FlexNet Publisher 2018 R2 (11.16.0)

Development Environment Guide

Legal Information

Book Name: FlexNet Publisher 2018 R2 Development Environment Guide

Part Number: FNP-11160-DEV00

Product Release Date: June 2018

Copyright Notice

Copyright © 2018 Flexera.

This publication contains proprietary and confidential information and creative works owned by Flexera 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 is strictly prohibited. Except where expressly provided by Flexera in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera, 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, see https://www.flexera.com/producer/company/about/intellectual-property/. All other brand and product names mentioned in Flexera 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

Introduction 9

Toolkit Functionality 10

FlexNet Publisher Product Documentation 11

Identifying the Toolkit Directories 11

Path Designations for Windows and UNIX in This Book 12

Part 1: Building the Production Licensing Tool Kit 13

1 First Build of the Production Licensing Toolkit 14

Major Toolkit Components 14

Configuring the FlexNet Publisher Licensing Toolkit 15

Editing lm_code.h 16

Editing the makefile 17

Editing publisher.xml (Trusted Storage Only) 18

Editing the Vendor Daemon Preptool Configuration File (Trusted Storage Only) 19

Defining Trusted Configuration (Trusted Storage Only) 19

Building the Production Licensing Toolkit 23

Building on UNIX 23

Building on Windows 25

Building on OS X 30

Testing the Production FlexNet Publisher Licensing Toolkit 32

Licensing Toolkit Components 32

Configuration Files 32

Generated Components 33

Standard Components 36

Source Code 43

2 Testing the Production Licensing Toolkit With a License File 44

Testing a FlexEnabled Application Using a License File 44
Testing the Vendor Daemon Using License Files 45
Creating a Publisher-Specific License File 45
Starting the License Server 45
Checking Out Licenses 46

3 Testing the Production Licensing Toolkit with Trusted Storage 47
About the Test 47
What To Know Before Starting the Test 47
Components and Files Tested 48
Preparing for the Test 48
Installing the FlexNet Licensing Service on Linux 49
Installing the FlexNet Licensing Service on OS X 50
Installing the FlexNet Licensing Service (Windows) 50
Uninstalling the FlexNet Licensing Service (Windows) 51
Resetting Trusted Storage 52
Setting Up the Activation Response Parameter File 53
Activating License Rights 56
Borrowing License Rights 58
Setting the Borrow Reclaim Count 58
Resetting the Borrow Reclaim Count 58
Testing Vendor Daemon Use of Trusted Storage 59
Creating a Publisher-Specific License File 59
Starting the License Server 60
Checking Out a License Held in Trusted Storage 61

4 Rebuilding the Production Licensing Toolkit 62
Upgrading to a New Toolkit Version 62
Editing lm_code.h 63
Editing the makefile 64
Copying publisher.xml 65
Editing Vendor Daemon Preptool Configuration File (Trusted Storage Only) 65
Rebuilding to Create a Trusted Storage Production Toolkit 65
Editing makefile.act 66
Editing publisher.xml 66
Editing Vendor Daemon Preptool Configuration File 67
Using a TRL-Only Version of the Vendor Daemon 67
Using the C++ Dynamic Run-time Library 67

Part 2: For Trusted Storage–Based Licensing 68

5 Building Your FlexEnabled Application for Trusted Storage–Based Licensing 69
Identifying Security Functionality in Your Toolkit 69

Security Considerations **70**
Generating a New Version of lm_new for Each Build 70
Tamper-Resistant Licensing: TRL-Only Libraries 70
sdtconfig.h 71
Building UNIX Applications **71**
Build Considerations for Shared Objects 71
Building the Application Without Secure Data Types 72
Building the Application With Secure Data Types 73
Building Mac Applications With FlexNet ID Dongle Support 75
Building Windows Applications **75**
Building the Application Without Secure Data Types 76
Building the Application With Secure Data Types 77
Prepping the Application **79**
Signing the Application **80**
Testing the Application **80**
Creating the ASR 81
Activating License Rights 82
Checking Out a License 84

6 Distributing the FlexEnabled Application for Trusted Storage–Based Licensing **86**
Identifying Security Functionality in Your Toolkit **86**
FlexNet Publisher Components Your End User Requires **87**
FlexNet Licensing Service 87
Application Components 87
License Rights Components 88
Installer Considerations **91**
Installing the FlexNet Licensing Service 91
Trusted Storage Directory Initialization (UNIX Platforms) 92
Installing the FlexEnabled Application 92
Installing the Activation Utilities 92
Licenses from Trusted Storage 93
Installing the License Server 93
What to Tell Your End Users **93**
Licensing Agreements 93
Up-to-date FlexNet Publisher Components 93
License Server Need 94
License Lifecycle Information 94

Part 3: For License File–Based Licensing **95**

7 Building Your FlexEnabled Application for License File–Based Licensing **96**
Security Considerations **96**
Generating a New Version of lm_new for Each Build 96
Tamper-Resistant Licensing: Using TRL-Only Libraries 97

Using Non-TRL Libraries . . . 97
Building on UNIX . . . 97
Build Considerations for Shared Objects . . . 98
Building the Application Without Secure Data Types . . . 98
Building the Application With Secure Data Types . . . 99
Building Mac Applications With FlexNet ID Dongle Support . . . 101
Building Windows FlexEnabled Applications . . . 102
Building the Application Without Secure Data Types . . . 102
Building the Application With Secure Data Types . . . 107
Signing the Application . . . 108
Testing the Application . . . 109
Using lmstrip for Additional Security . . . 110
lmstrip Variants . . . 110
lmstrip for Windows: Functionality and Usage . . . 110
lmstrip for UNIX (ELF Platforms): Functionality and Usage . . . 113
Linking with a Library that Already Uses FlexNet Licensing . . . 115

8 Distributing the FlexEnabled Application for License File–Based Licensing . . . 116
FlexNet Publisher Components Your End User Requires . . . 116
Generating the License Certificate . . . 117
Delivering the License Certificate . . . 119
Single License Server vs. Redundant License Servers . . . 119
Enabling Support for Windows Event-Logging . . . 119
Shipping the Message File DLLs . . . 120
Configuring the End User's Registry . . . 120
Localizing the Event-Logging Message Files . . . 121
Installing the License File . . . 121
Vendor Considerations . . . 121
End User Considerations . . . 122
Installing the License Server . . . 122
What to Tell Your End Users . . . 123
Up-to-date FlexNet Publisher Components . . . 123
License Server Need . . . 123
Hostid Types . . . 124
Default License File Location . . . 124
License File Environment Variable Enabled . . . 124
License File Location in Registry Enabled . . . 125
Reading the Registry . . . 125
License Finder (Windows Only) . . . 125
License Denial/Checkout Failure . . . 125
License Timeout . . . 126
License Lingering . . . 126
Reconnection Strategy . . . 126
License Borrowing Interface . . . 127

Early Return of Borrowed Licenses . . . 127
Duplicate Grouping . . . 128
Host Naming Convention . . . 128
License Server Organization . . . 128

Part 4: Other Development Tasks . . . 130

9 Implementing Secure Data Types (SDTs) . . . 132

Implementation Overview . . . 132
Operation of SDTs . . . 133
SDT Behavior During a Valid Checkout . . . 133
SDT Behavior in the Absence of a Valid Checkout . . . 134
Independence of Features . . . 135
Mapping SDTs to Features . . . 136
Choosing Constants and Variables as SDTs . . . 136
Building and Testing Code That Includes SDTs . . . 136
Guidelines for Using SDTs . . . 138
Files Provided in the Toolkit for Implementing SDTs . . . 138
sdtconfig.h . . . 139
Additional Function Calls Needed . . . 139
Product with Multiple Executables . . . 140
Running preptool on Applications with SDTs . . . 141
Preparing Applications That Use Trusted-Storage Licensing . . . 142
Preparing Applications That Use License File-Based Licensing . . . 142
SDT Sample Files Included in Toolkits . . . 143
For Trusted Storage–Based Licensing . . . 144
For License File-Based Licensing . . . 145
SDTs for C++ . . . 146
C++ SDT Macro Reference . . . 149
SDT_FEATURE_DEFINE . . . 149
SDT_FEATURE_DEFINE_CONST . . . 149
SDT_FEATURE_GET_NAME . . . 150
SDT_FEATURE_IMPLEMENT . . . 150
SDT_FEATURE_IMPLEMENT_BEGIN . . . 151
SDT_FEATURE_IMPLEMENT_END . . . 151
SDTs for C . . . 151
C SDT Macro Reference . . . 153
SDT_FEATURE_DECLARE_VAR . . . 153
SDT_FEATURE_DEFINE . . . 154
SDT_FEATURE_GET . . . 154
SDT_FEATURE_IMPLEMENT_BEGIN . . . 155
SDT_FEATURE_IMPLEMENT_C . . . 155
SDT_FEATURE_IMPLEMENT_END . . . 156

SDT_FEATURE_*operator_name* *156*
SDT_FEATURE_SET 156
SDT_FEATURE_SET_CONST 157

10 Providing the License Server Installation Instructions 158

Installing the License Server lmadmin 158
Important Notes About lmadmin Installation 158
Downloaded lmadmin Installer 159
lmadmin Installer Wrapped in Your Own Installation Program 159
Your Own lmadmin Installer 163
Properties File for the lmadmin Installer 163
lmadmin Command-Line Installation Variables 163
Command-Line lmadmin Options for Installers 167
Installing lmadmin as a Service (Windows Platforms) 169

Installing the License Server lmgrd 170
Installing lmgrd and Vendor Daemons 170
Starting lmgrd at System Startup Time (UNIX Platforms) 171
Installing lmgrd as a Service (Windows Platforms) 171
lmgrd Installation Examples 173
Starting lmgrd Manually 173

A Setting Up Microsoft Visual Studio for a FlexEnabled Application 175

Including the FlexNet Publisher Header Files 176

Adding a preptool Build Step for Trusted Storage–Based Applications 176

Building an MFC Application 177

Setting Up Other Application Types 178
Activation Utilities 178
Manual License Generator Applications 178

Glossary 179

Index 190

Introduction

This manual, *Development Environment Guide*, describes how to use the FlexNet Publisher to do the following:

1. Build your production FlexNet Publisher
2. Build your FlexEnabled application
3. Package your FlexEnabled application for distribution

It contains the following chapters:

Table 1 • Navigation Table

Chapter	Content
Introduction	Introduction to this document and the documentation for FlexNet Publisher.
Part 1, "Building the Production Licensing Toolkit"	
First Build of the Production Licensing Toolkit	Instructions for the first build of the *production* FlexNet Publisher toolkit. Publisher-specific settings for encryption and the publisher's vendor daemon name are defined during this first build.
Testing the Production Licensing Toolkit With a License File	How to test that the components created by the toolkit operate correctly with license files.
Testing the Production Licensing Toolkit with Trusted Storage	Optional, how to test that components created by a toolkit that supports trusted storage–based licensing operate correctly with licenses rights held in trusted storage.
Rebuilding the Production Licensing Toolkit	Instructions for subsequent builds of the production Licensing toolkit. These builds use the publisher specific settings originally defined in the first build.
Part 2, "For Trusted Storage–Based Licensing"	

Table 1 • Navigation Table

Chapter	Content
Building Your FlexEnabled Application for Trusted Storage–Based Licensing	Instructions for building a licensed application that accesses local license rights held in trusted storage.
Distributing the FlexEnabled Application for Trusted Storage–Based Licensing	What licensing components must be packaged with a licensed product that uses trusted storage. Also includes installation instructions.
Part 3, "For License File-Based Licensing"	
Building Your FlexEnabled Application for License File–Based Licensing	Instructions for building a licensed application that uses license rights held in license files.
Distributing the FlexEnabled Application for License File–Based Licensing	What licensing components must be packaged with a licensed product that uses license files.
Part 4, "Other Development Tasks"	
Implementing Secure Data Types (SDTs)	Steps for implementing Secure Data Types in your code to prevent end-users from bypassing the licensed-feature checkout.
Providing the License Server Installation Instructions	Methods and procedures for providing the license-server installer to end users.
Appendixes	
Setting Up Microsoft Visual Studio for a FlexEnabled Application	Instructions on how to set up Visual Studio to compile, link and build FlexEnabled applications.

Toolkit Functionality

The licensing and security functionality available in the toolkit depends on the platform on which the toolkit and FlexEnabled applications are run. There are three levels of functionality:

- **Level One: Licensing using license files only**—Instructions in Parts 1 and 3 are relevant for these platforms.
- **Level Two: Licensing using license files and trusted storage but without additional security**—Instructions in Parts 1, 2, and 3 are relevant. In Part 2, use the instructions for *notr* toolkits.
- **Level Three: Licensing using license files and trusted storage and including additional security**—Instructions in Parts 1, 2, and 3 are relevant. In Part 2, use the instructions for *standard* toolkits.

For details of the functionality available in your toolkit, see the Release Notes issued with your toolkit.

FlexNet Publisher Product Documentation

Flexera provides a comprehensive documentation set to assist you in working with the FlexNet Publisher. See the *Reference Bulletin for Documentation* for the current release for information about the most up-to-date documentation.

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
`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.

`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:
	`<install_dir>\<platform_dir>`
	UNIX platforms:
	`<install_dir>/<platform_dir>`

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

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

Path Designations for Windows and UNIX in This Book

For simplification in this book, file paths for locations common to both UNIX and Windows platforms are constructed using forward slashes.

Part 1

Building the Production Licensing Tool Kit

This part of the Development Environment Guide includes the following chapters:

- First Build of the Production Licensing Toolkit
- Testing the Production Licensing Toolkit With a License File
- Testing the Production Licensing Toolkit with Trusted Storage
- Rebuilding the Production Licensing Toolkit

1

First Build of the Production Licensing Toolkit

FlexNet Publisher provides all of the components that you need to create a licensed—or *FlexEnabled*—application, along with additional licensing components. As shipped, the FlexNet Publisher Licensing Toolkit (also simply referred to as the *Toolkit*) provides a pre-built sample vendor daemon (called `demo`) and a set of sample FlexEnabled applications. These components are used with the tutorials available in the *Getting Started with License File–Based Licensing* and *Getting Started with Trusted Storage–Based Licensing* guides.

To create your own FlexEnabled application, you must first build a production licensing toolkit that is customized with your private, publisher-specific information. The output of this build is a collection of publisher-specific tools that you use to enable flexible licensing in your products.

This chapter provides the information you need to build your production toolkit:

- **Major Toolkit Components**—Provides an overview of important licensing toolkit files.
- **Configuring the FlexNet Publisher Licensing Toolkit**—Describes how to configure the licensing toolkit components in preparation for building your custom production licensing toolkit.
- **Building the Production Licensing Toolkit**—Explains how to build your production licensing toolkit.
- **Licensing Toolkit Components**—Defines each component of the Toolkit and its interaction with the FlexEnabled application and licensing components.

If you are upgrading to a new version of FlexNet Publisher or rebuilding your production licensing toolkit, see Chapter 4, Rebuilding the Production Licensing Toolkit.

Major Toolkit Components

This section introduces the toolkit files, both the configuration files used to build your production licensing toolkit and the publisher-specific licensing components generated as a result of the build.

Configuration Files Used to Build the Toolkit

To create the publisher-specific components, you use the following toolkit files:

- **lm_code.h**—Contains encryption settings and the vendor daemon name.

- **makefile**—Builds the production licensing toolkit. Two makefiles are available:
  - **makefile**—Builds components that use license files only
  - **makefile.act**—Builds components that use trusted storage or license files

Note • *It is recommended to use GNU Make tool on all non Windows platforms.*

- **publisher.xml**—Contains encryption settings for your publisher-specific trusted storage.
- **DemoTrustedConfigIn.xml** —Serves as a template for defining a prepped Trusted Configuration used to create the initial section in trusted storage on a physical or virtual machine.

Generated Publisher-Specific Licensing Components

The following publisher-specific components are created when you build the production licensing toolkit:

- **Vendor daemon**—The publisher-specific component in a FlexNet license server. Operating with `lmadmin` (or `lmgrd`), the vendor daemon serves licenses to FlexEnabled applications. You can build the daemon in one of two ways:
  - **As license-file vendor daemon**—Obtains license rights from license files only
  - **As a trusted-storage vendor daemon**—Obtains license rights from license files and trusted storage
- **lmcrypt**—A utility that adds a signature to license rights. License rights are held either in license files or in feature-definition lines within fulfillment records in trusted storage. `lmcrypt` is used to sign both of these types of license rights.
- **lm_new.o**—A file (`lm_new.obj` on Windows and `lm_new.o` on UNIX) linked to the FlexEnabled application and containing publisher-specific parameters.

Configuring the FlexNet Publisher Licensing Toolkit

The sections that follow describe how to edit the various configuration files needed to build your toolkit. In general, by editing these files, you do the following to prepare for the build:

- Define your vendor keys that enable the toolkit functionality that you have purchased from Flexera.
- Identify your vendor name used to build your specific vendor daemon.
- Define settings that enable your publisher-specific encryption (used by your licensed applications and licensing components).

All sections that follow apply when building a licensing toolkit that supports license files or trusted storage except when specifically stated.

Important • *Before building your production licensing toolkit, save a copy of the FlexNet Publisher Licensing Toolkit (or install a separate copy from the downloaded Toolkit* `.zip` *or* `.tar` *file) so that you can continue to run the tutorials using the sample* `demo` *files, not your publisher-specific files.*

Keeping Encryption Settings Secure

It is important that you keep your encryption settings secure. Ensure the following:

- Encryption settings are kept secret and not made generally available.
- Encryption settings and the vendor name are not changed. If they are changed, then licensing components—for example, the vendor daemon—will not work with licensed applications created using the previous encryption settings.

Specifically `lm_code.h and publisher.xml` (or `sdt_publisher.xml`) should be kept secret and not be made generally available to your development teams.

Editing lm_code.h

The file `lm_code.h`, located in the `machind` directory, contains the vendor name definition, your encryption keys, and information about the encryption strength used when signing licenses. Keep the `lm_code.h` file and the publisher-specific values it contains secret.

In summary, the toolkit-build process uses `lm_code.h` to build the following components:

- The license generator, `lmcrypt.exe` (`lmcrypt` on UNIX platforms)
- The vendor daemon
- `lmnewgen.exe` (`lmnewgen` on UNIX platforms)
- `lm_new.obj` (`lm_new.o` on UNIX platforms), on which all FlexEnabled applications depend

Task ***To edit lm_code.h***

1. Open `machind/lm_code.h` in a text editor.
2. Replace the five `VENDOR_KEY` lines with your production vendor keys that you received from Flexera. These keys define your publisher rights from Flexera.

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

3. Replace `demo` in the `VENDOR_NAME` line with your vendor name, as registered with Flexera:

   ```
   #define VENDOR_NAME "<your_vendor_name>"
   ```

4. Add seeds that are used to generate unique, publisher-specific encryption keys. Change each dummy `LM_SEED` to a 32-bit number that you make up. Use different numbers for each `LM_SEED`.

   Tip • *If you do not want to make up `LM_SEED` seed numbers on your own, you can use FlexNet Publisher's special utility to generate random seed numbers for you. See Generating Encryption Seed Numbers Automatically.*

5. Replace the two `TRL_KEY` values with those that you received from Flexera:

   ```
   #define TRL_KEY1 0x0
   ```

```
#define TRL_KEY2 0x0
```

6. If required, alter the default setting for TRL signature length. To alter the default signature length (`LM_STRENGTH_113BIT`), change the `<strength>` value in the following line:

   ```
   #define LM_STRENGTH <strength>
   ```

   Define *`<strength>`* as one of these signature values:

Type	Signature Value	Bits	Number of hexadecimal characters
TRL	`LM_STRENGTH_113BIT`	113	60
	`LM_STRENGTH_163BIT`	163	84
	`LM_STRENGTH_239BIT`	239	120
Non-TRL (deprecated)	`LM_STRENGTH_DEFAULT` (Non-public key used)	N/A	12

   For further information about `LM_STRENGTH`, Tamper-Resistant Licenses (TRL), and signature values, see the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”.

7. Save and close `lm_code.h`.

Generating Encryption Seed Numbers Automatically

FlexNet Publisher provides a utility to help you generate random `LM_SEED` numbers if you do not want to create them yourself. The numbers are unique for each seed.

1. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmrand1 -seed
   ```

   A message appears explaining that, once generated, the seeds display in the command window and are stored in `<platform_dir>/lmseed.dat`.

2. Press **Enter** again to generate the seeds. Output similar to the following is displayed:

   ```
   #define LM_SEED1 0x8676c7c8
   #define LM_SEED2 0xa264ce64
   #define LM_SEED3 0x4dd27d8c
   ```

3. Replace each `LM_SEED` value in `lm_code.h` with your values.

Important • *Keep the encryption seeds secret.*

Editing the makefile

The `<platform_dir>` directory contains two makefiles:

- **makefile**—For use with license file–based licensing only
- **makefile.act**—For use with both trusted storage–based licensing and license file–based licensing

The following instructions apply to both makefiles.

Task ***To edit the makefile:***

1. From the `<platform_dir>` directory, open `makefile` or `makefile.act` in a text editor.
2. Change the vendor daemon name from `demo` to your own vendor daemon name as registered with Flexera and used as `VENDOR_NAME` in `lm_code.h`. The vendor daemon name is identified either as `VENDORNAME=` or `DAEMON=`.

Editing publisher.xml (Trusted Storage Only)

If you are implementing trusted storage–based licensing, you need to edit the activation settings file `publisher.xml` to define the following:

- If you are building the toolkit for trusted storage–based licensing, the encryption seeds are used to generate security keys. These keys, in turn, are used to do the following for your FlexEnabled applications:
  - Generate required trusted-storage signatures
  - Enable the use of Secure Data Types (SDTs)
- If you are building the toolkit for license file–based licensing, the encryption seeds are used to generate security keys that enable the use of SDTs in your FlexEnabled applications.

If you are implementing license file–based licensing only, you need to edit the activation settings file `sdt_publisher.xml` to include the encryption seeds used to generate security keys that enable SDTs.

To provide a new set of unique security keys, edit the encryption seeds in the file with your own set of new values.

Tip • *The recommended method for keeping the encryption seeds secret is to embed the activation settings file in a publisher* `preptool`*. For instructions on creating a publisher* `preptool`*, see the chapter "Working with the Software Publisher Utilities" in the Programming Reference for Trusted Storage–Based Licensing.*

Task ***To add your encryption seeds to publisher.xml***

1. Using a text editor, open the file `publisher.xml` (or `sdt_publisher.xml`), which Flexera has provided in your toolkit.
2. Locate the eight `<Seed>` elements similar to the following:

   ```
   <Seed>01234567</Seed>
   ```

3. Change each `<Seed>` to an eight-character, hexadecimal number that you make up. Each of the eight seeds must be unique.
4. To overwrite the current activation settings file, save this file to the following location:

   `<platform_dir>/ActivationSettings/publisher.xml` (or `sdt_publisher.xml`)

Editing the Vendor Daemon Preptool Configuration File (Trusted Storage Only)

The `preptool` *configuration file,* `vendor_daemon.xml`*,* is used by `preptool` to create a vendor daemon that can access license rights in trusted storage. This file is used when running the `makefile` to create your vendor daemon and an associated activation library. The activation library enables the vendor daemon to access your publisher-specific trusted storage.

Task ***To edit the vendor daemon preptool configuration file:***

1. Using a text editor, open the platform-specific configuration tool.

Platform	Preptool Configuration File
Windows	`machind/activation/prep_xml/vendor_daemon.xml`
Mac	`machind/activation/prep_xml/vendor_daemon_mac.xml`
UNIX	`machind/activation/prep_xml/vendor_daemon_unix.xml`

2. Replace the two instances of the string `demo` with your vendor name, and update the path as needed.
3. Save and close the file.

Defining Trusted Configuration (Trusted Storage Only)

An activation utility manages license rights in trusted storage either on the license server or on the FlexEnabled client. When an activation utility is running and detects that no trusted storage exists, it automatically creates the trusted-storage file. However, to create the initial section within trusted storage that actually stores fulfillment records, the utility requires a Trusted Configuration. This Trusted Configuration can exist in an activation response or an activation specification record (ASR) so that when the utility processes either, the trusted section is created. However, best practice, is to embed a *prepped* Trusted Configuration within the activation utility itself so that it can create an initial trusted section immediately, without waiting to process an activation response or an ASR. (For local activations on the license server, using prepped Trusted Configuration is the *required* method for creating the initial trusted section.) For more information on the security benefits of using a prepped Trusted Configuration, see Chapter 5, “Configuring Security for Trusted Storage” in FlexNet Publisher’s *Programming Reference for Trusted Storage–Based Licensing*.

Currently, only certain toolkit platforms support prepped Trusted Configurations. (See the current *FlexNet Publisher Release Notes* for a list of these toolkits.) For toolkits that do not support prepped Trusted Configurations (that is, for toolkits that implement *non-prepped* Trusted Configurations only), you must create all trusted sections by an activation response or ASR only.

The next sections describe how to set up your own Trusted Configurations so that you can test and use your toolkit once it is built. Follow the specific setup summary for the type of Trusted Configuration you are implementing--prepped or non-prepped.

Setup Summary for Prepped Trusted Configuration

Setting up a prepped Trusted Configuration involves two phases:

Phase 1: Defining the Trusted Configuration

Phase 2: Updating the preptool Configuration Files Used to Build Activation Utilities

Then, when you build the activation utilities, the trusted configuration is embedded in the utility code and ready to be used to create trusted sections as needed. You can test the implementation of the prepped Trusted Configuration when you test your newly built toolkit in Chapter 3, Testing the Production Licensing Toolkit with Trusted Storage.

Setup Summary for Non-Prepped Trusted Configuration

This process manually sets up non-prepped Trusted Configurations. Because these configurations are not embedded in your activation utilities, you can define and implement them independently of the `preptool` workflow.

Note • *This setup process is not required when using FlexNet Operations or FlexNet Operations On-Demand. These products establish non-prepped trusted configurations using their own publisher server components (for example, FlexNet Operations's Transaction Key step within the Package Products workflow).*

Setting up a non-prepped Trusted Configuration involves two phases:

Phase 1: Defining the Trusted Configuration

Phase 2: Incorporating the Trusted Configuration into a response parameter file, as described in Chapter 3, Testing the Production Licensing Toolkit with Trusted Storage

Test the implementation of your non-prepped Trusted Configuration when you test your newly built toolkit in Chapter 3, Testing the Production Licensing Toolkit with Trusted Storage.

Defining the Trusted Configuration

A Trusted Configuration, contained in an XML file, is a publisher-defined configuration for binding and anchoring trusted storage to a specific end-user machine (license server or FlexEnabled client). The following procedure describes how to use sample files to set up Trusted Configuration input files and how to encrypt and sign these files. The procedure results in the following output:

- The signed and encrypted Trusted Configuration file, `<vendor>TrustedConfigOut.xml`, which defines your specific seeds, anchors, and binding for trusted storage on a physical end-user machine.
- The associated configuration key file, `<vendor>KeyFile.xml`, required by the Manual Response Generator.

Note • *You can set up your own Trusted Configuration input files from scratch using the* `O_ConfigPhaseOne` *file as a template (see Appendix A, "XML Schemas," in FlexNet Publisher's Programming Reference for Trusted Storage–Based Licensing). Additionally, you can create and build your own activation utilities (see later chapters in this guide). Best practice is to test your resulting Trusted Configuration files and activation utilities using the procedures described in Chapter 3, Testing the Production Licensing Toolkit with Trusted Storage.*

Task ***To create your publisher-specific versions of the Trusted Configuration files***

1. In the directory `<platform_dir>/publisher`, do the following, where `<vendor>` is your vendor name as registered with Flexera:

   - To configure trusted storage in a physical environment, copy `DemoTrustedConfigIn.xml` to `<vendor>TrustedConfigIn.xml`.

   The next steps describe how to modify and encrypt `<vendor>TrustedConfigIn.xml` first.

2. Open `<vendor>TrustedConfigIn.xml` in a text editor.

3. Locate the `<PublisherId>` element, and change its value to your publisher ID. (This is the publisher ID assigned to you by Flexera and supplied in the file `publisher.xml`.)

4. Locate the eight `<SeedPart>` elements similar to the following: <SeedPart>01234567</SeedPart>

5. Change each `<SeedPart>` to an eight-character, hexadecimal number that you determine. Each of the eight seeds must be unique. These seeds are used to generate transactions keys that, in turn, are used to sign activation responses.

   Note • *These seeds are different than those defined in the* `<SeedPart>` *element in* `<platform_dir>/ActivationSettings/publisher.xml`*. Seeds in* `publisher.xml` *generate keys used to sign Trusted Configurations and ASRs.*

6. To update or define other elements (such as anchoring, binding, or timing) in the Trusted Configuration, refer to the section "O_ConfigPhaseOne.xsd" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing* for element descriptions.

7. Save and close `<vendor>TrustedConfigIn.xml`.

8. Encrypt the file. To do so, at a command prompt from the `<platform_dir>/publisher` directory, enter the following:

   ```
   tcencrypt -i <vendor>TrustedConfigIn.xml \
       -o <vendor>TrustedConfigOut.xml \
       -k <vendor>KeyFile.xml
   ```

   The following output is displayed:

   ```
   Processing complete, config output written to file <vendor>TrustedConfigOut.xml
   Key output written to file <vendor>KeyFile.xml.
   ```

9. Sign `<vendor>TrustedConfigOut.xml`, created in the previous step. To do so, at a command prompt from the `<platform_dir>/publisher` directory, enter the following:

   ```
   xmlsign -a ../ActivationSettings/publisher.xml \
       -i <vendor>TrustedConfigOut.xml \
       -o <vendor>TrustedConfigOut.xml
   ```

   The following output is displayed:

   ```
   <vendor>TrustedConfigOut.xml signed.
   ```

10. Copy `<platform_dir>/publisher/<vendor>KeyFile.xml` to `<platform_dir>/publisher/xml`, and then delete `<platform_dir>/publisher/xml/DemoKeyout.xml`. This process replaces the Demo configuration key file with your publisher-specific key file. (Your key file is used by the Manual Response Generator.)

Updating the preptool Configuration Files Used to Build Activation Utilities

The `preptool` configuration files are used to prepare the server-side and client-side activation utilities that come with your toolkit (or that you create). To enable an activation utility to use the prepped Trusted Configuration to create initial trusted sections, modify the `preptool` configuration files to point to the Trusted Configuration files (created in the previous section). Then, when the activation utilities are built and prepared, the Trusted Configurations are embedded in the utilities, enabling the utilities to create initial trusted sections when needed.

The following procedure describes how to update the `preptool` configuration files for the example activation utilities that come with the toolkit. When you build your toolkit with your publisher-specific information, these example utilities are also rebuilt and prepared with the updated `preptool` configuration files. You can use these rebuilt utilities to test your toolkit (as described in Chapter 3, Testing the Production Licensing Toolkit with Trusted Storage).

Note • *Use this procedure for only those toolkits that support the embedding of prepped Trusted Configurations in activation utilities to create initial trusted sections. For a list of these toolkits, see the current FlexNet Publisher Licensing Toolkit Release Notes.*

Task ***To update the preptool configuration files with pointers to Trusted Configuration files***

1. Locate the `preptool` configuration files for your client-side and server-side activation utilities. The following lists locations for the sample utilities that ship with the toolkit:

   - Under `<install_dir>/examples/activation/comptranutil`, locate the two files (specific to the toolkit operating system) used to build activation utilities.

     For the client-side activation utility:

     `appCompTranUtil.xml` (Windows), `appCompTranUtil_unix.xml`, or `appCompTranUtil_mac.xml`

     For the server-side activation utility:

     `serverCompTranUtil.xml` (Windows), `serverCompTranUtil_unix.xml`, or `serverCompTranUtil_mac.xml`

     Note • *Although activation can still be performed using single-action transactions, best practice is to use composite transactions, which offer a more secure activation process. The utilities mentioned above support composite transactions. However, if you want to prepare activation utilities that support single-action transactions, perform the following steps:*

     Under `<install_dir>/examples/activation`*, locate the two files (specific to the toolkit operating system) used to build activation utilities that support single-action transactions.*

     For the client-side activation utility:

     `appActUtil.xml (Windows), appActUtil_unix.xml, or appActUtil_mac.xml`

     For the server-side activation utility:

     `serverActUtil.xml (Windows), serverActUtil_unix.xml, or serverActUtil_mac.xml`

2. Open each file, locate the `trustedconfiguration path` parameters, and replace the value for each parameter with the appropriate name of the output Trusted Configuration file you created in the previous section, Defining the Trusted Configuration:

```
<trustedconfiguration path="publisher/<vendor>TrustedConfigOut.xml"
```

Note • *If these parameters do not exist in your* `preptool` *configuration file, manually add them to embed the prepped Trusted Configurations in the activation utilities.*

3. Save and close each file.

Building the Production Licensing Toolkit

Refer to the appropriate section for your development platform. See Table 1-3 for a list of the licensing toolkit components that are built during this process.

- Building on UNIX
- Building on Windows
- Building on OS X

Building on UNIX

Setting Up the C Development Environment

To build the production licensing toolkit, you must set up the proper C development environment.

Task ***To set up your C development environment***

Configure your system so that the correct compiler is invoked by default. Consult the *FlexNet Publisher Release Notes* for information about the compiler required for the platform on which you are building. If no specific compiler is listed, use the ANSI C compiler.

Note • *To build a FlexEnabled application for Linux Standard Base (LSB) certification, you must follow the guidelines of the Linux Foundation. For more information about LSB certification, see the Linux Foundation's website http://www.linuxfoundation.org. For details about LSB support in FlexNet Publisher, see your FlexNet Publisher Release Notes.*

Running ranlib

Before you build the toolkit, process the FlexNet Publisher libraries with `ranlib`. This step ensures that the internal and archive file dates match after files are copied. If this step is omitted, link errors result when the dates do not match.

Task ***To run ranlib***

At a command prompt from the `<platform_dir>` directory, enter these commands:

```
ranlib *.a
ranlib activation/lib/*.a
```

PIE Support

The FlexNet Publisher Licensing Toolkit for Linux supports building client-side executables and utilities as Position-Independent Executables (PIE). Both `makefile` and `makefile.act` contain the `PIE` option. With the default setting, `PIE=0`, executables are not built as position-independent executables. To build position-independent executables, set `PIE=1`; the resulting executables will be relocatable at load time.

Note • *Ensure that you use the appropriate PIC libraries when building position-independent executables.*

For supported platforms and limitations, consult the *FlexNet Publisher Release Notes*.

Associating the Correct preptool Archive (Trusted Storage Only)

Some operating systems supported by FlexNet Publisher have both 32-bit and 64-bit toolkits. If you have installed both toolkits on your system, ensure your system is configured so that the correct `preptool` archive is associated with the toolkit you are building.

Task ***To configure the correct preptool archive for your toolkit***

At a command prompt, enter the following command to set the `PREPARCHIVE` environment variable:

```
setenv PREPARCHIVE <install_dir>/<platform_dir>/preparchive_11.nn.nnn.yaa
```

where *`11.nn.nnn`*.yaa is the archive file name, and *`1.nn.nnn`* is the archive build number. Locate the exact file name from the toolkit `<install_dir>`.

Building the UNIX Licensing Toolkit

To build your production licensing toolkit, use the `make` command with the appropriate `makefile` as a parameter. Use the following examples to help you set up the command.

Note • *Dongle drivers are not compliant with Linux Standard Base (LSB) certification. Therefore, omit dongle support if you plan to submit your FlexEnabled application to the Linux Foundation for LSB certification. For more information about LSB, see the Linux Foundation's website http://www.linuxfoundation.org.*

Building Toolkits for Trusted Storage–Based Licensing

Use the following procedure to build a toolkit that implements trusted storage–based licensing.

Task ***To build the toolkit for trusted storage–based licensing***

1. At a command prompt from the `<platform_dir>` directory, enter the following command to delete any previously installed sample activation components:

   ```
   make -f makefile.act clean
   ```

2. Enter the appropriate command to build the toolkit for trusted storage–based licensing:

   `make -f makefile.act` to build the toolkit *without* dongle support

or

`make -f makefile.act DONGLE=1` to build the toolkit *with* dongle support

Note • *To update the libtsJavaAcc.so and libtsJavaAcc_libFNP.so files run this command:* `make -f makefile.act libtsJavaAcc.so`

Building Toolkits for License File–Based Licensing

Use the following procedure to build a toolkit that implements license file–based licensing.

Task ***To build the toolkit for license file–based licensing***

At a command prompt from the `<platform_dir>` directory, enter one of the following commands to build the toolkit for license file–based licensing:

- **`make -f makefile clean all`** to build the toolkit without dongle support.

or

- **`make -f makefile DONGLE=1 clean all`** to build the toolkit with dongle support .

Build Results

The command window displays a list of files being built. The `make` command builds `lm_new.o`, which contains security information from `lm_code.h` and is used to build your application, your vendor daemon, as well as the sample applications. If you have edited only `lm_code.h`, the `make` command rebuilds everything except your application.

Building on Windows

Setting Up the C Development Environment

To build the production licensing toolkit, you must set up the proper C development environment. To set the development environment

Do the following:

- Compile using Microsoft Visual Studio. Refer to the latest *FlexNet Publisher Release Notes* for supported compiler and version information.

  Additionally, if your licensed applications run on Windows, review Complying with Windows Logo Programs for Software before building your toolkit.

- Make sure that you have the command-line interface of the Microsoft Visual C++ development environment correctly configured. (The licensing toolkit is built using command-line tools rather than the Microsoft Visual C++ integrated development environment, called IDE.) Consult the Visual C++ documentation for details about building from the command line.

Building the Windows Licensing Toolkit

To build your production licensing toolkit, use the `nmake` command with the appropriate `makefile` as a parameter. Use the following examples to help you set up the command.

Note • *If you link to the dongle libraries on Windows, your FlexEnabled application cannot use the* `/SafeSEH` *flag. Dongle files on the Windows platforms are not built with the* `/SafeSEH` *flag of the Visual Studio compiler.*

Building Toolkits for Trusted Storage–Based Licensing

Use the following procedure to build a toolkit that implements trusted storage–based licensing.

Task ***To build the toolkit for trusted storage–based licensing***

1. At a command prompt from the `<platform_dir>` directory, enter the following command to delete any previously installed sample activation components:

   `nmake -f makefile.act clean`

2. Enter either command to build the toolkit for trusted storage–based licensing:

   - **`nmake -f makefile.act`** to build the toolkit without dongle support

     or

   - **`nmake -f makefile.act DONGLE=1`** to build the toolkit with dongle support

Building Toolkits for License File–Based Licensing

Use the following procedure to build a toolkit that implements license file–based licensing.

Task ***To build the toolkit for license file–based licensing***

At the command prompt from the `<platform_dir>` directory, enter one of the following commands to build the basic toolkit for license file–based licensing:

- **`nmake -f makefile clean all`** to build the toolkit without dongle support.

or

- **`nmake -f makefile DONGLE=1 clean all`** to build the toolkit with dongle support.

Build Results

The command window displays a list of files being built. The `nmake` command builds `lm_new.o`, which contains security information from `lm_code.h` and is used to build your application, your vendor daemon, as well as the sample applications. If you have edited only `lm_code.h`, the `nmake` command rebuilds everything except your application.

Note • *To update the tsJavaAcc.dll file, run this command:* `nmake -f makefile.act tsJavaAcc.dll`.

Building Toolkits with Visual Studio

The procedure to build a toolkit using Visual Studio 2015 or later is identical with the procedures for license file–based or trusted storage–based licensing. For more information, see Building Toolkits for Trusted Storage–Based Licensing on page 26 and Building Toolkits for License File–Based Licensing on page 26, respectively.

An additional library file `libredir_std.lib` is required and is generated (from `machind\lm_redir_std.c`) when building the toolkit with Visual Studio version 2015 or later.

In addition, two new files are added in the toolkit that act as a support in resolving conflicts: `machind\lm_redir_std.h`, `machind\lm_redir_std.c`

Complying with Windows Logo Programs for Software

This section provides information about how the FlexNet Publisher components have been built and packaged, so that you can identify the additional steps necessary to certify your FlexEnabled applications and license server. It does not comprehensively describe the requirements for the Windows Logo Programs for Software. For information about those program, see the Microsoft documentation.

Preparation through Signing, Manifests, and Version Resources

The Windows Logo Programs for Software have certain requirements for software applications to be certified under this program. Although FlexNet Publisher is not certified under these programs, some components have been prepared in such a way that they should not prevent FlexEnabled applications from being certified. To support this, toolkit components have been prepared as follows:

- Certain executables contain embedded manifests that define the execution level.
- Dynamic link libraries and executables contain a version resource.
- Dynamic link libraries and executables have been signed with an Authenticode certificate.

Not all dynamic link libraries and executables have been prepared. For example, binaries built by the packaged `makefile` or `makefile.act` (for example, the vendor daemon) files are not signed. The software publisher must prepare these files to comply with the Windows Logo Programs for Software. In addition, publisher utilities are not signed because they should not be packaged into a FlexEnabled application. The following table summarizes the files that are affected and how they are affected.

Table 1-1 • FlexNet Publisher Licensing Toolkit files

File Name and Installation Location	Version Resource	Embedded Manifest	Signed
i86_n3\appactutil_libFNP.dll	Yes	No	No
i86_n3\demo_libFNP.dll	Yes	No	No
i86_n3\ezcalc_libFNP.dll	Yes	No	No
i86_n3\ezcalcsdtplus_libFNP.dll	Yes	No	No
i86_n3\ezcalcsdt_libFNP.dll	Yes	No	No
i86_n3\FNP_Act_Installer.dll	Yes	Yes	Yes

Table 1-1 • FlexNet Publisher Licensing Toolkit files

File Name and Installation Location	Version Resource	Embedded Manifest	Signed
i86_n3\FNPLicensingService.exe (embedded inside FNP_Act_Installer.dll)	Yes	Yes	Yes
i86_n3\installanchorservice.exe	Yes	Yes	Yes
i86_n3\lmborrow.exe	Yes	Yes	Yes
i86_n3\lmdiag.exe	Yes	Yes	Yes
i86_n3\lmdown.exe	Yes	Yes	Yes
i86_n3\lmadmin\lmadmin.exe	Yes	Yes	Yes
i86_n3\lmgrd.exe	Yes	Yes	Yes
i86_n3\lmhostid.exe	Yes	Yes	Yes
i86_n3\lminstall.exe	Yes	Yes	Yes
i86_n3\lmpath.exe	Yes	Yes	Yes
i86_n3\lmrand1.exe	Yes	Yes	Yes
i86_n3\lmremove.exe	Yes	Yes	Yes
i86_n3\lmreread.exe	Yes	Yes	Yes
i86_n3\lmstat.exe	Yes	Yes	Yes
i86_n3\lmstrip.exe	Yes	Yes	Yes
i86_n3\lmswitch.exe	Yes	Yes	Yes
i86_n3\lmswitchr.exe	Yes	Yes	Yes
i86_n3\lmtools.exe	Yes	Yes	Yes
i86_n3\lmutil.exe	Yes	Yes	Yes
i86_n3\lmver.exe	Yes	Yes	Yes
i86_n3\preptool.exe	Yes	Yes	Yes
i86_n3\serveractutil_libFNP.dll	Yes	No	No
i86_n3\tsreset_app_libFNP.dll	Yes	No	No

Table 1-1 • FlexNet Publisher Licensing Toolkit files

File Name and Installation Location	Version Resource	Embedded Manifest	Signed
i86_n3\tsreset_svr_libFNP.dll	Yes	No	No
i86_n3\uninstallanchorservice.exe	Yes	Yes	Yes
i86_n3\examples\serviceinstall\installshield12\FLEXnetActivationService\FNP_Act_Installer.dll	Yes	Yes	Yes
i86_n3\publisher\FnpCommsSoap.dll	Yes	Yes	Yes
i86_n3\publisher\responsegen.dll	Yes	Yes	Yes
i86_n3\publisher\responsegen.exe	Yes	Yes	Yes
i86_n3\publisher\responsegenUI.exe	Yes	Yes	Yes
i86_n3\publisher\tcencrypt.exe	Yes	Yes	Yes
i86_n3\publisher\xmlsign.exe	Yes	Yes	Yes

Using the FlexNet Licensing Service Installer

The FlexNet Licensing Service is delivered embedded inside an installer DLL (`FNP_Act_Installer.dll`). To install or uninstall the service, software publishers must call functions exported from this DLL. This file is needed for only those applications that use trusted storage. The Windows Logo Program for Software has specific requirements for installation, make sure you deploy this file in a way that meets those requirements.

For an example about how do install, see the InstallShield 2008 sample project supplied with the kit. This is available in the *<platform-dir>*`\examples\serviceinstall` directory. This directory also contains a README file explaining how to use the example. It is recommended that you use the InstallShield 2008 project example because it has additional support for Microsoft Windows.

Building the Vendor Daemon

The vendor daemon is used with trusted storage–based and license file–based licensing when licenses are served from a license server. Each software publisher must build their own custom vendor daemon that contains unique vendor keys and encryption seeds.

To prepare your publisher-specific vendor daemon for certification, you must perform extra steps in the toolkit-build process to meet these specific requirements:

- Every executable file (with an `.exe` extension) included with an application must have an embedded manifest that defines its execution level.
- All executable files must be signed with an Authenticode certificate.

For full details, see the latest requirements for the Logo Program for Software, as issued by Microsoft Corporation.

The following outline instructions show the sequence of operations required.

Task ***To build a vendor daemon for Windows***

1. Create a resource file for your publisher-specific vendor daemon. It must be named `<vendor_name>.rc`, where `<vendor_name>` is your vendor name as used in `lm_code.h` (see Editing lm_code.h).

   The toolkit contains an example resource file, `demo.rc`, used to create a manifest for the demo vendor daemon. See Microsoft's documentation for further information about the required contents of this resource file.

2. Add the resource file `<vendor_name>.rc` to the `<platform_dir>` directory.

3. Build the toolkit as described in Building the Windows Licensing Toolkit. This builds your publisher-specific vendor daemon, `<vendor_name>.exe`.

4. Run the Microsoft manifest tool `mt.exe` to create a manifest for your publisher-specific vendor daemon.

   This manifest tool, included in the Windows platform SDK, is the recommended tool because it has been tested with the Toolkit.

5. Sign your publisher-specific vendor daemon as required by Microsoft.

Adding Support for Windows Event Logging

Windows toolkits contain the file `VendorLicenseServerMsgs.dll`, which you can use with the license server manager to write messages to the Windows Event Log, in addition to the debug log. (It can be used to log English messages only.) This file is not compliant with the Windows Logo Programs for Software because it has not been signed, nor does it contain a version resource.

In general, each publisher should generate their own messages DLL, rather than use this one. For more information about finding and using the source files to build a message DLL, see Enabling Support for Windows Event-Logging.

Building on OS X

You must build the Licensing toolkit on your specific OS X platform in order to generate the components necessary to build FlexEnabled applications that run on one or more OS X versions.

Requirements Building the Toolkit on OS X Platforms

Refer to the current *FlexNet Publisher Release Notes* for the latest toolkit-build requirements and the platforms supported for the FlexEnabled applications that a toolkit built on a specific OS X platform produces.

Running ranlib

Before you build the toolkit, process the FlexNet Publisher libraries with `ranlib`. This step ensures that the internal and archive file dates match after files are copied. If this step is omitted, link errors result when the dates do not match.

Task ***To run ranlib***

At a command prompt from the `<platform_dir>` directory, enter these commands:

```
ranlib *.a
ranlib activation/lib/*.a
```

Building the Mac Licensing Toolkit

To build your production licensing toolkit, use the `make` command with the appropriate `makefile` as a parameter. Use the following examples to help you set up the command.

Building Toolkits for Trusted Storage–Based Licensing

Use the following procedure to build a toolkit that implements trusted storage–based licensing.

Task ***To build the toolkit for trusted storage–based licensing***

1. At a command prompt from the `<platform_dir>` directory, enter the following command to delete any previously installed sample activation components:

   `make -f makefile.act clean`

   Enter the appropriate command to build the trusted-storage toolkit:

   `make -f makefile.act` to build the toolkit *without* dongle support

   or

   `make -f makefile.act DONGLE=1` to build the toolkit *with* dongle support

   or

   `make -f makefile.act DONGLE=1 USE_WIBU=1` to build the toolkit with Wibu dongle support

Building Toolkits for License File–Based Licensing

Use the following procedure to build a toolkit that implements license file–based licensing.

Task ***To build the toolkit for license file–based licensing***

At the command prompt from the `<platform_dir>` directory, enter the appropriate command to build the license-file toolkit:

- **`make -f makefile clean all`** to build the toolkit without dongle support.

or

- **`make -f makefile DONGLE=1 clean all`** to build the toolkit with dongle support.

or

- **`make -f makefile DONGLE=1 USE_WIBU=1`** to build the toolkit with Wibu dongle support.

Build Results

The command window displays a list of files being built. The command `make` builds `lm_new.o`, which contains security information from `lm_code.h` and is used to build your application, your vendor daemon, as well as the sample applications. If you have edited only `lm_code.h`, the command `make` rebuilds everything except your application.

Testing the Production FlexNet Publisher Licensing Toolkit

After building the Production Toolkit, test its operation before releasing it for use by your development teams.

- **License Files**—If you used `makefile` to build toolkit components that support license file–based licensing only, see Testing the Production Licensing Toolkit With a License File.
- **Trusted Storage**—If you have used `makefile.act` to build toolkit components that support both license file–based and trusted storage–based licensing, test using the instructions in both of the following sections:
  - Testing the Production Licensing Toolkit With a License File
  - Testing the Production Licensing Toolkit with Trusted Storage

Licensing Toolkit Components

The components that compose the licensing toolkit are divided into four groups. All files are located in the `<platform_dir>` directory of your licensing toolkit distribution unless otherwise noted.

- Configuration Files—Files you modify to include your publisher-specific information before the build.
- Generated Components—Modules that the build process generates using your modified configuration files.
- Standard Components—Components that are used as provided by Flexera. They do not contain publisher-specific information.
- Source Code—Source code for components that you might need to modify.

Configuration Files

The configuration files, listed in Table 1-2, are files you modify to include your publisher-specific parameters. See Configuring the FlexNet Publisher Licensing Toolkit for instructions on how to modify these files.

Table 1-2 • Configuration Files

Component	Description
lm_code.h	Defines vendor keys, vendor daemon name, TRL keys, and signature strength. See Editing lm_code.h for further details.
makefile	Contains rules that build the toolkit components for license file–based licensing only. This component also defines the vendor daemon name.
makefile.act	Contains rules that build the toolkit components for trusted storage–based licensing.
publisher.xml	Activation settings file that provides security keys for your products that use trusted storage–based licensing. These keys also enable the use of Secure Data Types (SDTs) in license file–based licensing products built with this same toolkit.
sdt_publisher.xml	Activation settings file that provides security keys for your products that are built with a toolkit that has rights to license file–based licensing only *and* that use SDTs. (Customers with this set of rights do not receive the `publisher.xml` file.)

Generated Components

The files listed in Table 1-3 are generated as a result of the licensing toolkit build process. Building the Production Licensing Toolkit describes the build process in detail.

Table 1-3 • Generated Components

Windows	UNIX	LF/ TS	Description
Intermediate and System Components			
`lm_new.c`	`lm_new.c`	LF/ TS	Publisher-specific source code module, generated by `lmnewgen.exe`.
`lm_new.obj`	`lm_new.o`	LF/ TS	Intermediate file linked into the FlexEnabled application, license generators, and the vendor daemon. The file contains publisher- specific parameters.
N/A	`lm_new_pic.o`	LF/ TS	PIC version of `lm_new.o` to support the dynamically linked model (for use with shared libraries).
`lmgr11.dll`	N/A	LF	Client library supporting the dynamically linked model. (The library is used specifically to support the use of native hostids with the Java toolkit.) See Dynamically Linking Your Application for further details.
`lmnewgen.exe`	`lmnewgen`	LF/ TS	Program that generates `lm_new.c`, which, in turn, is used to generate `lm_new.o` (UNIX) or `lm_new.obj` (Windows).
`lmseeds.h`	`lmseeds.h`	LF/ TS	Header file required by license-generator applications.
`lsrvend.c`	`lsrvend.c`	LF/ TS	File generated during build of vendor daemon and containing compile-time errors based on errors in `lsvendor.c`. For more information about the vendor daemon, see the *Programming Reference for License File–Based Licensing*.
<vendor>.exe	*<vendor>*	LF/ TS	The vendor daemon executable. This file depends on `lm_new.obj` and `lsvendor.c`.
FlexNet Licensing API Examples			

Table 1-3 • Generated Components

Windows	UNIX	LF/ TS	Description
`lmflex.exe`	`lmflex`	LF/ TS	Example application using Flexible API. (Its source code is found in the `machind` directory.) This example application is built in two forms: built with `makefile`, it uses license files; built with `makefile.act`, it uses trusted storage.
`lmflex_libFNP.dll`	`lmflex_libFNP.so`	TS	Activation library that enables `lmflex` to access trusted storage.
`installanchorservice.exe`	n/a	TS	Sample application used to install the FlexNet Licensing Service on Windows.
`uninstallanchorservice.exe`	n/a	TS	Sample application used to uninstall the FlexNet Licensing Service on Windows.
`ezcalcsdtplus.exe`	`ezcalcsdtplus`	TS	FlexEnabled application example that uses trusted storage and Secure Data Types (SDTs).
`ezcalcsdtplus_libFNP.dll`	`ezcalcsdtplus_libFNP.so`	TS	Activation library that allows `ezcalcsdtplus` to access trusted storage and use SDTs.
`ezcalcsdt.exe`	`ezcalcsdt`	TS	FlexEnabled application example that uses trusted storage and Secure Data Types (SDTs).
`ezcalcsdt_libFNP.dll`	`ezcalcsdt_libFNP.so`	TS	Activation library that allows `ezcalcsdt` to access trusted storage and use SDTs.
`ezcalc.exe`	`ezcalc`	TS	FlexEnabled application example that uses trusted storage.
`ezcalc_libFNP.dll`	`ezcalc_libFNP.so`	TS	Activation library that allows `ezcalc` to access trusted storage.
`lmflexsdt.exe`	`lmflexsdt`	LF	FlexEnabled application example that uses license file–based licensing and Secure Data Types (SDTs). ***Note •*** *This sample must be built and prepared. To do so, run this command from* `<platform_dir>`*:* `nmake SDT=1` (Windows) `make SDT=1` (UNIX)
`lmflex_coavail.exe`	`lmflex_coavail`	LF/TS	FlexEnabled application example that uses license file–based and trusted storage file -based.

Table 1-3 • Generated Components

Windows	UNIX	LF/ TS	Description
`lmflexsdt_libFNP.dll`	`lmflexsdt_libFNP.so`	LF	Run-time component prepared for `lmflexsdt` to handle SDTs.
`serveractutil.exe`	`serveractutil`	TS	Example server activation utility that manages license activations in trusted storage on license servers.
`serveractutil_libFNP.dll`	`serveractutil_libFNP.so`	TS	Activation library that allows `serveractutil` to access trusted storage.
`servercomptranutil.exe`	`servercomptranutil`	TS	Example activation utility that supports composite transactions to manage license activations in trusted storage on license servers.
`servercomptranutil`	`servercomptranutil_libFNP.dll`	TS	Activation library that allows `servercomptranutil` to access trusted storage.
`appactutil.exe`	`appactutil`	TS	Example application activation utility that manages license activations in trusted storage on FlexEnabled clients.
`appactutil_libFNP.dll`	`appactutil_libFNP.so`	TS	Activation library that allows `appactutil` to access trusted storage.
`appcomptranutil.exe`	`appcomptranutil`	TS	Example activation utility that supports composite transactions to manage license activations in trusted storage on FlexEnabled clients.
`appcomptranutil_libFNP.dll`	`appcomptranutil_libFNP.so`	TS	Activation library that allows `appcomptranutil` to access trusted storage.
Licensing Utilities			
`asrgen.exe`	`asrgen`	TS	Program that creates an activation specification record (ASR). For information about `asrgen`, see the *Programming Reference for Trusted Storage–Based Licensing*.
`lmcrypt.exe`	`lmcrypt`	LF/ TS	Program that creates feature definition line signatures based on a template. This file depends on `lm_code.h` and `lm_new.obj`. For information about `lmcrypt`, see the *Programming Reference for License File–Based Licensing*.
`tsreset_svr.exe`	`tsreset_svr`	TS	Utility that clears license rights from trusted storage or removes trusted-storage files on the license server.

Table 1-3 • Generated Components

Windows	UNIX	LF/ TS	Description
`tsreset_svr_libFNP.dll`	`tsreset_svr_libFNP.so`	TS	Activation library that allows `tsreset_svr` to access trusted storage.
`tsreset_app.exe`	`tsreset_app`	TS	Utility that clears license rights from trusted storage or removes trusted-storage files on a FlexEnabled client.
`tsreset_app_libFNP.dll`	`tsreset_app_libFNP.so`	TS	Activation library that allows `tsreset_app` to access trusted storage.

Standard Components

The standard components are used as shipped and do not require modification. They are listed in the following sections:

- License Server Manager (lmadmin and lmgrd)—The program installed on a server and used to serve licenses to FlexEnabled clients
- Flexible API—Files required to use the Flexible API
- Activation API and Activation Borrow API—Files required to use the Activation API or Activation Borrow API
- Secure Data Types—Files required to add extra security using SDTs
- FlexNet Licensing Service API—Files required to install the FlexNet Licensing Service API
- Utilities—Utilities for managing a license server and also for use with the licensing toolkit
- FlexNet ID Drivers—Files for installing FlexNet ID drivers
- Windows Event-Logging—Files required for implementing Windows event logging on a license server that includes `lmgrd`.

License Server Manager (lmadmin and lmgrd)

The toolkit includes the latest version of the license server manager `lmadmin`. An older license server manager, `lmgrd` (`lmgrd.exe` on Windows), also ships with the toolkit.

Flexible API

The following table lists standard components that may be required to use the Flexible API on Windows and UNIX platforms. Components marked with * are required when applications use certificate licensing only. Components marked with # are required when applications use certificate licensing and trusted storage licensing. All other components are required in either case. Include the appropriate dongle library or dongle stub library, depending on whether or not your FlexEnabled application uses licenses bound to a hardware dongle.

Table 1-4 • Standard Components for Flexible API

Description	Windows	UNIX
FlexNet Publisher client libraries ***Important*** *• Use a non-TRL library (lmgr.lib or liblmgr.a) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients. For more information about TRL, see the Programming Reference for License File–Based Licensing, chapter 9, "License Signature Configurations".*	• `libcrvs.lib` • `libsb.lib` • `lmgr.lib` or `lmgr_trl.lib` • `activation\lib\libnoact.lib` * • `activation\lib\libact.lib` # • `libFNPload.lib` # • Either `lmgr_dongle.lib` (for dongle usage) or `lmgr_dongle_stub.lib` ***Note*** *• Use these libraries whether you are building an /MT or an /MD application.*	• `libcrvs.a` • `libsb.a` • `liblmgr.a` or `liblmgr_trl.a` • `activation/lib/libnoact.a` * • `activation/lib/libact.a` # • `libFNPload.a` # • `libFNPload_notr.a` # • either `liblmgr_dongle.a` or `liblmgr_dongle_stub.a`
FlexNet Publisher client libraries used for linking Licensing into shared libraries. (Not available on all UNIX platforms.) ***Important*** *•* • *Note that the libraries liblmgr_nomt_pic.a and liblmgr_nomt_pic_trl.a have been deprecated and may be removed in a future release.* • *Use a non-TRL library (liblmgr_pic.a or liblmgr_nomt_pic.a) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*	N/A	• `libcrvs_pic.a` • `liblmgr_pic.a` or `liblmgr_pic_trl.a` • `liblmgr_nomt_pic.a` or `liblmgr_nomt_pic_trl.a` • `libsb_pic.a` • `activation/lib/libnoact_pic.a` * • `activation/lib/libFNPload_pic.a` *

Table 1-4 • Standard Components for Flexible API

Description	Windows	UNIX
FlexNet Publisher client library used with applications that are not able to use native multithreaded libraries: disables multithreaded heartbeats. ***Important •*** • *Note that these libraries have been deprecated and may be removed in a future release.* • *Use a non-TRL library (liblmgr_nomt.a) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*	N/A	• `liblmgr_nomt.a` • `liblmgr_nomt_trl.a`

Table 1-4 • Standard Components for Flexible API

Description	Windows	UNIX
System Libraries required for Windows platform	• `oldnames.lib` • `comctl32.lib` • `wsock32.lib` • `netapi32.lib` • `Rpcrt4.lib` • `oleaut32.lib` • `Ole32.lib` • `Wbemuuid.lib` • `wintrust.lib` • `crypt32.lib` • `Ws2_32.lib` • `psapi.lib` • `Shlwapi.lib` • `kernel32.lib` • `user32.lib` • `gdi32.lib` • `winspool.lib` • `comdlg32.lib` • `advapi32.lib` • `shell32.lib` • `ole32.lib` • `oleaut32.lib` • `uuid.lib` • `odbc32.lib` • `odbccp32.lib` • `dhcpcsvc.lib` **Visual Studio 2015 and later** • `legacy_stdio_wide_specifiers.lib` • `legacy_stdio_definitions.lib` • `libvcruntime.lib` • `ucrt.lib` • `Dhcpcsvc.lib`	

Activation API and Activation Borrow API

The following table lists standard components that may be required to use the Activation API or the Activation Borrow API.

Table 1-5 • Standard Components for Activation API and Activation Borrow API

Description	Windows	UNIX
Activation API and Activation Borrow API library	`activation\lib\libact.lib` ***Note*** • *Use this library whether you are building an /MT or an /MD application.*	`activation/lib/libact.a`
Library used for loading the activation library (run-time component) created by `preptool`.	`libFNPload.lib`	• `libFNPload.a` • `libFNPload_notr.a`
Shared version of the library used for loading the activation library. (Not available on all UNIX platforms.)	N/A	`activation/lib/libFNPload_pic.a`
Module that provides Soap connection. This is required for transactions with FlexNet Operations.	`publisher\FNPCommsSoap.dll`	`FnpCommsSoap.so`

Secure Data Types

The following libraries are required for applications with Secure Data Types (SDTs) that use trusted-storage licensing:

- **On Windows**—`libact.lib` and `libFNPload.lib`
- **On UNIX**—`libact.a` or `libact_pic.a` and `libFNPload.a` or `libFNPload_pic.a`

FlexNet Licensing Service API

The toolkit includes the following API to install and uninstall the FlexNet Licensing Service:

- `FNP_Act_Install.lib`
- `FNP_Act_Install.dll`

Utilities

The following table lists the standard utilities provided within the toolkit.

Table 1-6 • Standard Utilities

Description	Windows	UNIX
Random number generator used for `LM_SEEDs` in `lm_code.h`.	`lmrand1.exe`	`lmrand1`
Licensing utilities for managing a FlexNet license server.	`lmutil.exe`	`lmutil`
Licensing utilities. Each is a copy (or link on UNIX) of the `lmutil` executable, but renamed with the specific command name. See the *License Administration Guide* for usage information.	• `lmhostid.exe` • `lmver.exe` • `lmdown.exe` • `lmnewlog.exe` • `lmremove.exe` • `lmreread.exe` • `lmswitchr.exe` • `lmstat.exe` • `lmborrow.exe` • `lmdiag.exe` • `lminstall.exe` • `lmpath.exe`	• `lmhostid` • `lmver` • `lmdown` • `lmnewlog` • `lmremove` • `lmreread` • `lmswitchr` • `lmstat` • `lmborrow` • `lmdiag` • `lminstall` • `lmpath`
GUI for end-user `lmgrd` utilities. `lmgrd` is the older license server product that `lmadmin` is replacing.	`lmtools.exe`	N/A
Utility that obfuscates and adds security to the FlexEnabled application executable. See Using lmstrip for Additional Security for more information.	`lmstrip.exe`	`lmstrip`
Service installer for the older license server product `lmgrd`, which `lmadmin` is replacing.	`installs.exe`	N/A
Utility used to prepare applications so that they can access trusted storage or security devices such as SDTs. For trusted-storage licensing, `preptool` creates a product-specific activation library component that manages trusted storage and security features, including SDTs. For license file–based licensing, it simply creates the product-specific run-time component that handles SDTs.	`preptool.exe` Requires `preparchive_<build_no>.yaa`	`preptool` Requires `preparchive_<build_no>.yaa`

FlexNet ID Drivers

For a complete list of the FlexNet ID driver files included in the toolkit, refer to the *Driver Installation Guide for FlexNet ID Dongles*.

Note • *Dongle drivers are not compliant with Linux Standard Base (LSB) certification. Therefore, omit dongle support if you plan to submit your FlexEnabled application to the Linux Foundation for LSB certification. For more information about LSB, see the Linux Foundation's website http://www.linuxfoundation.org.*

Vendor Daemon Libraries

The following vendor daemon libraries are used to build the vendor daemon:

- On Windows—`lmgras.lib` and `lmgrs.lib`
- On UNIX—`liblmgr_as.a` and `liblmgr_s.a`

Windows Event-Logging

The following table lists files related to Windows event-logging that are included in the toolkit. See Enabling Support for Windows Event-Logging for a description of how to use these files.

Table 1-7 • Windows Event-Logging

Description	Windows	UNIX
Registration entry templates for adding and removing support for Windows Event Logging	• `eventlogadd.reg` • `eventlogremove.reg` ***Note*** • *For* `lmadmin`*, these files are located in* `lmadmin\eventlog`*.*	N/A
Message-file DLLs containing localized text to describe license-server events	For `lmadmin`: • `FnplmLicenseServerMsgs.dll` for 32-bit `lmadmin` or `FnplmLicenseServerMsgs64.dll` for 64-bit `lmadmin` (located in `lmadmin\eventlog`) • `VendorLicenseServerMsgs.dll` For `lmgrd`: • `VendorLicenseServerMsgs.dll` • `FnplmLicenseServerMsgs.dll`	N/A

Source Code

Several of the components listed in Table 1-3 and Table 1-6 have corresponding source code. In most cases, these components are used as shipped or generated. The source code files are included in the Toolkit in the event that you need to customize them. Table 1-8 lists these files; they are located in the `machind` directory.

Table 1-8 • Source Code Files

Source Code File	Description
Utility and Vendor Daemon Sources	
`lsvendor.c`	Configuration file for vendor daemon options. Refer to the chapter "Customizing the Vendor Daemon" in the *Programming Reference for License File–Based Licensing* for details on modifying this file before you build the licensing toolkit. ***Important*** • *Contact Flexera Technical Support before modifying this file.*
FlexNet Licensing API Example Sources	
`lmflex.c`	Flexible API example source code.
License Generator Source	
`lmcrypt.c`	Source code for `lmcrypt`, which generates signatures for feature definition lines.
Windows Event-Logging	
For lmadmin: **`lmadmin\eventlog\`** **`FnplmLicenseServerMsgs.mc`** **`VendorLicenseServerMsgs.mc`** **For lmgrd:** • `VendorLicenseServerMsgs.mc` • `FnplmLicenseServerMsgs.mc`	Source code for strings used to describe license-server events. See Enabling Support for Windows Event-Logging for further details.

2

Testing the Production Licensing Toolkit With a License File

This chapter describes how to test production toolkit components and licensed applications that can access license rights in license files. Complete the tests described in this chapter if you have built a toolkit using either `makefile` or `makefile.act`.

Testing a FlexEnabled Application Using a License File

To test whether a FlexEnabled application created with the toolkit operates correctly, generate a license file and use it with the sample application `lmflex`.

If virtualization features are used in the license file, the FlexNet Licensing Service must be installed. For more information, see Preparing for the Test on page 48.

Task ***To test the FlexEnabled application***

1. From the `<platform_dir>` directory, open `uncounted.lic` in a text editor.

   ```
   # an uncounted license
   FEATURE f2 demo 1.000 permanent uncounted HOSTID=ANY \
       SIGN="00A7 9966 2256 351C CB60 248E 2654 B000 8724 73F4 2F49 C626 9A89 E103 DCB6"
   ```

2. Change the vendor name from `demo` to your vendor name, as provided by Flexera.
3. Save and close `uncounted.lic`.
4. Open a command window and change to the `<platform_dir>` directory.
5. Type **`lmcrypt uncounted.lic`**. This creates a new signature (`SIGN=`) based on your publisher-specific vendor name and encryption settings.
6. Run `lmflex.exe` (`lmflex` on UNIX) and try to check out feature **f2**.

The license file containing **f2** (`uncounted.lic`) is in the same directory as `lmflex` and has an extension of `.lic`, so it is found and used. `lmflex` reports that **f2** is checked out.

Testing the Vendor Daemon Using License Files

To test whether your vendor daemon created with the toolkit operates correctly, perform these testing phases:

- Phase 1—Generate a license file for a counted license for your vendor daemon.
- Phase 2—Start the license server using the license file created in step 1. (These instructions use `lmadmin` as the license server manager.)
- Phase 3—Use the sample application `lmflex` to check that the vendor daemon is operating correctly.

Creating a Publisher-Specific License File

The license server accesses a local license file to determine which vendor daemon to start up.

Task ***To create and sign the publisher-specific license file used to start the license server***

1. In the `<platform_dir>` directory make a copy of the file `counted.lic` and rename it to `mycounted.lic`.
2. Open `mycounted.lic` in a text editor.

   ```
   SERVER this_host ANY
   VENDOR demo
   USE_SERVER
   # a counted license
   FEATURE f1 demo 1.0 permanent 4 SIGN=”05E5 ....”
   ```

3. Edit the following elements:
   - Change the `VENDOR` value from `demo` to your vendor name as given to you by Flexera.
   - In the `VENDOR` line, add the path to the executable of your vendor daemon.
   - Change the feature name from `f1` to `f3`.
4. Save and close `mycounted.lic`.
5. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmcrypt mycounted.lic
   ```

