESYNTAX	Bad Feature Line Syntax
-506	LM_INTERNALERROR	Internal FlexNet Publisher Licensing Toolkit Error
-507	LM_BADDATE	Bad Date Format in License File
-508	LM_BADSERVERLINE	Bad SERVER Line
-509	LM_BADLICSTRING	Bad License String
-510	LM_FEATUREBADONSERVER	Server's Feature Doesn't Authenticate On Client Side
-511	LM_NOLICENSEOUT	No License Checked Out
-512	LM_LICENSEOUT	License Already Checked Out
-513	LM_ERRORLIST	Error List Returned
-514	LM_NOCERTICOM_MODULE	No Certicom Module Available
-515	LM_BADCERTICOM_MODULE	Wrong or Incomplete Certicom Module
-516	LM_SIGNREQUIRED	SIGN or SIGN2 Required in License Certificate
-517	LM_NOLICENSESOURCE	Feature Object Has No LicenseSources
-518	LM_SAMELICENSE	An Identical License Is Already Checked Out On This License Source
-519	LM_CHECKOUTQUEUED	This License Has An Asynchronously Queued Checkout Pending
-521	LM_NONATIVELIB	Library for Native Hostid Couldn't Be Loaded

Table 5-5 ▪ FlexNet Publisher Licensing Toolkit for Java Error Codes

Value	Symbolic Name	Description
-522	LM_WRONGVENDOR	Already Connected to Another Vendor Daemon
-523	LM_NOUSER	No Such User, Host, or Display
-524	LM_SHUTDOWNFAILED	Shutdown of License Server System Failed
-525	LM_ALREADY_CONN	Shutdown Failed - Already Connected To License Server System
-526	LM_BADSOURCESPEC	Invalid License Source String
-527	LM_SWITCHERR	Log File Switch Error
-528	LM_CANTWRITEFILE	Cannot Write License File
-529	LM_BADOPTFILESYNTAX	Bad Options File Syntax
-530	LM_NATIVE_CALL_FAILED	Native Method Call Failed
-531	LM_NO_NATIVE_TS_LIB	Could not Load Native Library for TS Access
-532	LM_NOFEATURES	License Source Contains No Features

Glossary

The following glossary terms are used in this product.

Term	Meaning
activation specification record (ASR)	A signed XML file used with local activation and short-code activation that contains information about license rights, configuration settings for the publisher's trusted storage segment, and additional information required by the activation transaction.
activatable license	A license right in trusted storage that can be activated to a FlexEnabled client. See also license group.
activation	In the FlexNet Publisher Licensing Toolkit, this is a process used to load license rights into trusted storage. Different activation types include: ● programmatic activation ● manual activation ● short-code activation ● local activation In FlexNet Operations, this is a process used to generate license rights in either trusted storage and in license files.
activated feature line	See feature definition line.
Activation API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that enable you to initiate transactions and manage license rights in trusted storage. The Activation API is used to develop the functionality in an activation utility.
activation client	The system that works with an activation server to receive license rights. This system can be either a license server or a FlexEnabled client. The activation client initiates an activation, repair, or return transaction with an activation server, using an activation utility. The following transaction types can be initiated by an activation client: ● activation transaction ● return transaction ● repair transaction
activation request	A message sent by an activation utility to an activation server that initiates an activation transaction.