Starting 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 -configDir <conf_path>
   ```

where <conf_path> is the path to the conf folder. configDir and <conf_path> are required only if the conf folder is not installed in the same location as lmadmin.exe.

2. 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.

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

Checking Out Licenses

Task ***To check out licenses***

1. Open a command window, and run lmflex.exe (lmflex on UNIX).
2. Check out feature f3.

   lmflex and the vendor daemon report that you have checked out **f3**.

3. Try to check out feature f1.

   f1 is available in counted.lic, which is for use by the demo vendor daemon and, therefore, not usable by your production vendor daemon.

   The vendor daemon and the license server report that this feature is not supported.

4. Stop the license server. To do so, from the <platform_dir> directory, enter the following:

   lmdown -c mycounted.lic

   This concludes the test.

3

Testing the Production Licensing Toolkit with Trusted Storage

This chapter describes how to test production toolkit components and licensed applications that can access license rights in trusted storage. Complete the tests described in this chapter if you have built a toolkit using `makefile.act`. You should also complete the tests in Testing the Production Licensing Toolkit With a License File.

About the Test

The following parts make up this test:

- Preparing for the Test
- Activating License Rights
- Borrowing License Rights
- Testing Vendor Daemon Use of Trusted Storage

What To Know Before Starting the Test

The following lists information that you should know before you start the test:

- For purposes of this test, the machine on which you built the toolkit serves as both the FlexEnabled client and the license server.
- During the test process, you assume the roles of both the end user and the publisher's activation server.
- Your toolkit is tested against the example product `ezcalc`, which comes with your toolkit.
- The Manual Response Generator, a test tool provided with the FlexNet Publisher toolkit, is used in this test to help simulate the processing performed by the publisher's activation server.
- The test uses `lmadmin` as the license server manager.
- Instructions for performing the test focus on UNIX-environment syntax to designate path and file names for toolkit components. For toolkits built on a Windows platform, simply substitute the appropriate path and file-name syntax for the toolkit components.

- Taking the end-user role during the actual activation phase of this test (see Activating License Rights), you invoke `servercomptranutil`, the example activation utility that handles composite-transaction activations between the activation client (in this case, the license server) and the publisher's activation server. If you have created your own server-side activation utility for composite transactions (and have configured it for prepped Trusted Configuration as described in Chapter 1, First Build of the Production Licensing Toolkit), you can substitute your utility for the `servercomptranutil`.

Important • *Best practice is always to use composite transactions for activation activities between the activation server and an activation client, such as the license server or a FlexEnabled client.*

Components and Files Tested

The tests verify the correct operation of the following components generated during a toolkit build:

- `lmcrypt`
- publisher-specific vendor daemon, `<vendor>`

The tests verify the correct operation of the following example utilities and applications that are rebuilt during a toolkit build:

- `installanchorservice`
- `appcomptranutil`
- `servercomptranutil`
- `ezcalc`
- `tsreset_svr`
- `tsreset_app`
- Manual Response Generator

For further information on the toolkit utilities used within this chapter, see the "Working with the Software Publisher Utilities" chapter in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Preparing for the Test

Perform the following tasks before starting the test:

- Installing the FlexNet Licensing Service on Linux, Installing the FlexNet Licensing Service (Windows), or Installing the FlexNet Licensing Service on OS X
- Resetting Trusted Storage
- Setting Up the Activation Response Parameter File

Installing the FlexNet Licensing Service on Linux

On Linux, the FlexNet Licensing Service comprises the FlexNet Licensing Service executable and the FlexNet Licensing Service daemon. The daemon is launched by the FlexNet Licensing Service executable.

The following command-line options are available when running `install_fnp.sh`.

Table 3-1 • `install_fnp.sh` options

Option	Description
`--nolsb`	Creates a symlink to the native loader to enable the FlexNet Publisher binaries to run without full lsb support on the target platform.
`--nodaemon`	Prevents the installation script from automatically starting the FlexNet Licensing Service daemon. This can be useful when using systemd.

Task ***To install the FlexNet Licensing Service***

1. Run `install_fnp.sh`.

   This shell script configures the FlexNet Licensing Service executable to run as a root-privilege setuid process.

2. The FlexNet Licensing Service daemon needs to run continuously. Therefore, a further installation step is required to ensure it is started a boot time. Because the FlexNet Licensing Service daemon does not need to run with root privilege, it can be started by adding the following line to a nominated user's crontab:

   ```
   @reboot /usr/local/share/FNP/service64/<version>/FNPLicensingService -r 2>&1 >/tmp/fnpd.log
   ```

Notes

- For the i86_lsb FlexNet Licensing Service daemon, the path is `/usr/local/share/FNP/service/<version>/FNPLicensingService`.
- The machine's administrator is free to redirect output as required.
- The machine's administrator may want to add an additional line to the crontab to run the above command at regular intervals to ensure that if the FlexNet Licensing Service daemon gets stopped for some unexpected reason, it will be restarted by the *crond*. The FlexNet Licensing Service daemon itself ensures there is never more than one instance running.
- Typical messages output on running the startup command are `Licensing Service daemon activated` and `Licensing Service daemon already active`.
- Rather than using cron, administrators may want to add a boot-script for the FlexNet Licensing Service daemon or optionally use systemd. The following commands are required:

  To start the FlexNet Licensing Service daemon

  ```
  /usr/local/share/FNP/service64/<version>/FNPLicensingService -r
  ```

Note • *The process ID of the daemon will be written to `/var/run/FNPLicensingService64.pid` (`/var/run/FNPLicensingService.pid` on 32-bit).*

To stop the FlexNet Licensing Service daemon

```
/usr/local/share/FNP/service64/<version>/FNPLicensingService -k
```

- When the Linux FlexNet Licensing Service daemon is not installed and an operation occurs that requires it, a descriptive error is displayed. For example:

  ```
  ./lmhostid -ptype VM -uuid
  ```

  results in

  ```
  lmhostid: The VM Host ID is not available. (-215,14704) - The FlexNet Licensing Service does not
  appear to be running.
  ```

Installing the FlexNet Licensing Service on OS X

On OS X, a shell script is used to configure the FlexNet Licensing Service executable to run as a root-privilege setuid process.

Task ***To install the FlexNet Licensing Service***

Run `install_fnp.sh`.

Installing the FlexNet Licensing Service (Windows)

On Windows, before trusted storage is created or accessed on a machine, the FlexNet Licensing Service must be installed on that machine. To prepare for the test, run the example program `installanchorservice.exe` to install the service on your machine.

Note • *For purposes of this test, you install the FlexNet Licensing Service once since both server-side and client-side trusted storage are configured on the same machine. In a live scenario, the FlexNet Licensing Service is installed separately on the FlexEnabled client machine and on the license-server machine to support trusted storage at each location.*

Important • *You must have administrator permissions to install the FlexNet Licensing Service on Windows.*

Task ***To install the FlexNet Licensing Service***

1. Enter the following at a command prompt from the `<platform_dir>` directory:

   ```
   installanchorservice <vendor> ezcalc
   ```

   where `<vendor>` is your vendor name.

2. You receive the following confirmation message on a system that already has a version of the service installed:

```
Successfully installed anchor service for publisher <vendor>, product ezcalc
Success code 2002: The new service is the same as, or older than, the installed
service so no upgrade is required.
```

3. You receive the following confirmation message on a clean system

```
Successfully installed anchor service for publisher <vendor>, product ezcalc
Success code 2003: There was no service on the system so the new one was install
ed.
```

For information about the APIs used to install the FlexNet Licensing Service at the customer site, see the "FlexNet Licensing Service API" chapter in *C/C++ Function Reference*.

Uninstalling the FlexNet Licensing Service (Windows)

Task ***To uninstall the FlexNet Licensing Service from a system where more than one users of the service are registered***

1. Enter the following at a command prompt from the `<platform_dir>` directory:

   `uninstallanchorservice <vendor> ezcalc`

   where `<vendor>` is your vendor name.

2. You receive the following confirmation message:

```
Error uninstalling anchor service for publisher ezcalc, product fnp
Error code 1011: The service is still being referenced by another application so
was not removed.
```

For information about the APIs used to uninstall the FlexNet Licensing Service at the customer site, see the "FlexNet Licensing Service API" chapter in *C/C++ Function Reference*.

Note • *Even though the FlexNet Licensing Service was not removed, its installation reference count will have been decremented.*

Task ***To uninstall the FlexNet Licensing Service from a system where only one user of the service is registered***

1. Enter the following at a command prompt from the `<platform_dir>` directory

   `uninstallanchorservice <vendor> ezcalc`

   where `<vendor>` is your vendor name.

2. You receive the following confirmation message:

```
Successfully uninstalled anchor service for publisher demo, product fnp
Success code 2004: The installed service has been successfully removed from the
system.
```

For information about the APIs used to uninstall the FlexNet Licensing Service at the customer site, see the "FlexNet Licensing Service API" chapter in *C/C++ Function Reference*.

Resetting Trusted Storage

To ensure a consistent testing process, remove any trusted-storage files (server-side or client-side) that might exist from previous demonstration or trial-licensing activities on the machine where your toolkit resides. If this is the first time you have built a production toolkit, you can skip this section.

Note • *On UNIX hosts, set the environment variable,* `LD_LIBRARY_PATH`*, to the current working directory before proceeding. Use a command similar to the following:* `setenv LD_LIBRARY_PATH.`

Task ***To reset trusted storage***

1. To remove any server-side trusted-storage files previously created for demonstration or trial-licensing activities, enter the following from the `<platform_dir>` directory:

   `tsreset_svr -delete all`

   This produces output ending with the following information:

   ```
   Performing reset operation, please wait...
   Trusted Storage Contents have been reset...
   Performing anchor reset operation, please wait...
   Orphan anchors have been reset.
   ```

2. To verify that no server-side trusted-storage information exists, enter the following at a command prompt from the `<platform_dir>` directory:

   `servercomptranutil -v`

   The following message is displayed if no fulfillment records are present:

   ```
   No fulfillment records in trusted storage.
   ```

3. To remove any client-side trusted-storage files previously created for demonstration or trial-licensing activities, enter the following from the `<platform_dir>` directory:

   `tsreset_app -delete all`

   The output produced is similar to that for deleting server-side trusted storage.

4. To verify that no client-side trusted-storage information exists, enter the following at a command prompt from the `<platform_dir>` directory:

   `appcomptranutil -v`

   The following message is displayed if no fulfillment records are present:

   ```
   No fulfillment records in trusted storage.
   ```

Setting Up the Activation Response Parameter File

The publisher's activation server sets up an activation response parameter file containing license rights and other publisher-specific information that it needs to generate a response to an activation request. To simulate the role of the activation server, you must set up this response parameter file. Later, during the test, you use the Manual Response Generator to simulate the activation server's role in actually creating an activation response from this parameter file.

Setting up the response parameter file for this test involves the following procedures. These procedures use the sample files and licensed features that come with the `ezcalc` example product in the toolkit.

Note that the response parameter file can contain multiple response actions. An activation action is always required. If your activation utility does not use prepped Trusted Configuration, then a configuration action (see Specifying the Trusted Configuration) is also required.

- Specifying the Trusted Configuration (not needed if you are using a prepped Trusted Configuration)
- Creating Feature Definition Lines Containing Publisher-Specific Signatures
- Adding Signed Feature Definition Lines to and Modifying the Activation Response

Specifying the Trusted Configuration

If you are implementing a non-prepped Trusted Configuration, use this procedure to incorporate the configuration details into the response parameter file. This information is used to create the specified trusted section in trusted storage. For more information about non-prepped Trusted Configuration, see Defining Trusted Configuration (Trusted Storage Only) in Chapter 1, First Build of the Production Licensing Toolkit.

Note • *The following procedure uses the file* `<platform_dir>/publisher/<vendor>TrustedConfigOut.xml` *as the source for the non-prepped Trusted Configuration information needed to configure trusted storage on a physical machine and a virtual machine. Substitute this file name in the following instructions.*

Task ***To specify the Trusted Configuration in the response parameter file***

1. Open `<platform_dir>/publisher/<vendor>TrustedConfigOut.xml`.
2. Copy `<platform_dir>/publisher/xml/Version6/W7_EZCalcSvrSingleStepActivate1.xml` to a new file named `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivate1.xml` where `<vendor>` is your vendor name.
3. Open this new file and locate the `<ConfigData>` element (which starts with `<ConfigData>` and ends with `</ConfigData>`).
4. Replace the contents of this element with the entire `<ConfigData>` element found within the `<ServedConfig>` element of `<platform_dir>/publisher/<vendor>TrustedConfigOut.xml`.

   Note • *The TrustedConfigOut.xml file contains two* `<ConfigData>` *elements—one within* `<ServedConfig>` *and one within* `<AsrConfig>`*. Make sure you copy the* `<ConfigData>` *element within* `<ServedConfig>`*.*

5. Save and close both `<platform_dir>/publisher/<vendor>TrustedConfigOut.xml` and `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivate1.xml`.

Creating Feature Definition Lines Containing Publisher-Specific Signatures

This parameter file contains the license rights that are loaded as a single fulfillment record into trusted storage. Within this fulfillment record, the license rights are represented as feature definition lines.

Use this procedure to modify the fulfillment record within the response you are creating for the test. This involves extracting the feature definition lines for `ezcalc` features from a sample response parameter file, signing these lines with publisher-specific information (just as the activation server would do), and reincorporating them in a fulfillment record within your own publisher-customized response parameter file.

Task ***To create ezcalc feature definition lines signed with your publisher-specific keys***

1. Locate or create the appropriate response parameter file:
   - If you are implementing a prepped Trusted Configuration, copy `<platform_dir>/publisher/xml/Version6/W7_EZCalcSvrSingleStepActivatePreppedConfig1.xml` to a new file named `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivatePreppedConfig1.xml` where `<vendor>` is your vendor name. Open this new file.
   - If you are implementing a non-prepped Trusted Configuration, open the `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivate1.xml` file created in the previous section, Specifying the Trusted Configuration.
2. Copy `<platform_dir>/counted.lic` to a new file named `<platform_dir>/test.lic`. Leave this new file open.
3. Locate and copy the five `INCREMENT` lines towards the end of the appropriate response parameter file opened in Step 1.
4. Append the five INCREMENT lines from Step 3 to the end of `test.lic`. The file now looks something like:

```
SERVER this_host ANY
VENDOR demo
#a counted license
FEATURE f1 demo 1.0 permanent 4 SIGN="003D B062 54D1 47E3 1FF6 145F \
 59CE 0200 8002 FE69 0FBC F4FF AA10 FFB3 6EB3"
INCREMENT ADD demo 1.0 31-dec-2020 1 TS_OK SIGN="005F 03B6 1B41 C196 \
 9754 62AB 5F23 5500 3153 E210 952D B005 BAB4 E5C0 B333"
INCREMENT SUBTRACT demo 1.0 31-dec-2020 1 TS_OK SIGN="00C8 A83F 8D17 \
 545A 497E 01EF 341D 1E00 76D0 4EF2 282D 9434 B9A8 F2D4 16C6"
INCREMENT MULTIPLY demo 1.0 31-dec-2020 1 TS_OK SIGN="007A 413F 3BD4 \
 AA6D 4516 9BE1 A271 1800 4305 535F 0F7E 24E4 AB73 83D8 7A00"
INCREMENT DIVIDE demo 1.0 31-dec-2020 1 TS_OK SIGN="0075 E4A9 80A5 \
 522C 0E68 2165 6CE8 CA00 C8B3 4610 2A99 3138 D9DD 4ACA 5E0C"
INCREMENT AVG demo 1.0 31-dec-2020 1 TS_OK SIGN="0075 D4B8 6C08 13A9 \
 7CD6 1A3D B6EE 8800 352B 6658 CF5C 4F27 BD9B 445A EDCE"</Value>
```

5. In `test.lic`, change all occurrences of `demo` to your vendor name.

   Note • *Check the expiration dates. You may need to change the expiration date within the* `INCREMENT` *lines if you are using these files some time after the toolkit was first issued.*

6. In the response parameter file opened in Step 1, locate the `<Key>serverHostID</Key>` element, and copy the hostid from the `<Value>` element nested in this `<Key>` element.

7. In `test.lic`, locate the `SERVER` line and replace `ANY` with the value of the hostid copied from the response parameter file. The resulting `SERVER` line should look something like:

   ```
   SERVER this_host INTERNET=0.0.0.0
   ```

   This hostid is used when generating the signatures for the feature definition lines and must match the hostid specified in the fulfillment record for signature verification to be successful.

8. Save and close `test.lic`. Continue to leave the response parameter file open.

9. Sign the license rights you just created in `test.lic`. To do so, at a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmcrypt test.lic
   ```

   There is no confirmation output.