Term	Meaning
activation response	A message sent from an activation server to an activation client, in response to an activation request. This message defines the license rights.
activation server	A system that responds to requests for license rights from an activation client. The activation server can be one of the following: ● a license server ● FlexNet Operations ● in a test environment, the Manual Response Generator (responsegen.exe) An activation server can also perform repair and return transactions.
activation transaction	A sequence of activation request and activation response messages, and related actions that result in license rights being loaded into trusted storage.
activation utility	A utility that initiates an activation transaction, repair transaction, or return transaction by sending a request to an activation server. This utility also processes the response received from the activation server. It uses the Activation API to perform these capabilities. It is written by the software publisher to meet specific business process requirements. This application can be either separate from or embedded in the FlexEnabled application.
anchor	A link between the system and the trusted storage that is used to detect whether trusted storage has been tampered with, deleted, or restored. Anchors are also used to ensure that information in trusted storage about trial licenses persists beyond the expiration date.
ASR file	See activation specification record (ASR).
binding	A capability used with trusted storage where one or more properties of a system, combined together into a unique identifier, are used to ensure that license rights held in trusted storage are not copied to another system.
borrowing	A capability available in FlexEnabled applications that enables a user to temporarily obtain a license in such a way that the license can be used locally for a limited period of time while the user is disconnected from the local network.
bulk entitlement	An entitlement created for an order of multiple copies of a single product or suite. Bulk entitlements are used for sales through reseller channels.
checkin	The transaction that a FlexEnabled application performs to return a license after it has been checked out. This operation is not the same as an activation transaction. FlexEnabled applications must perform a checkout and checkin transaction to use a license.

Term	Meaning
checkout	The transaction that a FlexEnabled application performs to request a license. This transaction is different than activating a license. FlexEnabled applications must perform a checkout and checkin transaction to use a license.
client system	A general term used to define any system that interacts with a server in a client-server relationship. When using the FlexNet Publisher Licensing Toolkit there are two types of clients: ● activation client ● FlexEnabled client
composite transaction	Available in FlexNet Publisher 11.8, composite transactions allow multiple actions to be combined into a single transaction with FlexNet Operations. Also called multi-action or V2 transactions. From FlexNet Publisher 11.14.0, composite transactions are also supported for all license servers.
concurrent license	A license that can be shared among users by allowing each user to check out and then return the license. These licenses are served to FlexEnabled applications from a license server. If all licenses are being used, an additional user cannot run the FlexEnabled application until one of the other users check in their license. When one user finishes using the license, another user can begin using it. Also called floating license or served license rights.
config response	A response to any request from an activation client that contains information about how the publisher's trusted storage segment must be configured (called the trusted storage segment configuration). This response is processed by the activation utility and results in a segment of trusted storage being created or re-configured for that publisher.
consumer	An individual or small business who purchases a license for their individual use. Compare with enterprise.
counted	A type of license right or license model that defines a multiple quantity of licenses and allows concurrent use of the licenses. This term is taken from the fact that the license server counts the number of licenses that are being used.
debug log file	A file used by the license server to record status and error messages that are useful for debugging the license server. Each license server can have one or more of these files.
demo license	See trial license.
duplicate grouping	A capability in the FlexNet Publisher Licensing Toolkit used to uniquely identify a single client (or client checkout request) for the purpose of counting the number of licenses that are used. Duplicate grouping defines a set of rules under which multiple checkout requests can share the same license(s).
embedded activation	See local activation.

Term	Meaning
encryption seed	A set of values defined by a software publisher that are used when creating a publisher-specific vendor daemon.
end-user	A person who uses a FlexEnabled application. This role is not the same as a license administrator.
end-user system	See client system.
end-user license administrator	See license administrator.
enterprise	An organization that purchases licenses on behalf of a number of individual users. Compare with consumer.
entitlement	A representation of the license model and product that a customer has bought. It is a definition only. The customer will need to fulfill the entitlement so that the license rights can be populated to license files or trusted storage. In FlexNet Operations, there are two types of entitlements: ● bulk entitlement ● simple entitlement Each entitlement is uniquely identified by an entitlement ID.
entitlement ID	In the FlexNet Publisher Licensing Toolkit, an attribute in trusted storage that stores a value used to identify the customer's entitlement. In FlexNet Operations, a value that uniquely identifies the entitlement.
expiring license	License rights granted to a customer for a limited duration. When the license rights have expired, the licenses can no longer be checked out.
feature	In the context of a software application, this is a single unit of capability. In the FlexNet Publisher Licensing Toolkit, this is an identifier used to associate a license model to a unit of capability in a software application. The software publisher defines which features in their software application map to a single feature in the FlexNet Publisher Licensing Toolkit. For example, a feature in the FlexNet Publisher Licensing Toolkit could represent any of the following: ● The entire software application. ● A subset of menu items. ● A specific set of data available from the software application. ● A process in the software application. The software publisher also defines the license model for each FlexNet Publisher Licensing Toolkit feature. This gives software publishers the ability to define a different license model for each feature in the software application.

Term	Meaning
feature bundle	A collection of features in FlexNet Operations. It exists primarily to enable modularity and re-use.
feature definition line	An entry in a license file or fulfillment record that describes a feature. Each line begins with the keyword FEATURE, INCREMENT, or UPGRADE. In license files, feature definition lines are used with SERVER and VENDOR lines to define the license model. In trusted storage, feature definition lines are used with the fulfillment record properties to define the license model.
feature line	See feature definition line.
file-based activation	See manual activation.
FlexEnabled application	A software application that uses the FlexNet Publisher Licensing Toolkit to implement its license models.
FlexEnabled client	A client system where a FlexEnabled application is installed.
Flexible API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that allow FlexEnabled clients to access license rights.
FlexNet Agent	A utility that runs on a license server and allows communication between the license server and other FlexNet products.
FlexNet Manager	This is a license server management and license management decision support solution in the FlexNet product family that allows companies to centrally manage license servers and license usage across the enterprise.
FlexNet Operations	A product in the FlexNet family of products that manages entitlements and generates license rights.
floating license	A license that can be shared among users and is served to FlexEnabled applications from a license server. When a user finishes using the license, another user can begin using it. Also called concurrent license or served license rights.
fulfill	The action of getting license rights.
fulfillment record	In the FlexNet Publisher Licensing Toolkit, a data structure specific to trusted storage that defines license rights.
heartbeat	A type of message sent periodically between two systems (for example, a FlexEnabled application and a license server) to identify whether each is running and able to communicate with the other.
hop count	In the FlexNet Publisher Licensing Toolkit, the number of times that license rights in a single fulfillment record have been transferred from one license server to another.

Term	Meaning
hostid	In the FlexNet Publisher Licensing Toolkit, the value of a specific system attribute that uniquely identifies the host. For example, if IP address is a hostid type, then the hostid might be 129.87.33.101. Hostids are used in node-locked license.
hostid type	A type of attribute that is available to uniquely identify a system. For example, disk serial number, IP address, or MAC address.
hybrid license	In the FlexNet Publisher Licensing Toolkit, a type of license right in the license group on trusted storage that can function as either a concurrent license or as an activatable license.
internet activation	See programmatic activation.
keyboard activation	See short-code activation.
license administrator	A person in an enterprise who is responsible for installing the FlexEnabled application and administering the licenses and license servers.
license file	A text file, usually with the `.lic` extension, that contains license certificates from one or more publishers.
license-file list	See license search path.
license generator	A general term used to describe any mechanism a software publisher uses to generate license rights for their customers. The following FlexNet capabilities and products can function as a license generator: ● FlexNet Operations: generates license rights for trusted storage and license files. ● lmcrypt: generates license rights for license files.

Term	Meaning
license group	A mechanism used to describe licenses rights held in server-side trusted storage. These groups are not used with license rights held in license files. ● **activatable**—A license right in trusted storage that can be activated to a FlexEnabled client. ● **concurrent**—license rights in this group can be shared among users by having a FlexEnabled client to check out, and then return the license. ● **hybrid**—these license rights have the same characteristics as both the activatable license group and concurrent license group. They can be activated to client-side trusted storage on a FlexEnabled client or checked out by a FlexEnabled client. Although you see these group names used to describe license rights in client-side trusted storage, the capabilities of the license rights do not change. Once license rights are in client-side trusted storage, you cannot transfer, serve, or activate them to another system. The FlexEnabled application can only check out and check in these licenses. The group names are used when a license right is returned to the license server, so that it is assigned to the correct license group on the license server.
license model	The set of characteristics that determine the conditions under which (for example, how, when, where, and by whom) a FlexEnabled application can be used.
license rights	Information in trusted storage or license files that defines the license model and controls user access to a specific FlexEnabled application.
license search path	A property that the FlexEnabled application uses to locate where license rights are stored. Sometimes known as a license-file list.
license server	A set of files from the FlexNet Publisher Licensing Toolkit, that work together to manage and serve licenses to FlexEnabled applications. A license server is required to support a concurrent license model. The license server consists of a license server manager, one or more vendor daemons, zero or more options files, zero or more debug logs, and, optionally, a report log. These components are available in the FlexNet Publisher Licensing Toolkit. If the license server supports license rights held in trusted storage, it also includes an activation utility. It may, optionally, act as an activation server.
license server system	See license server.
license server manager (lmadmin or lmgrd)	A license server manager `lmadmin` or `lmgrd` executable that forwards connections from a FlexEnabled application to the correct vendor daemon. Because the vendor daemons are responsible for authenticating requests and serving licenses, a single license server manager can be used with multiple vendor daemons on the license server.

Term	Meaning
license sharing	A capability available in the FlexNet Publisher Licensing Toolkit where multiple license requests from the same user, host, or display results in only one license being used. Also known as duplicate grouping.
licensing client	See FlexEnabled client.
lmadmin	The license server manager introduced in version 11.6 which includes a Web-based GUI.
lmgrd	A command-line-based license server manager.
local license rights	License rights that are stored on the same system where the FlexEnabled application is running, rather than on a license server.
local activation	A type of activation where license rights are loaded using an ASR file. The ASR file must reside with the FlexEnabled application. This does not require user intervention.
machine ID	A value, generated using attributes of the system, that is used to bind trusted storage to that system.
maintenance	A maintenance represents a customer's right to obtain updates and upgrades to a particular licensed product or suite for a specified duration. Used in FlexNet Operations.
manual activation	An type of activation where license rights are transmitted from an activation server to another system using a set of request and response messages that are formatted as XML. These messages are saved to and read from physical XML files. An activation utility loads the license rights from an XML file into trusted storage.
Manual Response Generator (responsegen.exe)	A tool included in the FlexNet Publisher Licensing Toolkit that enables software publishers to test the activation process offline using XML request and response messages stored as files.
node-locked license	A license that can be used only when the FlexEnabled application is run on a specific system or by a specific user (as defined by the hostid).
options file	A configuration file available on the license server that license administrators can use to either change license server behavior or implement certain licensing policies. Each software publisher has its own separate options file that works with a specific vendor daemon.
orderable	Anything that can be purchased by a customer. Orderables can include suites, products, documentation, and maintenances.

Term	Meaning
package	In the FlexNet Publisher Licensing Toolkit, a set of features that can be grouped and licensed together.
part number	A unique identifier, typically correlated with an order system, assigned by a publisher to a product.
permanent license	License rights granted to a customer that do not expire.
product	Anything that can be sold by a publisher and can be licensed.
product ID	In the FlexNet Publisher Licensing Toolkit, one of the values that uniquely identifies the license rights in trusted storage. See also entitlement ID.
programmatic activation	A type of activation where license rights are transmitted from one system to another across a network using a set of request and response messages that are formatted as XML. This process is the identical to manual activation, except that it does not generate a physical XML file. An activation utility loads the license rights from the XML message into trusted storage. This is an automated process and does not require user intervention.
publisher ID	A value used by trusted storage to uniquely identify the software publisher's logical and physical licensing components. Revenera distributes this identifier to software publishers. It is used to prepare certain FlexNet Publisher Licensing Toolkit files that access trusted storage. This value is not the same as the vendor daemon.
quorum	A condition specific to three-server redundancy that is met when at least two of the three license servers are running and can communicate with each other using heartbeats.
rehost	The process of reissuing a license for a different system. This process requires software publishers to perform certain activities to redefine the license rights so that they will work properly on the new system.
repair request	A message sent by an activation utility to an activation server that initiates a repair transaction. The activation server provides the information that repairs the license rights in trusted storage. An activation utility is an application that initiates the repair transaction.
repair response	A message sent from an activation server to an activation client, in response to a repair request. This message contains information that is used to the repair the license rights.
repair transaction	A sequence of repair request and repair response messages, and the related actions that result in license rights being repaired in trusted storage.
report log file	A file that runs on a license server that contains data about the features used by a single vendor daemon. Report logs are encrypted and cannot be read by a person, but are used by FlexNet Manager to produce reports.

Term	Meaning
responsegen	See Manual Response Generator (responsegen.exe).
return request	A message sent by an activation utility to an activation server that initiates a return transaction. The activation server receives the license rights returned by an activation client. An activation utility is an application that initiates the return transaction.
return response	A message sent from an activation server to an activation client, in response to a return request.
return transaction	A sequence of return request and return response messages, and the related actions, that result in license rights being returned from trusted storage to the activation server.
served license rights	License rights that are stored on a license server and are checked out when needed by FlexEnabled applications that run on client systems. These license rights are used for concurrent licensing.
short code	A multi-character code used with a short-code activation transaction, a repair transaction, or a return transaction.
short-code activation	A type of activation where license rights are loaded from an ASR file using an activation transaction that requires a specific code (called a short code). Often the user enters this code into a prompt from the application. This is also is used for licensing via automated telephony.
signature	A secure multi-character string added to the license certificate that ensures it has not been modified.
simple entitlement	An entitlement for a single, specific customer.
single-action transaction	This is the default transaction type for FlexNet Publisher versions 11.7 and earlier. Each transaction can include only a single action. Also known as legacy or V1 transactions. From FlexNet Publisher version 11.14, single actions are deprecated.
suite	A set products grouped together that are functionally complementary and associated with one or more license models.
supersede	A capability in the FlexNet Publisher Licensing Toolkit, used to replace the characteristics of old license rights with new characteristics.
three-server redundancy	A capability in the FlexNet Publisher Licensing Toolkit that enables license administrators to configure three license servers in a failover cluster.
triad	A set of three license servers which operate together to support three-server redundancy.

Term	Meaning
trial license	License rights granted to a user for a limited duration so that the user can evaluate the software application.
trusted storage	A tamper-proof area on the system that stores license rights. This storage location is different than license files and requires a different mechanism for loading and managing license rights. Compare with license file.
trusted configuration	A set of attributes that define the security configuration of a segment of trusted storage. Each software publisher can configure their segments of trusted storage differently. This configuration defines the settings for binding, anchoring, and windback detection.
trusted configuration file	A file that stores the attributes that define a software publisher's trusted storage segment configuration.
trusted storage configuration	See trusted configuration.
uncounted	A license right attribute that allows an unlimited number of concurrent uses of a FlexEnabled application. Compare with counted.
unserved license rights	These are license rights that are not served from a license server. These license rights are located with the FlexEnabled application. Also known as local license rights.
upgrade	A process where a user installs a newer version of a FlexEnabled application and also updates the license rights so that they can use this newer version.
upsell	The process of transitioning from a product of lesser functionality to one of greater functionality in the same product line.
vendor daemon	One of the files that is a part of the license server. This executable is customized and built by the software publisher using files in the FlexNet Publisher Licensing Toolkit. The vendor daemon is responsible for communicating with the FlexEnabled application and issuing licenses.
Vendor Certificate Generator	A publisher-built component, that is a part of FlexNet Operations, that creates license certificates or signed feature lines in fulfillment records. This functions similarly to lmcrypt.
vendor name	A value used to uniquely identify a software publisher's logical and physical licensing components. It is used to prepare certain files in the FlexNet Publisher Licensing Toolkit that manage license rights in both trusted storage and license files. Revenera distributes this identifier to software publishers. This value is not the same as the publisher ID.
windback	A condition where the system clock has been set to an earlier date or time. One reason users change the system clock is to extend the license duration.

Term	Meaning
windback detection	A capability in the FlexNet Publisher Licensing Toolkit that detects whether a user has reset the system clock. This prevents a user from extending the duration of the license.

Index

A

activatable license 53
activated feature line, See feature definition line
activation 53
Activation API 53
activation application 54
activation client 53
activation request 53
activation response 54
activation server 54
activation specification record 53
activation transaction 54
activation, short code 62
anchor 54
automatic heartbeats 40

B

basic implementation 30
binding 54
borrowing 54
bulk entitlement 54

C

checkin 54
checkout 55
checkout-data strings 38
class
    demoInfo 22
    vendorInfo 29
CLASSPATH 16, 22, 28
client system 55
composite transaction 55
concurrent license 55
config response 55
considerations, license model 13
consumer 55
counted license 55

D

daemon, vendor 63
debug log file 55
demo license, See trial license
demoInfo class, creating on UNIX 16
demoInfo class, creating on Windows 22
deployed files
    lmgr11.dll 33
    lmgrd 33
    lmtools.exe 33
    lmutil 33
    vendor daemon 33
deploying your application 33
duplicate grouping 55

E

embedded activation, See local activation
encryption seed 56
end user 56
end-user license administrator, See license administrator
end-user machine, See client system
enterprise 56
entitlement 56, 59
entitlement ID 56
error codes 48
error message localization 45
evaluation

UNIX 15
Windows 21
exceptions 45
executing your application 32
expiring license 56

F

feature 56
feature bundle 57
feature definition line 53, 57
feature specifiers 36
FEATURE/INCREMENT line, See feature definition line
file
license 58
options 60
report log 61
file-based activation, See manual activation
FlexEnabled application 57
FlexEnabled client 57
Flexible API 57
flexlm.jar 16, 22
FlexNet Agent 57
FlexNet Operations 57
FlexNet Publisher
components 11
UNIX evaluation 15
Windows evaluation 21
floating license 57
fulfill 57
fulfillment record 57

H

heartbeats 39, 57
automatic 40
manual 41
semi-automatic 42
hop count 57
hostid 58
hostid type 58
hybrid license 58

I

implementation,basic 30
installation files 10
internet activation, See programmatic activation

J

Java API, adding to your application 30

K

keyboard activation, See short-code activation

L

license
counted 55
served 62
unserved 63
license administrator 58
license count 38
license file
creating (UNIX) 18
creating (Windows) 24
creating to test application 31
glossary definition 58
specifying location 29
license generator 58
license group 59
license management 11
license model 59
license model considerations 13
license rights 59
local 60
license search path 59
license server 59
starting 18
stopping 19
license server manager 59
license server,starting 24
license server,stopping 25
license sharing 60
license sources 36
license-file list, See license search path
licenses
served 12
unserved 12
licensing client 60
lmadmin 60
lmgr11.dll 33
lmgrd 33, 60
lmtools 33
lmutil 33
local activation 60
local license rights 60
localization, error messages 45
locating the license server
license file 29
port@host 29
log file
debug 55
report 61

M

machineid 60
manual activation 60
manual heartbeats 41
Manual Response Generator 60
model, license 59

N

newlink GlossaryC 54

O

options file 60

P

package 61
part number 61
permanent license 61
port@host 29
prerequisites 9
product 61
programmatic activation 61
publisher id 61

Q

quorum 61

R

regular entitlement, See simple entitlement
rehost 61
repair request 61
repair response 61
report log file 61
responsegen 62
return request 62
return response 62
return transaction 62

S

sample application, UNIX 17
sample application, windows 23
security 14
semi-automatic heartbeats 42
served license rights 62
served licenses 12
server
    license 59
short code 62
short-code activation 62
signature 62
simple entitlement 62
single-action transaction 62
starting a license server 18
stopping a license server 19
suite 62
supersede 62

T

tamper-resistant licenses 14
testing your application 31
three-server redundancy 62
toolkit 9
    customizing 28
triad 62
trial license 63
trusted configuration file 63
trusted storage 10, 63
trusted storage configuration 63
type, hostid 58

U

uncounted license
    uncounted 63
UNIX evaluation 15
unserved license rights 63
unserved licenses 12, 63
upgrade 63
upsell 63

V

Vendor Certificate Generator 63
vendor daemon 33, 63
vendor name 63
vendor-defined host ids 42
vendorInfo.class, creating 29
VendorInfo() 37

W

what's installed 10
windback 63
windback detection 64
Windows evaluation 21