Adding Signed Feature Definition Lines to and Modifying the Activation Response

Use this procedure to do the following:

- Replace the license rights within the fulfillment record in the response parameter file with the updated rights
- Make additional modifications to the fulfillment record

Task ***To add signed feature definition lines to and modify the activation response***

1. Reopen `test.lic`, and locate the five `INCREMENT` lines.

2. Replace the five `INCREMENT` lines in response parameter file opened in the previous section with the five from `test.lic`. Make sure you are updating the correct response parameter file:

   - For prepped Trusted Configuration, `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivatePreppedConfig1.xml`
   - For non-prepped Trusted Configuration, `<platform_dir>/publisher/xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivate1.xml`

3. In the same response parameter file, locate the `<FulfillmentId>` element, and change its value to **`FIL-test`**. This line now looks like this:

   ```
   <FulfillmentId>FIL-test</FulfillmentId>
   ```

4. Locate the `expiry` `<Key>` element. If necessary, edit the value of this element, which defines the expiration date of the license rights specified by the response file. For example, you might want to change the value to the following:

   ```
   <Key>expiry</Key>
   <Value>31-dec-2020</Value>
   ```

   Note • *This expiration date is fixed once the response is generated. If you need to change the expiration date to match the expiry date specified in the activation request, do so by editing the response parameter file after the request is received and before the response is generated.*

5. Locate the `ProductID` `<Key>` element. Change the value to **`EZCALC-test`**. These lines now look like this:

   ```
   <Key>ProductId</Key>
   <Value>EZCALC-test</Value>
   ```

6. Locate the `EntitlementId` `<Key>` element. Change the value to **`ENTL-EZCALC-test1`**. These lines now look like this:

```
<Key>EntitlementId</Key>
<Value>ENTL-EZCALC-test1</Value>
```

7. In the `<FlexNetDictionary>` section (after the `INCREMENT` lines), locate the `COMMENT` `<Key>` element. Change the value to **`Single Step Activate - Loading 10 Hybrid Licenses for EZCalculator Test1`**. These lines now look like this:

```
<Key>COMMENT</Key>
<Value>Single Step Activate - Loading 10 Hybrid Licenses for EZCalculator Test1</Value>
```

8. Save and close the response parameter file.

9. Delete `test.lic`.

Activating License Rights

In this section, you use the `responsegen` command-line utility in conjunction with the example server activation utility (`servercomptranutil`) to activate license rights on the license server. This process includes issuing an activation request, generating an activation response that contains a fulfillment record, and processing the response to add the fulfillment record to trusted storage.

This process simulates (although in a manual format) what occurs automatically between the license server (requesting the activation) and the activation server, such as the FlexNet Operations server or a FlexNet Publisher License Generator system.

Task ***To load license rights into trusted storage***

1. Using the server activation utility, generate the activation request. To do so, enter the following at a command prompt from the `<platform_dir>` directory:

```
servercomptranutil -new activate_request.xml -activate ENTL-EZCALC-test1 \
    expiration=31-dec-2020 hybrid=10
```

This command assumes that the Entitlement ID, `ENTL-EZCALC-test1`, has been communicated between the publisher and end user. The following response is displayed:

```
Request written to activate_request.xml
```

2. Create the response. To do so, enter the following from the `<platform_dir>/publisher` directory:

```
responsegen -i ../activate_request.xml -o ../activate_response.xml \
    -k ./xml/<vendor>KeyFile.xml \
    -m ./xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivatePreppedConfig1.xml
```

This command assumes you are doing the following:

- Implementing prepped Trusted Configuration. If you are implementing non-prepped Trusted Configuration, specify **`./xml/Version6/<vendor>_W7_EZCalcSvrSingleStepActivate1.xml`** instead.
- Activating licenses on a physical machine. Therefore, you specify the **`KeyFile.xml`** to provide the keys for signing the response. If you are activating licenses on a virtual machine, specify **`KeyFileVmid.xml`** instead. (The key file was generated when you encrypted Trusted Configuration in Defining Trusted Configuration (Trusted Storage Only) in Chapter 1, First Build of the Production Licensing Toolkit.)

No confirmation message is output.

3. Process `activate_response.xml` with the server activation utility. To do so, from the `<platform_dir>` directory, enter the following:

   `servercomptranutil -process activate_response.xml`

   The following confirmation message is displayed:

```
Response processed successfully. Actions were:
        Create         fulfillment "FIL-test"

    FlexNet response dictionary:
                   COMMENT=Single Step Activate - Loading 10 Hybrid Licenses for EZCalculator
                   Test1
```

   This output confirms that ten hybrid licenses for EZCalculator have been loaded into the license server. You can now view the contents of trusted storage.

4. To confirm the contents of trusted storage, enter the following:

   `servercomptranutil -v format=long`

   The output should be similar to the following, showing a fulfillment record for ten hybrid licenses of the EZCalculator:

```
-------------------------------------------------------------------
Trust Flags: 0x7 Fully Trusted
Fulfillment Type: Publisher Activation
Status: Enabled
Fulfillment Id: FIL-test
Entitlement Id: ENTL-EZCALC-test1
Product Id: EZCALC-test
Expiration Date: 31-Dec-2020
Suite ID: NONE
Concurrent: 0
Concurrent overdraft:0
Hybrid: 10
Hybrid overdraft: 0
Activatable: 0
Activatable overdraft: 0
Repair: 0
Feature Line(s):

INCREMENT ADD Company2 1.0 31-dec-2020 1 TS_OK SIGN="037B 15C6 8B42 D486 \
8414 73FA 7D26 7318 9003 F504 033D C865 F0A4 G8D4 C211"
INCREMENT SUBTRACT Company2 1.0 31-dec-2020 1 TS_OK SIGN="00B2 C766 C43A \
3A41 E981 63ED A3D8 4800 BDC5 4C48 7981 F0BF 9A39 48DF EB1A"
INCREMENT MULTIPLY Company2 1.0 31-dec-2020 1 TS_OK SIGN="00B1 75C6 6481 \
2C18 AC4D 87FE 6C23 F400 166E 1586 07FA 827E ED0C B480 8B94"
INCREMENT DIVIDE Company2 1.0 31-dec-2020 1 TS_OK SIGN="00FF 515A 3015 \
4ABE 8413 02C6 1214 E100 3670 E9C2 8DFD D2EF D46E 7656 D104"
INCREMENT AVG Company2 1.0 31-dec-2020 1 TS_OK SIGN="0162 A490 3P38 75D6 \
1DA8 4B7E E6FC 7941 450L 0721 FF6A 1D12 CD0F 2419 DFCD"

-------------------------------------------------------------------
```

Borrowing License Rights

A FlexEnabled application can request a temporary license from license rights held in trusted storage on a license server. The activation process writes the license rights locally into client-side trusted storage. The client can then disconnect from the network and continue to use the FlexEnabled application until the expiration date. On reaching the expiration date, the license rights are returned to the license pool on the license server. This capability (for activatable and hybrid licenses) is usually referred to as *borrowing* and implemented using the Activation API (see the *C/C++ Function Reference* for implementation information).

Note • *Do not confuse borrowing of license rights held in trusted storage with the borrowing functionality that is provided by using the BORROW keyword, which is used in license file–based licensing. For more information on the following topics:*

- ***Borrowing license rights held in trusted storage****—see the Programming Reference for Trusted Storage–Based Licensing*
- ***License borrowing using the BORROW keyword****—see the License Administration Guide, section* License Borrowing with BORROW

Setting the Borrow Reclaim Count

It is possible that borrowed licenses can no longer be accessed, for example, if the computing device that had borrowed the license rights is stolen, lost, or damaged. By default, the local license server does not return the license rights to the license pool until the borrow interval has expired. Depending on the specified borrow interval, this can result in a long period during which the license rights—and hence the FlexEnabled application—are unavailable. For this scenario, the publisher can allow license administrators to reclaim a certain number of licenses (expressed as a percentage of the total number of licenses) before the borrow interval expires.

This *borrow reclaim count* is specified using the `ls_ts_borrow_reclaim_percentage` vendor variable. For more information about this variable, see the *Programming Reference for Trusted Storage–Based Licensing*.

If the publisher has allowed the reclaim of a certain number of borrowed licenses, the license administrator can run the `lmremove` utility. This will remove the deduction records from the server-side trusted storage for the specified client. For more information about the `lmremove` utility, see the *License Administration Guide*, section *lmremove (in Trusted Storage–Based Licensing)*.

Resetting the Borrow Reclaim Count

If the maximum number of allowed license borrow reclaims (calculated as a percentage; for example, 10% of total licenses) is reached, it is at the publisher's discretion to allow the enterprise to reset the borrow reclaim count. This is a licensing lifecycle event, implemented by a transaction.

The borrow reclaim count is reset by deleting the fulfillment "FNP_SPFR_1" from the server-side trusted storage.

The following procedure demonstrates resetting the borrow reclaim count using the sample utilities from the toolkit.

Task ***To reset the borrow reclaim count***

1. Using the server activation utility, generate the borrow reset request. To do so, enter the following at a command prompt from the `<platform_dir>` directory:

   ```
   servercomptranutil -new request.xml reference=BorrowClaimReclaim
   ```

2. Create the response. To do so, enter the following from the `<platform_dir>/publisher` directory:

   ```
   responsegen -i request.xml -o response.xml -m xml\Version6\W7_ActivationBorrowReclaimReset.xml -k
       xml\DemoKeyout.xml
   ```

   In this step, the `responsegen` utility generates the response file, `response.xml`. In a production scenario, the response would be generated by the FlexNet Operations server or a FlexNet Publisher License Generator system.

   `DemoKeyOut.xml` is the transaction key file for the `demo` publisher keys used by the kit.

   If the kit has been rebuilt with your own publisher name, replace `DemoKeyout.xml` with your key file for trusted section 1. (The key file was generated when you encrypted the Trusted Configuration. For more information, see Defining Trusted Configuration (Trusted Storage Only) in Chapter 1, First Build of the Production Licensing Toolkit.)

3. Process `response.xml` with the server activation utility. To do so, from the `<platform_dir>` directory, enter the following:

   ```
   servercomptranutil -process response.xml
   ```

   This step deletes the fulfillment record “FNP_SPFR_1” which resets the borrow reclaim count.

To deny the reset, use parameter file `W7_ActivationBorrowReclaimResetDeny.xml` at Step 2.

Testing Vendor Daemon Use of Trusted Storage

This section attempts to check out licenses held in the license server’s trusted storage to test vendor daemon operation. The test uses the example licensed application, `ezcalc`. To complete this section you must have the required fulfillment record in trusted storage.

In this section you do the following:

- Creating a Publisher-Specific License File
- Starting the License Server
- Checking Out a License Held in Trusted Storage

Creating a Publisher-Specific License File

The license server accesses a local license file to determine which vendor daemon to start up.

Task ***To create and sign the publisher-specific license file used to start the license server***

1. From the `lmadmin` installation directory, go to the `demo` directory, and copy `demo.lic` to `<platform_dir>\<vendor>_test1.lic` in FlexNet Publisher. The `<vendor>` value is your vendor name.
2. In the `<platform_dir>` directory, open `<vendor>_test1.lic` in a text editor.
3. Change all occurrences of `demo` to your vendor name.
4. Save and close `<vendor>_test1.lic`.
5. At a command prompt from the `<platform_dir>` directory, enter the following:

   ```
   lmcrypt <vendor>_test1.lic
   ```

6. Move `<vendor>_test1.lic` to the directory containing your vendor daemon under the `lmadmin` installation directory.

Starting 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 -configDir <conf_path>
   ```

   where `<conf_path>` is the path to the `conf` folder. `configDir` and `<conf_path>` are required only if the `conf` folder is not installed in the same location as `lmadmin.exe`.

2. 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.

3. Select the Administration page, and sign in with your administrator credentials.
4. Go to the Vendor Daemon Configuration tab, and click **Import License**.
5. Click **Browse** to select the license file `<vendor>_test1.lic`.
6. Click **Import License** to import the license file which specifies the location of the vendor daemon.

Checking Out a License Held in Trusted Storage

Task ***To check out a license***

1. In a new command window, from the `<platform_dir>` directory in FlexNet Publisher, invoke EZCalculator by typing **`ezcalc`**. The usage is displayed:

```
EZ Calculator
Current value:      0

Enter   + to add
        - to subtract
        x to multiply
        / to divide
        ^ to raise to power
        a to average

        c to clear
        q to clear

Or, just enter a number >:
```

2. Enter **+** and notice the message, `ADD checked out`, is displayed.

3. In the license server window, the following publisher-specific vendor daemon messages are displayed:

```
11:25:27 (vendor) OUT: "ADD" user1@mysystem
11:25:27 (vendor) IN: "ADD" user1@mysystem
```

   These messages indicate that a license for the feature `ADD` has successfully been checked out and checked back in as a result of using the **+** operator.

4. Enter **q** at the EZCalculator prompt to exit `ezcalc`.

5. Stop the license server. To do so, from the `<platform_dir>` directory, enter the following:

```
lmdown <vendor>_test1.lic
```

This concludes the test.

4

Rebuilding the Production Licensing Toolkit

During the first build of your production Toolkit, you created a Toolkit using your own private settings (as described in First Build of the Production Licensing Toolkit). For subsequent toolkit builds, you must use the same settings to ensure compatibility with FlexEnabled applications created with your previous toolkits.

Rebuilding of the production toolkit is required in the following circumstances:

- You purchase a new version of the toolkit—Upgrading to a New Toolkit Version
- You are changing from using license files only to using license files and trusted storage—Rebuilding to Create a Trusted Storage Production Toolkit
- You are making changes to the functionality of the vendor daemon. For information, see customizing the vendor daemon in *Programming Reference for License File–Based Licensing*.
- You are changing from using license keys or the default license to tamper-resistant licenses (TRL). For more information see the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”.
- You require different versions of the toolkit files:
  - Using the C++ Dynamic Run-time Library
  - Complying with Windows Logo Programs for Software

Upgrading to a New Toolkit Version

To build a production Toolkit when you have received a new version of FlexNet Publisher and ensure compatibility with previously created FlexEnabled applications and license files, follow the instructions in the following sections.

Task ***To build a toolkit compatible with components created using previous toolkit versions***

1. If you have new vendor keys, add them to the toolkit; otherwise, use your existing vendor keys.
2. If you have new TRL keys, add them to the toolkit; otherwise, use your existing TRL keys.

***Note** • If you are implementing TRL for the first time, read the information in the Programming Reference for License File–Based Licensing, chapter 9, "License Signature Configurations".*

3. Add your vendor name to the toolkit, so that the toolkit builds your vendor daemon.
4. Copy publisher-specific encryption settings from your saved files, `lm_code.h` and `publisher.xml`.

   Detailed instructions for steps 1 to 4 are provided in the following sections:

   - Editing lm_code.h
   - Editing the makefile
   - Copying publisher.xml
   - Editing Vendor Daemon Preptool Configuration File (Trusted Storage Only)
5. To implement the prepped Trusted Configuration feature, create Trusted Configuration files, and then identify these files in the `preptool` configuration files for the activation utilities. For more information, see the section Defining Trusted Configuration (Trusted Storage Only) in Chapter 1, First Build of the Production Licensing Toolkit.
6. If you have edited `lsvendor.c`, copy customized settings from your old file to `lsvendor.c` in the new version of the toolkit.
7. Build the toolkit as described in Building the Production Licensing Toolkit.

Editing lm_code.h

Task ***To edit lm_code.h in the new toolkit***

1. Open machind/`lm_code.h` in the new toolkit in a text editor.
2. Replace the five `VENDOR_KEY` lines—

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

   —with your production vendor key lines.
3. Replace demo in the VENDOR_NAME line—

   ```
   #define VENDOR_NAME "demo"
   ```

   —with your vendor name, as registered with Flexera.
4. Copy any existing `ENCRYPTION_SEED`s from your older `lm_code.h`. Example seeds are:

   ```
   #define ENCRYPTION_SEED1 0xfa52984a
   #define ENCRYPTION_SEED2 0x382cb531
   #define ENCRYPTION_SEED3 0x84d2cf21
   #define ENCRYPTION_SEED4 0xa35f2c6b
   ```

If you have only `ENCRYPTION_SEED1` and `ENCRYPTION_SEED2`, add `ENCRYPTION_SEED3` and `ENCRYPTION_SEED4`. All four `ENCRYPTION_SEEDs` should be different from one another.

If there are no `ENCRYPTION_SEEDs` in your old `lm_code.h`, do not create them.

5. Copy the `LM_SEEDS` from your saved version of `lm_code.h`. Example seeds are:

```
#define LM_SEED1 0xac235bd8
#define LM_SEED2 0x672fad24
#define LM_SEED3 0xf3a671b2
```

The following steps (Step 6 and Step 7) assume you do not have legacy clients in production that are older than v10.8.6 and which have to be supported by the same updated license server as your new clients.

6. Replace the two `TRL_KEYs` with those that you received from Flexera.

```
#define TRL_KEY1 0x0
#define TRL_KEY2 0x0
```

7. Copy the `LM_STRENGTH` setting from your saved version of `lm_code.h`. If `LM_STRENGTH` does not exist in the older `lm_code.h`, set `LM_STRENGTH` to `LM_STRENGTH_113BIT` as follows:

```
#define LM_STRENGTH LM_STRENGTH_113BIT
```

For information on stronger license signature alternatives, see the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”.

8. Save and close `lm_code.h`.

Editing the makefile

The `<platform_dir>` directory contains two makefiles:

- `makefile`—For use with license file–based licensing only.
- `makefile.act`—For use with trusted storage–based licensing and license file–based licensing.

The following instructions are for both makefiles.

Task ***To edit the makefile***

1. Using a text editor, open `makefile` in the `<platform_dir>` directory.
2. Change the vendor daemon name from `demo` to your own vendor daemon name as registered with Flexera and used as `VENDOR_NAME` in `lm_code.h`. The vendor daemon name is identified either as `VENDORNAME=` or `DAEMON=`.
3. Determine whether you want your FlexEnabled application to support dongles:
   a. To include dongle support, add **`DONGLELIB=liblmgr_dongle.a`** (or **`lmgr_dongle.lib`**) to link to the dongle library.
   b. To omit dongle support, add **`DONGLELIB=liblmgr_dongle_stub.a`** (or **`lmgr_dongle_stub.lib`**) to link to the dongle stub instead.

Copying publisher.xml

The activation settings file `publisher.xml` provides security keys for your products that used trusted storage–based licensing. This file also provides security settings for products that use license file-based licensing (and contain SDTs) but are created with a trusted-storage licensing toolkit.

Note • *FlexNet Publisher provides an additional activation settings file,* `sdt_publisher.xml`*. This file provides security settings for your applications that use SDTs but are built with FlexNet Publisher toolkits that have rights for license file-based licensing only.*

If you have a previously edited version of `publisher.xml`, copy it into the new toolkit version to overwrite the example activation settings file that comes with the toolkit. Save it into the following location:

```
<platform_dir>/ActivationSettings/publisher.xml
```

It is recommended that you keep these encryption seeds secret by embedding this activation settings file in a *publisher preptool*. For instructions on creating a publisher `preptool` see the chapter *Working with the Software Publisher Utilities* in *Programming Reference for Trusted Storage–Based Licensing*.

For information on initial editing of `publisher.xml`, see the section Editing publisher.xml (Trusted Storage Only) in Chapter 1, First Build of the Production Licensing Toolkit.

Editing Vendor Daemon Preptool Configuration File (Trusted Storage Only)

The *preptool configuration file* is used by preptool to create a vendor daemon that can access license rights in trusted storage. This file is used when running the makefile to generate a vendor daemon.

Task ***To edit vendor daemon preptool configuration file***

1. Using a text editor, open:
   - `machind/activation/prep_xml/vendor_daemon.xml` (Windows platforms).
   - `machind/activation/prep_xml/vendor_daemon_unix.xml` (UNIX platforms).
2. Replace the two instances of the string, `demo`, with your vendor name.
3. Save and close the file.

Rebuilding to Create a Trusted Storage Production Toolkit

This section explains the process of rebuilding the production Toolkit when you have first built the license file toolkit and you want to use trusted storage–based licensing.

A trusted storage production toolkit contains:

- All the files required to build a FlexEnabled application that uses license files only.
- All the files required to build a FlexEnabled application that uses license files and trusted storage.
- A version of your vendor daemon that uses license files and trusted storage.
- Example licensed applications that use trusted storage and your publisher specific encryption settings.

Thus the trusted storage production toolkit can be used to FlexEnable products that use license file–based licensing, as well as trusted storage–based licensing.

The following instructions assume that you have already completed one of the following when building the license file production Toolkit:

- Configuring the FlexNet Publisher Licensing Toolkit as described in Chapter 1, First Build of the Production Licensing Toolkit.
- Upgrading to a New Toolkit Version as described in this chapter.

Task ***To rebuild the toolkit as a trusted storage toolkit***

1. Edit the vendor daemon name in `makefile.act`—Editing makefile.act
2. Add security key seeds to the activation settings file—Editing publisher.xml
3. Edit the vendor daemon name in the vendor daemon preptool configuration file—Editing Vendor Daemon Preptool Configuration File
4. Build the toolkit—Building the Production Licensing Toolkit

Editing makefile.act

Task ***To edit makefile.act***

1. Using a text editor, open `makefile.act` in the `<platform_dir>` directory.
2. Change the vendor daemon name from `demo` to your own vendor daemon name as registered with Flexera and used as `VENDOR_NAME` in `lm_code.h`. The vendor daemon name is identified either as `VENDORNAME=` or `DAEMON=`.

Editing publisher.xml

These activation settings files provides security keys for your products that use trusted-storage licensing or for products that use license file-based licensing (and contain SDTs) but are created with a trusted-storage licensing toolkit.

Provide a new set of unique security keys for this release by editing the encryption seeds in the appropriate file with your own set of new values. See the section Editing publisher.xml (Trusted Storage Only) in the chapter First Build of the Production Licensing Toolkit.

It is recommended that you keep these encryption seeds secret by embedding this activation settings file in a *publisher* `preptool`. For instructions on creating a publisher `preptool`, see the chapter *Working with the Software Publisher Utilities* in *Programming Reference for Trusted Storage–Based Licensing*.

Editing Vendor Daemon Preptool Configuration File

The *preptool configuration file* is used by preptool to create a vendor daemon that can access license rights in trusted storage. This file is used when running the makefile to create your vendor daemon and an associated activation library. The activation library allows the vendor daemon to access your publisher specific trusted storage.

Task ***To edit vendor daemon preptool configuration file***

1. Using a text editor, open:
   - machind`/activation/prep_xml/vendor_daemon.xml` (Windows platforms).
   - `machind/activation/prep_xml/vendor_daemon_unix.xml` (UNIX platforms).
2. Replace the two instances of the string, `demo`, with your vendor name.
3. Save and close the file.

Using a TRL-Only Version of the Vendor Daemon

When all your licensed products use tamper-resistant licensing (TRL), you can create a version of your vendor daemon that supports only TRL. This provides increased security for your licensing code. The TRL-only version of the vendor daemon cannot be used with any licensed products that use other license signatures. The toolkit makefiles are linked to `lmgr_trl.lib` (Windows) or `liblmgr_trl.a`/`liblmgr_pic_trl.a` (UNIX) by default.

To use a non-TRL version, link the vendor daemon to `lmgr.lib` (Windows) or `liblmgr.a`/`liblmgr_pic.a` (UNIX) by adding it in the linker flag during compilation of the makefile.

Caution • *A TRL-only version of the vendor daemon does not support products that use any other license signatures. Ensure that it is used only on systems where all of your licensed products use TRL. For more information refer to the section Tamper-Resistant Licensing: Using TRL-Only Libraries.*

Using the C++ Dynamic Run-time Library

You can build the production toolkit so that it uses the C++ dynamic run-time library, `msvcrt.lib`. This is not recommended because using the dynamic library is less secure. A hacker can replace a DLL on the system to find out the library and system calls used by FlexNet Licensing. For this reason the default build of the production Toolkit builds the toolkit using the C++ static run-time library.

However for internal testing of a license you can create a version of the Toolkit that uses the C++ dynamic run-time library. You should not release any test versions of FlexEnabled products that use the C++ dynamic run-time library.

To create a Toolkit that uses the C++ dynamic run-time library, add the `MD=1` option to your make command. For example on Windows:

```
nmake MD=1 -f makefile
```

Part 2

For Trusted Storage–Based Licensing

This part of the Development Environment Guide includes the following chapters:

- Building Your FlexEnabled Application for Trusted Storage–Based Licensing
- Distributing the FlexEnabled Application for Trusted Storage–Based Licensing

5

Building Your FlexEnabled Application for Trusted Storage–Based Licensing

This chapter gives the steps necessary to build your FlexEnabled application.

The procedures in this chapter assume you have built and tested the production FlexNet Publisher Licensing Toolkit with your publisher-specific information as detailed in First Build of the Production Licensing Toolkit or Rebuilding the Production Licensing Toolkit. Then you have added licensing code to your application as described in the following documents:

- Flexible API functions—described in *Programming Reference for License File–Based Licensing* and *C/C++ Function Reference*
- Activation API and Activation Borrow API functions—described in *Programming Reference for Trusted Storage–Based Licensing* and *C/C++ Function Reference*

Before building your application, read the following:

- Identifying Security Functionality in Your Toolkit
- Security Considerations

The basic steps in building your application are:

1. Compiling and linking the application.

   For further instructions, see Building UNIX Applications or Building Windows Applications as appropriate to your platform.

2. Prepping the Application.
3. Optionally, Signing the Application.

After building your application test it as described in Testing the Application.

Identifying Security Functionality in Your Toolkit

Before building your application, you need to determine the level of security functionality your toolkit supports. Consult the *FlexNet Publisher Release Notes* for information about your platform's security functionality. For the instructions in this chapter, the following nomenclature is used:

- *notr* platforms—Platforms with security functionality designated as *notr: tamper-resistant applications not supported* in the *FlexNet Publisher Release Notes*
- *standard* platforms—Platforms with security functionality designated as *standard: tamper-resistant applications supported* in the *FlexNet Publisher Release Notes*

Security Considerations

Generating a New Version of lm_new for Each Build

Use a newly generated version of the `lm_new` object, `lm_new.o` (UNIX platforms) or `lm_new.obj` (Windows platforms), each time you link your application: this makes the product more secure. There are two ways to generate the `lm_new` object:

- Using the FlexNet Publisher toolkit `makefile.act`, change to the `<platform_dir>` directory, and enter the following commands:

  ```
  rm lm_new.o
  make lm_new.o
  ```

- Using the `lmnewgen` utility, change to the `<platform_dir>` directory, and enter the following commands:

  ```
  lmnewgen vendor_daemon_name -o lm_new.c
  cc -c lm_new -c -o lm_new.o
  ```

It is suggested that each project be provided with a different version of the `lm_new` object file. There are two approaches to providing this file:

- A central authority can generate and distribute the object file.
- `lmnewgen` and the vendor daemon name can be distributed to each project so the object file can be generated as needed. This approach is less secure because `lmnewgen` leaves the central authority.

Tamper-Resistant Licensing: TRL-Only Libraries

The TRL-only libraries provide increased security when using TRL licenses, because they allow only TRL licenses and not a combination of TRL and non-TRL licenses, as is the case with `lmgr.lib` and `liblmgr.a`.

If you are using Tamper-Resistant Licensing (TRL) for your product and you do not need backward compatibility with non-TRL licenses, use one of the following TRL-only libraries:

- **Windows**—`lmgr_trl.lib`
- **UNIX**—`liblmgr_trl.a` or `libmgr_pic_trl.a`

If you are using TRL-only libraries, you cannot use the following attribute settings (that are used to provide backward compatibility with non-TRL licenses) within FlexEnabled applications built with these libraries:

- `LM_A_SIGN_LEVEL` set to zero
- `LM_A_STRENGTH_OVERRIDE` set to `LM_STRENGTH_DEFAULT`

When upgrading to TRL, you must reissue license rights. See the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”, for more information.

sdtconfig.h

If Secure Data Types (SDTs) have been added to your application code, you need to define your publisher-specific SDT configuration information by editing settings in `sdtconfig.h`. Change the `SDT_CONFIG_SEED` entries when building an application for a product release. For information on the additional security provided by SDTs and `sdtconfig.h`, see the sdtconfig.h section in the Chapter 9, Implementing Secure Data Types (SDTs).

Note • *SDTs are supported on standard platforms only. See Identifying Security Functionality in Your Toolkit to determine the level of support on your platform.*

Building UNIX Applications

A standard FlexEnabled application for UNIX platforms is built as a multithreaded executable. This is to accommodate the default automatic heartbeat scheme, which requires a multithreaded environment. Two build scenarios are presented based on whether you are implementing Secure Data Types in your application:

- Building the Application Without Secure Data Types
- Building the Application With Secure Data Types

Important • *For Mac applications, use OS X 10.5, 10.6, or 10.7 as appropriate to develop and build your FlexEnabled application. If you are building applications that run on 64-bit and 32-bit Mac platforms and they use FlexNet ID dongles, see Building Mac Applications With FlexNet ID Dongle Support.*

Note • *To build a FlexEnabled application that must be Linux Standard Base (LSB) certified, you must follow the guidelines of the Linux Foundation. For more information about Linux Standard Base certification, see the Linux Foundation's website: http://www.linuxfoundation.org. For details about LSB support in FlexNet Publisher, see your FlexNet Publisher Release Notes.*

Build Considerations for Shared Objects

The information in this section is applicable to FlexEnabled products that include one or more shared-library components.

Note • *All FlexEnabled products where the application is prepped contain a shared-library component—the activation library.*

- On platforms where position independent code (PIC) is not the default build model, you must build as PIC and link with PIC libraries. These are distinguished with the `_pic` addition to the directory containing static libraries or static library name itself. This addition indicates the PIC variant.
- It is recommended that an exports file be used when linking a shared object so that only the public interface of the shared object is exported. If this is not done, all global symbols will be exported, which can cause problems with duplicate symbols in applications where there are multiple protected components.

- Both the method used in a link command to produce a shared object and the compile and linker command-line flags for the exports file are different for each platform and build tool. Refer to the relevant compiler and linker documentation for details.

Building the Application Without Secure Data Types

Follow the instructions in this section if you do not have Secure Data Types (SDTs) implemented. For implementations that include SDTs, see Building the Application With Secure Data Types.

These instructions are written for applications developed in the C programming language. When reviewing these instructions, refer to the build rules for `ezcalc` in `<platform_dir>/makefile.act` for platform-specific examples. If your product includes shared objects, also review the section Build Considerations for Shared Objects.

Note • *Set the environment variable* `CC` *before proceeding. (Use a command similar to* `setenv CC gcc`*.) Consult the "Detailed Platform Information" section of the FlexNet Publisher Release Notes for information about the compiler used for your platform.*

To compile your application, issue a command similar to the following:

```
gcc -Imachind -I. -Imachind/activation/include $(CFLAGS) program.c
```

Table 5-1 • Command Description

Command	Description
machind	The machine-independent directory identified in Identifying the Toolkit Directories.
`$(CFLAGS)`	Additional `gcc` flags required for your platform. Refer to the build rules for `ezcalc` in `<platform_dir>/makefile.act` for the flags specific to your platform.
program`.c`	Your FlexEnabled application code.

When you have successfully compiled your application, link with a command similar to the following:

For standard platforms:

```
gcc -o program program.o lm_new.o $(OBJS) $(AALIB) $(CLIENTLIB) \
    $(XTRALIB) $(THREADLIB) $(ACT_LIBRARY)
```

For `notr` platforms:

```
gcc -o program program.o lm_new.o $(OBJS) $(AALIB) \
    program_FNPLOAD.o \
    $(CLIENTLIB) \
```

```
$(XTRALIB) $(THREADLIB) $(ACT_LIBRARY)
```

Table 5-2 • Command Descriptions

Command	Description
`program`	Your FlexEnabled application.
`lm_new.o`	A security component generated each time you build your production Toolkit. It can be generated manually. See Security Considerations.
`program``_FNPLOAD.o`	(For `notr` platforms only) The activation library component.
`$(OBJS)`	The list of the objects for your application.

`$(AALIB)`
`$(CLIENTLIB)`
`$(XTRALIB)`
`$(THREADLIB)`
`$(ACT_LIBRARY)`
	FlexNet Publisher and platform-specific libraries. Refer to these macros in `<platform_dir>/makefile.act` for the list specific to your platform.

Ensure that multithreaded system libraries are specified on the command line. Refer to your system's documentation for details on linking with multithreaded libraries. See the *Programming Reference for License File–Based Licensing* for more details on implementing heartbeats. For standard platforms, continue with Prepping the Application.

Building the Application With Secure Data Types

Note • *Secure Data Types are supported on standard platforms only. Consult Identifying Security Functionality in Your Toolkit to determine the level of support on your platform.*

Follow the build instructions in this section if you have implemented Secure Data Types (SDTs) into your code. This section demonstrates compiling an application written in C++, the recommended implementation language when you incorporate SDTs. When reviewing the instructions in this section, refer to the build rules for `ezcalcsdtplus` in `<platform_dir>\makefile.act` for examples specific to your platform. If your product includes shared objects, also review the section Build Considerations for Shared Objects.

Note • *Set the environment variables* `CC` *to* `gcc` *or* `LD` *to* `g++` *before proceeding. Use commands similar to* `setenv CC gcc` *or* `setenv LD g++`*. The syntax might vary with the particular command line shell in use. (Consult the "Detailed Platform Information" section of the FlexNet Publisher Release Notes for information about the compiler used for your platform.)*

Use commands similar to the following to compile `sdtdefs.c`, `sdttype.cpp`, and your application code:

```
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
    $(CFLAGS) sdtdefs.c
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
    $(CFLAGS) sdttype.cpp
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
```

```
$(CFLAGS) program.cpp
```

Table 5-3 • Command Descriptions

Command	Description
`machind`	The machine-independent directory identified in Identifying the Toolkit Directories.
`$(CFLAGS)`	Additional gcc flags required for your platform: Refer to the build rules for `ezcalc` in `<platform_dir>/makefile.act` for the flags specific to your platform.
`sdtdefs`	The name of your SDT feature implementation file. See Chapter 9, Implementing Secure Data Types (SDTs), for more information.
`sdttype.cpp`	Component required when inserting SDTs in C++ code.
`program``.cpp`	Your FlexEnabled application code.

When you have successfully compiled your application, link with a command similar to the following:

```
g++ -o program program.o lm_new.o sdtdefs.o sdttype.o $(OBJS) \
    $(AALIB) $(CLIENTLIB) $(XTRALIB) $(THREADLIB) $(ACT_LIBRARY)
```

Table 5-4 • Command Descriptions

Command	Description
`program`	Your FlexEnabled application.
`lm_new.o`	A security component generated each time you build your production Toolkit. It can be generated manually. See Security Considerations.
`sdtdefs.o` `sdttype.`*`o`	SDT object files.
`$(OBJS)`	The list of the objects for your application.

`$(AALIB)`
`$(CLIENTLIB)`
`$(XTRALIB)`
`$(THREADLIB)`
`$(ACT_LIBRARY)`
	FlexNet Publisher and platform-specific libraries. Refer to these macros in `<platform_dir>/makefile.act` for the list specific to your platform.

To accommodate the automatic heartbeat feature, specify the multithreaded system libraries on the command line. Refer to your system's documentation for details on linking with multithreaded libraries. See the *Programming Reference for License File–Based Licensing* for more details on implementing heartbeats.

Continue with Prepping the Application.

Building Mac Applications With FlexNet ID Dongle Support

Use these procedures when building Mac applications that support FlexID dongles.

Link to Appropriate Library

Like Linux applications, Mac applications must link to either the dongle library or the stub library, even if the application does not include dongle support. Make sure your application links to one of these libraries.

Task ***To include or omit dongle support***

Perform one of the following:

- Link to `liblmgr_dongle.a` to include dongle support.
- Link to `liblmgr_dongle_stub.a` to omit dongle support.

Create 32-Bit Applications

FlexNet ID dongles are not supported for 64-bit applications running on Mac 64-bit platforms. Products that use FlexNet ID dongles and that might be run on 64-bit platforms should be released as 32-bit applications only.

Task ***To create a licensed application binary that contains only 32-bit versions***

Build the licensing toolkit in the appropriate Mac development environment. (See the latest *FlexNet Publisher Licensing Toolkit Release Notes* for up-to-date environment requirements for your OS X Mac platform.) This toolkit can be used to build products for all the architectures that the toolkit supports including 64-bit. Ensure that your product build instructions build only 32-bit applications (select only 32-bit architectures within the SDK you are using).

Building Windows Applications

To build a FlexEnabled application on a Windows platform, you *must* use a supported Microsoft Visual Studio version, as listed in the latest *FlexNet Publisher Licensing Toolkit Release Notes*.

Two build scenarios are presented based on whether you are implementing Secure Data Types in your application:

- Building the Application Without Secure Data Types
- Building the Application With Secure Data Types

Note • *Make sure that you have the command-line interface of the Microsoft Visual C++ development environment correctly configured. Consult the Visual C++ documentation for details about building from the command line.*

These sections give instructions for the recommended options to use when building a FlexEnabled application that uses trusted storage. For information on alternative options, which are also available when using trusted storage, see Building Windows FlexEnabled Applications in Chapter 7.

Building the Application Without Secure Data Types

Follow the instructions in this section if you do not have Secure Data Types (SDTs) implemented. These instructions refer to the sample application `ezcalc.exe` to demonstrate the build process. When reviewing these instructions, refer to the build rules for `ezcalc.exe` in `<platform_dir>/makefile.act` for a platform-specific example. (For implementations that include SDTs, use the instructions in Building the Application With Secure Data Types instead.)

To compile your application, issue a command similar to the following:

```
c1 /nologo /c /MT /01 /Imachind /I. /Imachind\activation\include ..\examples\ezcalc\ezcalc.c
```

Where

Table 5-5 • Command Descriptions

Command	Description
machind	The machine-independent directory identified in Identifying the Toolkit Directories.
ezcalc.c	The code file for the sample application `ezcalc.exe`. Replace this value with the path and name of the file containing your FlexEnabled application code.

FlexNet Publisher licensing requires a multithreaded run-time environment. The `cl` compiler `/MT` switch indicates the static multithreaded C library and the `/MD` switch indicates the multithreaded C DLL. One or the other must be used when compiling. Include `/MT` (or `/MD`) on the `cl` command line.

When you have successfully compiled your application, link with a command similar to the following:

```
LINK /nologo /NODEFAULTLIB /OPT:NOREF /out:ezcalc.exe ezcalc.obj \
    $(LMNEW_OBJ_CLIENT) $(CLIENTLIB) $(ACTLIB) $(XTRALIB)
```

Where:

Table 5-6 • Command Descriptions

Command	Description
ezcalc.exe	The file name of the sample application *ezcalc*.exe. Replace this value with the file name of your FlexEnabled application.
$(LMNEW_OBJ_CLIENT)	The `lm_new.obj` file. This is a security component generated each time you build your production Toolkit. It can be generated manually.
$(CLIENTLIB)	FlexNet Publisher client libraries: • `libcrvs.lib` • `libsb.lib` • `lmgr_trl.lib` ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*

Table 5-6 • Command Descriptions

Command	Description
`$(ACTLIB)`	FlexNet Publisher activation libraries: ● `activation\lib\libact.lib` ● `libFNPload.lib` for standard toolkits and `libNFNPload_notr.lib` for notr toolkits ● `FNP_ACT_Installer.lib`
`$(XTRALIB)`	Additional dongle and Windows system libraries: ● `lmgr_dongle.lib` (for dongle usage) or `lmgr_dongle_stub.lib` (no dongle usage) ● Windows system libraries (view the contents of the `platform_dir>/makefile.act` file for a complete list)
`$(XTRALIB1)`	Additional system libraries: ● `Ws2_32.lib` - A system library for socket communication on Windows platform for 32bit architecture. ● `Psapi.lib` - for enhanced license server diagnostics in debug log. See Table 1-4 for the complete list of system libraries.
`$(LMSTDREDIR_LIB)`	FlexNet Publisher client library: `libredir_std.dir`

Continue with Prepping the Application.

Building the Application With Secure Data Types

Follow the build instructions in this section if you have implemented Secure Data Types (SDTs) into your code. These instructions refer to the sample application `ezcalcsdtplus.exe` to demonstrate the build process of an application written in C++, the recommended implementation language when you incorporate SDTs. When reviewing the instructions, refer to the build rules for `ezcalcsdtplus.exe` in `<platform_dir>\makefile.act` for examples specific to your platform.

Use commands similar to the following to compile `sdttype.cpp`, your SDT implementation file, and your application code:

```
cl /DWIN32 $(CFLAGS) /Zm1000 /Imachind /I. /Imachind\activation\include \
    /Imachind\sdt ..machind\sdt\sdttype.cpp
cl /DWIN32 $(CFLAGS) /Imachind /I. /Imachind\activation\include \
    /Imachind\sdt ..\examples\ezcalc\sdtdefs.c
cl $(CFLAGS) /EHsc /Imachind /Iplatform /Imachind\activation\include \
    /Imachind/sdt ..\examples\ezcalc\ezcalcsdtplus.cpp
```

Where

Table 5-7 • Command Descriptions

Command	Description
`machind`	The machine independent directory identified in Identifying the Toolkit Directories.
`sdttype.cpp`	Component required when inserting SDTs in C++ code.
`sdtdefs`	The SDT implementation file for the sample application `ezcalcsdtplus.exe`. Replace this value with the path and name of your SDT implementation file. See Chapter 9, Implementing Secure Data Types (SDTs), for more information about the implementation file.
`$(CFLAGS)`	`/nologo /c /MT /01`
`ezcalcsdtplus``.cpp`	The code file for the sample application `ezcalcsdtplus.exe`. Replace this value with the path and name of the file containing your FlexEnabled application code.

When you have successfully compiled your application, link with a command similar to the following:

```
LINK /nologo /NODEFAULTLIB /OPT:NOREF /out:ezcalcstdplus.exe ezcalcstdplus.obj \
    sdttype.obj sdtdefs.obj lm_new.obj $(LMNEW_OBJ_CLIENT) \
    $(CLIENTLIB) $(DONGLELIB) $(ACTLIB) $(CPPXTRALIB)
```

The following table describes the command syntax:

Table 5-8 • Command Descriptions

Command	Description
`ezcalcstdplus``.exe`	The file name of the sample application `ezcalcsdtplus.exe`, which uses SDTs. Replace this value with the file name of your FlexEnabled application containing SDTs.
`sdtdefs``.obj` `sdttype.obj`	SDT object files.
`$(LMNEW_OBJ_CLIENT)`	The `lm_new.obj` file. This is a security component generated each time you build your production Toolkit. It can be generated manually. See Security Considerations.
`$(CLIENTLIB)`	FlexNet Publisher client libraries: • `libcrvs.lib` • `libsb.lib` • `lmgr_trl.lib` ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*

Table 5-8 • Command Descriptions

Command	Description
`$(DONGLELIB)`	Dongle libraries: ● `lmgr_dongle.lib` (for dongle usage) ● `lmgr_dongle_stub.lib` (no dongle usage)
`$(ACTLIB)`	FlexNet Publisher activation libraries: ● `activation\lib\libact.lib` ● `libFNPload.lib` for standard toolkits and `libNFNPload_notr.lib` for notr toolkits ● `FNP_ACT_Installer.lib`
`$(CPPXTRALIB)`	View the contents of the `<platform_dir>/makefile.act` file for a complete list of required Windows system libraries.
`$(LMSTDREDIR_LIB)`	FlexNet Publisher client library: `libredir_std.lib`

Continue with Prepping the Application.

Prepping the Application

Prepping the application involves the following:

- Creating a publisher- and product-specific activation library component
- Injecting the FlexEnabled application binary with publisher-specific public and private encryption keys in order to enable its communication with the activation library component

For this step you need your application binary that you just built and an application-specific preptool configuration file. Complete instructions for creating the configuration file and prepping the application are found in the *Programming Reference for Trusted Storage–Based Licensing* section which describes the `preptool`.

Note • *The standard utility provided with FlexNet Publisher is called `preptool`. However, if a publisher-specific preptool is provided for your use, it may have a different name.*

After prepping, there are two components that comprise the application: the executable and the application-specific activation library component.

For Windows applications, these include:

- `program.exe`—The executable
- `program_libFNP.dll`—The application-specific activation library component

For UNIX applications, these include:

- `program`—The executable

- `program_libFNP.so`—The application-specific activation library component

The executable locates the library component as per the operating system default behavior. This may be modified by configuration file settings used by the preptool (see *Programming Reference for Trusted Storage–Based Licensing*). If the location (or relative location) of the library component is not known at build-time, the full path to the application-specific activation library component may be overridden at run-time by setting the environment variable "`FLEXNETPUBLISHER_RUN_TIME_PATH`".

Signing the Application

Mac OS X and Windows support the digital signing of executable code as well as other files. This enables both the operating system and the user to know the authenticity of files that are installed on the system.

Files provided within the version of the production toolkit that is used on Mac OS X or Windows are signed unless they are recompiled during use. Files that you create when you are creating a licensed product, including the licensed application, you should sign yourself. The following Table 5-9 for Mac OS X platforms lists files and indicates which are provided as signed files and which you need to sign. See Table 1-1, FlexNet Publisher Licensing Toolkit files, on page 27, for information relevant for Microsoft platforms.

For detailed information on signing, read the relevant manufacturer's documentation.

You must prep your licensed application using `preptool`, as described in the previous section, before you sign your licensed application. You should also sign the application-specific activation library component created by `preptool`.

Table 5-9 • Code Signing on Mac OS X

File Name	Description	Code Signing
	FlexEnabled application	Recommended
	Vendor daemon	Recommended
FnpCommsSoap.dylib	SOAP communications component for transactions with FlexNet Operations	Signed
lmadmin	Web-based license server manager	Signed
lmgrd	Command-line license server manager	Signed
lmrand1	Random number generator used to generate `LM_SEEDS`	Signed
lmstrip	Utility used to obfuscate and add security to FlexEnabled applications	Signed
lmutil	Utilities used for license server management and administration	Signed

Testing the Application

This section describes how to test your FlexEnabled application to ensure you built it to successfully access the license rights it needs. This testing procedure involves:

- Creating the ASR—You create a test activation specification record (ASR) using the components of your production toolkit. This ASR contains license rights for features that your application needs.
- Activating License Rights—You make the license rights available to the application by loading the test ASR into the application's trusted storage.
- Checking Out a License—You run your FlexEnabled application to test that features are successfully checked out.

All steps take place from the `<platform_dir>` directory.

Creating the ASR

In the following example, an activation specification record is created with an expiration date of tomorrow.

Task ***To create an activation specification record***

1. Create a license file, `template.lic`, with appropriate template `INCREMENT` lines for your application. As an example, `INCREMENT` lines for the `ezcalc` application would look something like this:

   ```
   SERVER this_host INTERNET=0.0.0.0
   VENDOR myvendor
   INCREMENT ADD myvendor 1.0 31-dec-2020 1 SIGN=
   INCREMENT SUBTRACT myvendor 1.0 31-dec-2020 1 SIGN=
   INCREMENT MULTIPLY myvendor 1.0 31-dec-2020 1 SIGN=
   INCREMENT DIVIDE myvendor 1.0 31-dec-2020 1 SIGN=
   ```

   In this example, `myvendor` is your vendor name, and the `SIGN=` values are left blank.

2. Create the ASR using the `asrgen` utility. From a command line, type the following:

   ```
   asrgen -c template.lic -t 1 -i publisher/vendorTrustedConfigOut.xml \
       -o test.asr
   ```

   You receive the following confirmation:

   ```
   ASR file test.asr generated
   ```

   This signs the license certificates in `template.lic` and inserts them in the fulfillment record in `test.asr`.

   Note • *When you create an ASR, best practice is always to include the* `-t` *switch to specify the trial ID.*

3. Open `test.asr` with a text editor and modify it as appropriate to your business model. The following table shows what would be modified for the `ezcalc` application:

   Table 5-10 • Test ASR Fulfillment Record Settings

Element in `test.asr`	Replace Value with	Notes

`<Key>`
`PUBLISHER_DEFINED_DURATION`
`</Key>`
`<Value>86400</Value>`
	The duration of the license in seconds. 86400 seconds represents one day.	This value is used to set the duration of a fulfillment record when it is written to trusted storage.

Table 5-10 • Test ASR Fulfillment Record Settings

Element in `test.asr`	Replace Value with	Notes

`<FulfillmentID>`
`FID-1120683846`
`</FulfillmentID>`
	FIL-test	A test Fulfillment ID for the activation transaction

`<Key>ProductId</Key>` `<Value>EZCALC-A</Value>`	PROD-test	A test Product ID

`<Key>EntitlementId</Key>`
`<Value>ENTL-EZCALC-AS1`
`</Value>`
	ENTL-test	A test Entitlement ID

4. Save and close `test.asr`.
5. Sign `test.asr` using the `xmlsign` utility. Type the following:

```
publisher/XMLSign \
    -a ActivationSettings/publisher.xml \
    -i test.asr \
    -o test_signed.asr
```

You receive the confirmation message `test_signed.asr signed`.

Activating License Rights

This section describes the steps necessary to activate the license rights you just created in `test_signed.asr` locally in the application machine's trusted storage, making them available to your FlexEnabled application.

If a trusted storage file for your vendor name does not exist, this procedure creates one; if one exists, it is modified. With this in mind, you can perform these steps either locally on the same machine where your production toolkit resides, or on a separate test machine. The following activation components need to reside on the machine on which the license rights are activated:

- The application activation utility—`appcomptranutil.exe` (Windows platforms) or `appcomptranutil` (UNIX platforms); and its activation library component, one of:
  - `appcomptranutil_libFNP.dll` (Windows platforms)
  - `appcomptranutil_libFNP.so` (Standard UNIX platforms)
  - `appcomptranutil_libFNP_notr.so` (notr UNIX platforms)
- The signed test ASR created in the last section—`test_signed.asr`

If you test on a separate, clean test machine for which trusted storage has never been initialized, these additional components need to be installed on the clean machine. They must reside in the same directory with the activation utility and its components.

- For Windows platforms
  - `<platform_dir>\FNP_Act_Installer.dll`

  This component must have the name `FNP_Act_Installer.dll`.
- For OS X platforms and Linux platforms

  - `<platform_dir>/publisher/FNPLicensingService/FNPLicensingService`
  - `<platform_dir>/publisher/install_fnp.sh`
- All other UNIX platforms
  - `<platform_dir>/publisher/install_fnp.sh`

Initializing the Trusted Storage Directory and Installing the FlexNet Licensing Service

Follow the instructions in this section only if you are testing your application on a separate, clean test machine for which trusted storage has never been initialized. Otherwise, proceed to Adding License Rights.

UNIX Hosts

Before trusted storage is accessed, its directory structure must be initialized. In addition, on OS X and Linux platforms, the FlexNet Licensing Service—an administrative proxy that runs on behalf of the user for certain types of interactions to and from trusted storage—needs to be installed. The steps below initialize the trusted storage directory and, on OS X and Linux platforms, install the FlexNet Licensing Service.

Important • *You must either be logged in as root or have* `sudo` *privileges to perform these steps.*

Initialize the trusted storage directory and install the FlexNet Licensing Service by running the following script:

```
sudo sh install_fnp.sh
```

At the end of the process, you should receive the following confirmation message:

Configuration completed successfully.

Important • *On OS X and other UNIX platforms, the trusted storage directory is typically* `/usr/local/share/macrovision/storage/FlexNet/`*. FlexNet Publisher does not support custom locations or symbolic links for the trusted storage directory.*

Windows Hosts

Before trusted storage is created or accessed, the FlexNet Licensing Service must be installed. Run the example application, `installanchorservice.exe`, by entering the following:

```
installanchorservice Demo ezcalc
```

This installs the FlexNet Licensing Service and registers that it is being used by the product EZcalc supplied by the publisher `Demo`.

Note • *You must have administrator permissions to install the FlexNet Licensing Service on Windows.*

Adding License Rights

Activate a license in trusted storage using the signed test ASR, `test_signed.asr`. From the `<platform_dir>` directory, enter the following:

```
appcomptranutil -local test_signed.asr
```

You see the following confirmation messages:

```
Local activation, ASR test_signed.asr
Local activation successful
```

Viewing Trusted Storage

View the contents of trusted storage to see exactly which licenses are activated. From the `<platform_dir>` directory, enter the following:

```
appcomptranutil -viewlong
```

Output similar to the following is displayed:

```
-------------------------------------------------------------------
       Fulfillment Id : FIL-test
     Fulfillment Type : 1 Trial
          Trust Flags : 0x7 Fully Trusted
               Status : ENABLED
       Entitlement ID : ENTL-test
           Product ID : PROD-test
     Expiration date  : 12-aug-2017
      Feature line(s) :
INCREMENT ADD demo 1.0 31-dec-2020 1 SIGN="0080 1DDF C055 6D9E 0476 \
        F4FB A7AF 3200 681E 8C6F C2D9 81DB AF50 57B5 347E"
INCREMENT SUBTRACT demo 1.0 31-dec-2020 1 SIGN="00A6 4216 51AE 7A8B \
        19DE 7461 0478 5100 BA0C 4D70 6392 9F79 BF37 7080 9A6D"
INCREMENT MULTIPLY demo 1.0 31-dec-2020 1 SIGN="00F7 12C4 21F4 FDA2 \
        8B11 7C15 295C EA00 9080 6CBA 2CEB C8C2 F256 1C20 9930"
INCREMENT DIVIDE demo 1.0 31-dec-2020 1 SIGN="009B 56B5 FF2C EBEC \
        BB36 084B 8AA4 D900 B7A1 520C D5AE FCF2 1717 3A7C 36D8"
-------------------------------------------------------------------
```

Note that the value of the `Expiration date` field corresponds to the value set with the `PUBLISHER_DEFINED_DURATION` key element in the ASR. See Table 5-10 for details.

Checking Out a License

At this point, license rights have been activated and are available to your FlexEnabled application. The following application components need to reside on the machine on which the license rights are activated and must reside together in the same directory:

- The application binary—`program.exe` (Windows platforms) or `program` (UNIX platforms)
- The activation library component, one of
  - `program_libFNP.dll` (Windows platforms)
  - `program_libFNP.so` (Standard UNIX platforms)

  - `program_libFNP_notr.so` (notr UNIX platforms)

- The FlexNet Licensing Service installer (Windows platforms)
  - `<platform_dir>\FNP_Act_Installer.dll`

  This component must have the name `FNP_Act_Installer.dll`.

Start your application and perform a task that checks out a feature. Confirm that the expected feature is checked out.

6

Distributing the FlexEnabled Application for Trusted Storage–Based Licensing

This chapter is for publishers who provide trusted storage–based licensing for their FlexEnabled applications. It contains the following sections:

- FlexNet Publisher Components Your End User Requires
- Installer Considerations
- What to Tell Your End Users

To ensure that your end user can use your FlexEnabled product easily and successfully, use the information in this chapter as a guideline for distributing your FlexEnabled application.

Note • *This chapter provides information pertinent to trusted storage–based licensing. If you provide license file–based licensing, see Chapter 8, Distributing the FlexEnabled Application for License File–Based Licensing.*

Identifying Security Functionality in Your Toolkit

Before deploying your FlexEnabled application, you need to determine the level of security functionality your toolkit supports. Consult the deployed platform section of *Release Notes* for your specific platform to find out its security functionality. For the instructions in this chapter, the following nomenclature is used:

- **notr platforms**—Platforms with security functionality designated as *tamper-resistant applications not supported* in the *Release Notes*.
- **standard platforms**—Platforms with security functionality designated as *tamper-resistant applications supported* in the *Release Notes*.

FlexNet Publisher Components Your End User Requires

When you ship your FlexEnabled application, you must include FlexNet Licensing Service components, application components, and license rights components in your software installation kit. This section discusses only those components necessary for trusted storage–based licensing. See Chapter 8, Distributing the FlexEnabled Application for License File–Based Licensing, for information about other FlexNet Publisher components.

FlexNet Licensing Service

The FlexNet Licensing Service is an administrative proxy that runs on behalf of the user to perform system tasks where elevated privilege is required.

For trusted storage–based licensing, the FlexNet Licensing Service must be installed to enable the FlexEnabled application, the activation utilities, and the vendor daemon to operate correctly.

In addition, the FlexNet Licensing Service is required to enable the FlexEnabled application and the vendor daemon to access virtualization data (on Windows and Linux platforms only). The FlexNet Licensing Service regularly polls the system to update virtualization data. Therefore, the FlexNet Licensing Service runs continuously. The virtualization poll interval is not configurable.

Table 6-1 lists the location of the FlexNet Licensing Service component for each supported platform.

Table 6-1 • FlexNet Licensing Service Components

Platform	FlexNet Licensing Service Component
Windows	`<platform_dir>\FNP_Act_Installer.dll` `<platform_dir>\FNP_Act_Installer.lib`
OS X	`<platform_dir>/publisher/FNPLicensingService/FNPLicensingService`
Linux	`<platform_dir>/publisher/FNPLicensingService/FNPLicensingService`
Solaris	FlexNet Licensing Service is not required on this platform.

Two versions of the Windows component are available: 32-bit and 64-bit. You must ensure that the correct version of `FNP_Act_Installer.dll` is shipped to your end user: the 32-bit version for 32-bit platforms and the 64-bit version on 64-bit platforms. Each version has the same name, `FNP_Act_Installer.dll`. The 32-bit version is provided in toolkits for 32-bit Windows platforms and the 64-bit version is provided in toolkits for 64-bit platforms. Always ship the copy of `FNP_Act_Installer.dll` from the toolkit you used to create your FlexEnabled application.

To determine the version of `FNP_Act_Installer.dll`, examine the properties of the file (**Properties|Version|Product Name**). The Product Name field indicates whether it is a 32-bit or 64-bit binary.

Application Components

Application components include the following:

- **Application binary**—*program*.exe (Windows platforms) or *program* (UNIX platforms)
- **Its corresponding activation library component**—One of the following:
  - *program*_libFNP.dll (Windows platforms)
  - *program*_libFNP.so (Standard UNIX (tr) platforms)
  - *program*_client_libFNP_notr.so (notr UNIX platforms)

License Rights Components

License rights components provide the means by which license rights are delivered, installed, and activated at your end user's enterprise. Depending on your license model and license lifecycle, you ship components used locally on the FlexEnabled client machine (where the end user's FlexEnabled application resides) or used both on one or more license server machines and FlexEnabled client machines.

FlexEnabled Client Components

Ship these components for installation on the FlexEnabled client machine.

For Activations From an Activation Server

If you support activation on the FlexEnabled client machine from a publisher's activation server (such as FlexNet Operations), ship these components:

- **An application activation utility and its activation library that support composite transactions**— Create and ship your own version of the following sample components (available with your Toolkit):
  - For Windows platforms, appcomptranutil.exe and the library appcomptranutil_libFNP.dll
  - For standard (tr) UNIX platforms, appcomptranutil and the library appcomptranutil_libFNP.so
  - For notr UNIX platforms, appcomptranutil and the library appcomptranutil_libFNP_notr.so

  For information on building your own application activation utility for composite transaction, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

  Note • *Although activations from a publisher's activation server still support single-action transactions, best practice is to use composite transactions to request and process activations from the activation server.*

- **SOAP communication component**—For internet-based transactions (programmatic activation) with a FlexNet Operations server, include FnpCommsSoap.dll (Windows platforms) or FnpCommsSoap.so (UNIX platforms).

For Local Activations

If you support local activations, provide these components.

***Note** • Best practice is to have the installer for your FlexEnabled application lay down these components. This method provides for a self-contained activation mechanism local to the end-user machine without the need to access an activation server.*

- **One or more activation specification records (ASRs)**—Create and ship one or more of these signed XML documents, each containing the license rights that get placed in local trusted storage on the FlexEnabled client machine. For more information, see the following in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*:
  - The chapter "Implementing Trial Licenses and Trial Packs"
  - The "asrgen" section in the chapter "Working with the Software Publisher Utilities"
  - The "B_ActivationSpecificationRecord.xsd" section in the appendix "XML Schemas"
- **An application activation utility and its activation library that support composite transactions**—Create and ship your own version of the following sample components (available with your Toolkit):
  - For Windows platforms, `appcomptranutil.exe` and the library `appcomptranutil_libFNP.dll`
  - For standard (`tr`) UNIX platforms, `appcomptranutil` and the library `appcomptranutil_libFNP.so`
  - For `notr` UNIX platforms, `appcomptranutil` and the library `appcomptranutil_libFNP_notr.so`

  For information on building your own application activation utility for composite transaction, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

  ***Note** • Although activations from a license server still support single-action transactions, best practice is to use composite transactions to request and process activations from the license server.*

For Short Code Transactions

If you support short code transactions, ship these components:

- **One or more activation specification records for short codes (ASRs)**—Create and ship one or more of these signed XML documents, each containing the license rights that get placed in local trusted storage on an end-user machine's. For more information, see the appendix "XML Schemas" (specifically the "B_ActivationSpecificationRecord.xsd" section) in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.
- **An application activation utility and its activation library that support request- and response-code transactions**—Create and ship your own version of the following sample components (available with your Toolkit):
  - For Windows platforms, `appcomptranutil.exe` and the library `appcomptranutil_libFNP.dll`
  - For standard (`tr`) UNIX platforms, `appcomptranutil` and the library `appcomptranutil_libFNP.so`
  - For `notr` UNIX platforms, `appcomptranutil` and the library `appcomptranutil_libFNP_notr.so`

  For information on building your own application activation utility that supports transactions for request and response codes, see the chapter "Building an Application Activation Utility for Composite Transactions" (specifically the "Transactions Using Request and Response Codes" section) in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Note • *Instead of shipping a stand-alone application activation utility, you can embed the utility's functionality within the FlexEnabled application or have the application's installer lay down the utility and library.*

License Server Machine Components

Ship these components if you support a served license model or if you require interaction with an activation server in order to manage license transactions in trusted storage. These components are installed on the license server and include the following:

- **License server manager**—Include `lmadmin` (`lmadmin.exe` on Windows) or `lmgrd` (`lmgrd.exe` on Windows).
- **Minimal license file**—Include a simple license file (with no license rights defined) used to inform the license server which vendor daemons to start up. The file must include the following basic information, where *vendor* is the name of your vendor daemon:

  ```
  SERVER thishost ANY
  VENDOR vendor
  ```

- **Your vendor daemon**—Include your vendor daemon *`vendor`* (*`vendor`*`.exe` on Windows) and its activation library component, which is one of the following:
  - *`vendor`*`_libFNP.dll` (Windows platforms)
  - *`vendor`*`_libFNP.so` (standard `tr` UNIX platforms)
  - *`vendor`*`_server_libFNP_notr.so` (`notr` UNIX platforms)
- **FlexNet Publisher utility programs**—Include `lmutil` (`lmutil.exe` on Windows). These utilities are used for license server management and administration.

  Note • *Some* `lmutil` *utilities are included as functionality in* `lmadmin`*. However,* `lmgrd` *uses all* `lmutil` *utilities. For more information, see the chapter "Migrating from lmgrd to lmadmin" in FlexNet Publisher's License Administration Guide.*

- **A server activation utility and its activation library that support composite transactions**—These components are used to activate licenses from a publisher's activation server to trusted storage on the license server. Create and ship your own version of the following sample components (available with your Toolkit):
  - For Windows platforms, `servercomptranutil.exe` and the library `servercomptranutil_libFNP.dll`
  - For standard (`tr`) UNIX platforms, `servercomptranutil` and the library `servercomptranutil_libFNP.so`
  - For `notr` UNIX platforms, `servercomptranutil` and the library `servercomptranutil_libFNP_notr.so`

  For information on building your own server activation utility for composite transactions, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

  You must also ship the application activation utility and its library for installation on the FlexEnabled client, as described in the previous section FlexEnabled Client Components).

Note • *Although activations from a license server still support single-action transactions, best practice is to use composite transactions to request and process activations from the license server.*

- **One or more activation specification records (ASRs)**—If you want to support local activation or short code activations, create and ship one or more of these signed XML documents, each containing the license rights that get placed in local trusted storage on the license server. For more information, see the following in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*:
  - The chapter "Implementing Trial Licenses and Trial Packs"
  - The "asrgen" section in the chapter "Working with the Software Publisher Utilities"
  - The "B_ActivationSpecificationRecord.xsd" section in the appendix "XML Schemas"
- **SOAP communication component**—For internet-based transactions (programmatic activation) with a FlexNet Operations server, include `FnpCommsSoap.dll` (Windows platforms) or `FnpCommsSoap.so` (UNIX platforms).
- **Entitlement and product ID information**—The end user requires this information to initiate license rights transactions. This information can be provided at a later date.
- **Instructions on how to perform activations**—Provide the end user with instructions on how to use the server activation utility to perform license rights transactions, as applicable.

Installer Considerations

You must provide an installer and installation instructions to your end user. This section highlights the areas that need special consideration.

Installing the FlexNet Licensing Service

You can install the FlexNet Licensing Service component, as listed in Table 6-1, using the application installer or the application initialization script. The recommendation is to include the FlexNet Licensing Service installation as part of the FlexEnabled application installation to avoid issues with specific user privileges that the FlexNet Licensing Service installation might require.

Note • *Make sure your end user is aware of the authorization requirements. On non-Windows platforms, the end user must have root privilege to install the FlexNet Licensing Service. On Windows, the end user must have administrator privileges.*

Table 6-2 • FlexNet Licensing Service Installers

Platform	FlexNet Licensing Service Installer
Windows	Install and uninstall functions provided in the FlexNet Licensing Service API.
OS X and Linux	`<platform_dir>/publisher/install_fnp.sh` This script both installs the FlexNet Licensing Service and initializes the trusted storage directory structure.

Table 6-2 • FlexNet Licensing Service Installers

Platform	FlexNet Licensing Service Installer
All other UNIX platforms	`<platform_dir>/publisher/install_fnp.sh` This script initializes the trusted storage directory structure. The FlexNet Licensing Service is not required on these platforms See Trusted Storage Directory Initialization (UNIX Platforms) for further details.

Important • *On most UNIX platforms, the trusted storage directory is typically* `/usr/local/share/macrovision/storage/FlexNet/`*. FlexNet Publisher does not support custom locations or symbolic links for the trusted storage directory.*

Trusted Storage Directory Initialization (UNIX Platforms)

For UNIX platforms on which trusted storage is accessed, you must provide a means to initialize the trusted storage directory structure. You must include logic similar to that provided in `<platform_dir>/publisher/install_fnp.sh`. Make sure your end user is aware they need to perform this initialization as `root`.

Installing the FlexEnabled Application

You must ensure that the activation library component—`program_libFNP.dll` (Windows platforms) or `program_libFNP.so` (standard UNIX platforms)—is installed in the correct location as specified in the preptool configuration file when the application was prepped. This is not an issue for notr platforms. Generally, this is the same location as the main executable file. See “Preptool Configuration File:Examples” in *Programming Reference for Trusted Storage–Based Licensing* for further details.

The FlexNet Licensing Service installer, `FNP_Act_Installer.dll` (Windows platforms) or `install_fnp.sh` (UNIX platforms), must be installed in the same folder as the FlexEnabled application.

Installing the Activation Utilities

If you provide separate activation utilities, you must ensure that their activation library components are installed in the exact location specified in the `preptool` configuration file when the application was prepared. Generally, this is the same location as the main executable file. See the chapter “Working with the Software Publisher Utilities” in *Programming Reference for Trusted Storage–Based Licensing* for information about using `preptool` and for examples of `preptool` configuration files.

SSL Certificates

If you have enabled client-side SSL certificate support for your transactions with the FlexNet Operations server and you are providing a certificate to your end user, you must ensure that you install your certificate in the location configured in your activation utility. See “flxActCommonHandleSetSSLDetails” in *C/C++ Function Reference* for further details.

Licenses from Trusted Storage

The default setting is for FlexEnabled applications to use licenses in the following order:

1. From trusted storage
2. From local license files
3. From license servers

Notify your end user if you have modified this order by:

- Using `lc_set_attr` with `LM_A_SORT_TS_FIRST` set to zero in the FlexEnabled application which turns off this default behavior.
- Using the *sort* element in feature definition lines to change the order in which feature definition lines are searched.
- Setting the `LM_A_LICENSE_DEFAULT` attribute setting inside the application.

For detailed information on these settings, see the *Programming Reference for License File–Based Licensing*.

Installing the License Server

If your license lifecycle requires a license server, you need to provide end users with an installer and instructions for installing the license server manager. For more information about installer types and installation considerations, see Chapter 10, Providing the License Server Installation Instructions.

What to Tell Your End Users

This section lists implementation decisions that you make that have a possible effect on end-user administration and the FlexEnabled product behavior relative to trusted storage–based licensing. The decisions presented here supplement those found in *Programming Reference for License File–Based Licensing*.

Licensing Agreements

Several of the FlexNet Publisher components for trusted storage that you install at your end-user site contain code and techniques derived from external third party products. For a list of copyright notices and licenses for these external libraries, see a supplementary document that accompanies this one. Forward the copyright information to your end user as designated by each third party.

Up-to-date FlexNet Publisher Components

Make sure your end user uses the latest license server manager, `lmadmin` or `lmgrd`, and licensing utilities, which are available from www.flexerasoftware.com.

The following diagram summarizes the rules for Toolkit component compatibility.

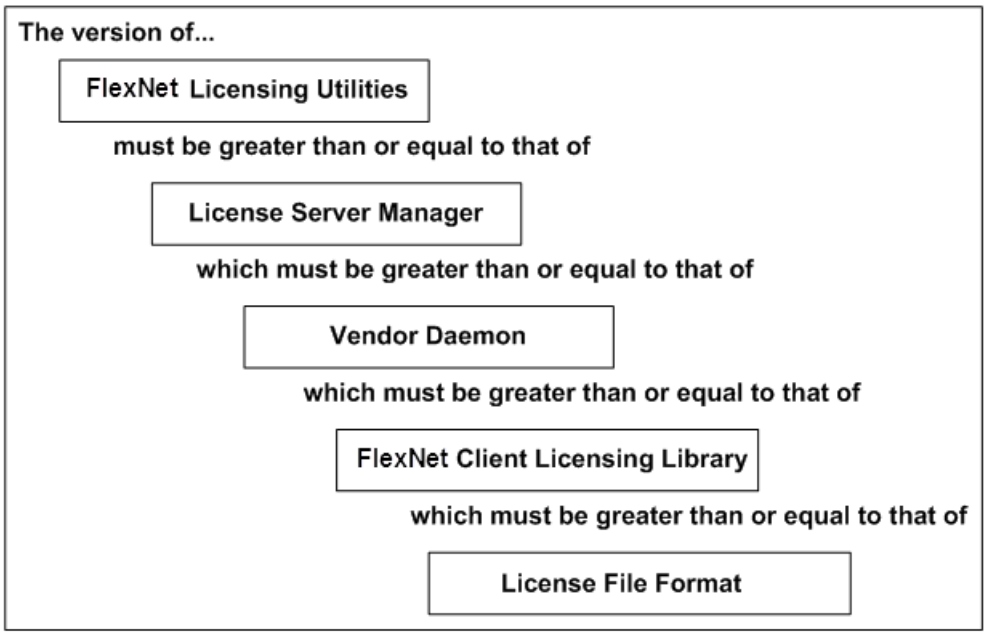

Figure 6-1: FlexNet Licensing Component Version Rules

License Server Need

Notify your end user of your requirement for licenses to be served by a license server. This requirement is dictated by your particular license model.

License Lifecycle Information

You need to supply information to your end user about your license lifecycle. Here are suggestions for the type of information you may want to include, as appropriate to your lifecycle:

- Instructions for using the application activation utility, as applicable
- Usage information for the server activation utility and instructions on performing license rights transactions, as applicable
- Repair transaction information
- Return transaction information
- Instructions for transferring license rights
- Local activation implementation details

Part 3
For License File–Based Licensing

This part of the Development Environment Guide includes the following chapters:

- Building Your FlexEnabled Application for License File–Based Licensing
- Distributing the FlexEnabled Application for License File–Based Licensing

7

Building Your FlexEnabled Application for License File–Based Licensing

The procedures in this chapter assume you have built and tested the production Toolkit with your publisher-specific information as detailed in First Build of the Production Licensing Toolkit or Rebuilding the Production Licensing Toolkit.

Before building your application as described in this chapter, add licensing functions to the application, as described in the *Programming Reference for License File–Based Licensing*. Read the important information in the next section, Security Considerations.

For further instructions see Building on UNIX or Building Windows FlexEnabled Applications as appropriate to your platform.

Security Considerations

Generating a New Version of lm_new for Each Build

Use a newly generated version of the `lm_new` object, `lm_new.o` (UNIX platforms) or `lm_new.obj` (Windows platforms), each time you link your application: This makes the product more secure.

Task ***To generate the lm_new object***

Do one of the following:

- Using the FlexNet Publisher toolkit `makefile`, change to the `<platform_dir>` directory, and enter the following commands:

  ```
  rm lm_new.o
  make lm_new.o
  ```

- Using the `lmnewgen` utility, change to the `<platform_dir>` directory, and enter the following commands:

  ```
  lmnewgen vendor_daemon_name -o lm_new.c
  cc -c lm_new.c -o lm_new.o
  ```

It is recommended that each project be provided with a different version of the `lm_new` object file. There are two approaches to providing this file:

- A central authority can generate and distribute the object file.
- `lmnewgen` and the vendor daemon name can be distributed to each project so the object file can be generated as needed. This approach is less secure because `lmnewgen` leaves the central authority.

Tamper-Resistant Licensing: Using TRL-Only Libraries

The TRL-only libraries are by default linked to the Toolkit. The TRL-only libraries provide increased security when using TRL licenses, because they allow only TRL licenses and not a combination of TRL and non-TRL licenses, as is the case with `lmgr.lib` and `liblmgr.a`.

- **Windows**—`lmgr_trl.lib`
- **UNIX**—`liblmgr_trl.a` or `libmgr_pic_trl.a`.

If you using TRL-only libraries, you cannot use the following attribute settings (that are used to provide backward compatibility with non-TRL licenses) within FlexEnabled applications built with these libraries:

- `LM_A_SIGN_LEVEL` set to zero
- `LM_A_STRENGTH_OVERRIDE` set to `LM_STRENGTH_LICENSE_KEY` or `STRENGTH_DEFAULT`

When upgrading to TRL, you must reissue license rights. See the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”, for more information.

Using Non-TRL Libraries

The Toolkit is linked to TRL-only libraries by default. Customers who cannot use TRL-strength keys (because they have clients older than v10.8.6 still in production which have to be supported by the same license server as new clients) should modify the toolkit makefile in order to be backward compatible.

- **Windows**—`lmgr.lib`
- **UNIX**—`liblmgr.a` or or `libmgr_pic.a`.

See the *Programming Reference for License File–Based Licensing*, chapter 9, “License Signature Configurations”, for more information.

Building on UNIX

A standard FlexEnabled application for UNIX platforms is built as a multithreaded executable. This is to accommodate the default automatic heartbeat scheme, which requires a multithreaded environment.

When reviewing the instructions in this section, refer to the build rules for `lmflex` (without Secure Data Types) and `lmflexsdt` (with Secure Data Types) in `<platform_dir>/makefile` for an example specific to your platform. Two build scenarios are presented based on whether you are implementing Secure Data Types in your application:

- Building the Application Without Secure Data Types
- Building the Application With Secure Data Types

Important • *For Mac applications, use OS X 10.5, 10.6, or 10.7 as appropriate to develop and build your FlexEnabled application. If you are building applications that run on 64-bit and 32-bit Mac platforms and they use FlexNet ID dongles, see Building Mac Applications With FlexNet ID Dongle Support.*

Note • *To build a FlexEnabled application that must be Linux Standard Base (LSB) certified, you must follow the guidelines of the Linux Foundation. For more information about Linux Standard Base certification, see the Linux Foundation's website: http://www.linuxfoundation.org. For details about LSB support in FlexNet Publisher, see your FlexNet Publisher Release Notes.*

Build Considerations for Shared Objects

The information in this section is applicable to FlexEnabled products that include one or more shared library components:

- On platforms where position independent code (PIC) is not the default build model, you must build as PIC and link with PIC libraries. These are distinguished with the `_pic` addition to the directory containing static libraries or static library name itself. This addition indicates the PIC variant.
- It is recommended that an exports file be used when linking a shared object so that only the public interface of the shared object is exported. If this is not done, all global symbols will be exported, which can cause problems with duplicate symbols in applications where there are multiple protected components.
- Both the method used in a link command to produce a shared object and the compile and linker command line flags for the exports file are different for each platform and build tool. Refer to the relevant compiler and linker documentation for details.

Building the Application Without Secure Data Types

Follow the instructions in this section if you do not have Secure Data Types (SDTs) implemented. For implementations that include SDTs, see Building the Application With Secure Data Types.

These instructions are written for applications developed in the C programming language. When reviewing the instructions, refer to the build rules for `lmflex` in `<platform_dir>\makefile` for examples specific to your platform. If your product includes shared objects, also review the section Build Considerations for Shared Objects. Additionally, it is recommended that your application be linked with dynamic OS libraries.

Task ***To build the application without SDTs***

1. Compile your application.
2. After you have successfully compiled your application, link with a command similar to the following:

```
cc -o <program> <program>.o $(OBJS) lm_new.o \
-L<platform_dir> $(LIBS) $(OSLIBS)
```

   Note the following:

   - `<program>` is your FlexEnabled application.
   - `$(OBJS)` is the list of the objects for your application.

- `lm_new.o` is generated as described in Generating a New Version of lm_new for Each Build.
- `<platform_dir>` is the platform directory in the production FlexNet Publisher toolkit that contains the necessary Licensing client libraries.
- `$(LIBS)` is a list of FlexNet Publisher client libraries. Consult the section UNIX Library Set for information about the library set.
- `$(OSLIBS)` is a list of platform-specific libraries. Refer to the rules for building the `lmflex` sample FlexEnabled application in `<platform_dir>/makefile` for the list specific to your platform.

To accommodate the automatic heartbeat feature, specify the multithreaded system libraries on the command line. Refer to your system's documentation for details on linking with multithreaded libraries. See *Programming Reference for License File–Based Licensing* for more details on implementing heartbeats.

UNIX Library Set

Use the following library set for standard, multithreaded applications.

- Object file: `lm_new.o`
- FlexNet Licensing client libraries:
  - `liblmgr_trl.a` or `liblmgr_pic_trl.a`
  - `libcrvs.a` or `libcrvs_pic.a`
  - `libsb.a` or `libsb_pic.a`
  - `activation/lib/libnoact.a` or `activation/lib/libnoact_pic.a`
  - Either `liblmgr_dongle.a` or `liblmgr_dongle_stub.a`

Note • *Keep these points in mind:*

- *Use a non-TRL library (`liblmgr.a` or `liblmgr_pic.a`) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*
- *If you do not include `lm_new.o` to your link list, an unresolved reference to `l_n36_buf` occurs during link time.*

Building the Application With Secure Data Types

Follow the build instructions in this section if you have implemented Secure Data Types (SDTs) into your code. These instructions refer to the sample application `lmflexsdt` to demonstrate the build process. When reviewing these instructions, refer to the build rules for `lmflexsdt` in `<platform_dir>\makefile` for examples specific to your platform. If your product includes shared objects, also review the section Build Considerations for Shared Objects.

Note • *Set the environment variables `CC` to `gcc` or `LD` to `g++` before proceeding. Use commands similar to* **`setenv CC gcc`** *or* **`setenv LD g++`**. *The syntax might vary with the particular command line shell in use. (Consult the "Detailed Platform Information" section of the FlexNet Publisher Release Notes for information about the compiler used for your platform.)*

Task ***To build the application with SDTs***

1. Use commands similar to the following to compile the SDT implementation file, `sdtdata.c`, and your application code:

```
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
    $(CFLAGS_COMMON) ../examples/sdt/lmflexsdtdefs.c
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
    $(CFLAGS_COMMON) ../machind/sdt/sdtdata.c
gcc -Imachind -I. -Imachind/activation/include -Imachind/sdt \
    $(CFLAGS_COMMON) ../examples/sdt/lmflexsdt.c
```

Command	Description
`machind`	The machine-independent directory as described in Identifying the Toolkit Directories.
`$(CFLAGS_COMMON)`	Additional flags required for your platform. Refer to the build rules for `lmflexsdt` in `<platform_dir>/makefile` for the flags specific to your platform.
`lmflexsdtdefs``.c`	The SDT implementation file for the sample application `lmflexsdt`. Replace this value with the path and name of your SDT implementation file. See Chapter 9, Implementing Secure Data Types (SDTs), for more information about the implementation file.
`sdtdata.c`	Component required when inserting SDTs in C code.
`lmflexsdt``.c`	The code file for the sample application `lmflexsdt`. Replace this value with the path and name of the file containing your FlexEnabled application code.

2. When you have successfully compiled your application, link with a command similar to the following:

```
g++ $(CFLAGS_COMMON) $(LDFLAGS) -o lmflexsdt lmflexsdt.o sdtdata.o lmflexsdtdefs.o lm_new.o \
            $(CLIENTLIB) $(FNPLOADLIB) $(XTRALIB) $(THREADLIB)
```

Command	Description
`$(CFLAGS_COMMON)`	Additional gcc flags required for your platform. Refer to the build rules for `lmflexsdt` in `<platform_dir>/makefile` for the flags specific to your platform.
`$(LDFLAGS)`	Required linker flags. Refer to the build rules for `lmflexsdt` in `<platform_dir>/makefile` for the required flags.
`lmflexsdt`	The file name of the sample application `lmflexsdt`, which uses SDTs. Replace this value with the file name of your FlexEnabled application containing SDTs.
`lmflexsdtdefs``.o` `sdtdata`*`.o`*	SDT object files. `lmflexsdtdefs.o` is the name of the object file generated for the sample implementation file `lmflexsdtdefs.c`. Replace this value with the name of your implementation object file.

Command	Description
`lm_new.o`	A security component generated each time you build your production Toolkit Licensing toolkit. It can be generated manually. See Security Considerations.

`$(CLIENTLIB)`
`$(FNPLOADLIB)`
`$(XTRALIB)`
`$(THREADLIB)`
	FlexNet Publisher and platform-specific libraries. Refer to these macros in `<platform_dir>/makefile` for the list specific to your platform.

Make sure that multithreaded system libraries are specified on the command line. Refer to your system's documentation for details on linking with these libraries. See the *Programming Reference for License File–Based Licensing* for more details on implementing heartbeats.

3. Prepare the application for SDT usage. See Running preptool on Applications with SDTs.

Building Mac Applications With FlexNet ID Dongle Support

Use these procedures when building Mac applications that support FlexID dongles.

Link to Appropriate Library

Like Linux applications, Mac applications must link to either the dongle library or the stub library, even if the application does not include dongle support. Make sure your application links to one of these libraries.

Task ***To include or omit dongle support***

Perform one of the following:

- Link to `liblmgr_dongle.a` to include dongle support.
- Link to `liblmgr_dongle_stub.a` to omit dongle support.

Create 32-Bit Applications

FlexNet ID dongles are not supported for 64-bit applications running on Mac 64-bit platforms. Products that use FlexNet ID dongles and which may be run on 64-bit platforms should be released as 32-bit applications only.

Task ***To create a licensed application binary that contains only 32-bit versions***

Build the licensing toolkit in the appropriate Mac development environment. (See the latest *FlexNet Publisher Release Notes* for up-to-date environment requirements for your OS X Mac platform.) This toolkit can be used to build products for all the architectures that the toolkit supports including 64-bit. Ensure that your product build instructions build only 32-bit applications (select only 32-bit architectures within the SDK you are using).

Building Windows FlexEnabled Applications

To build a FlexEnabled application on a Windows platform, you *must* use a supported Microsoft Visual Studio version, as listed in the latest *FlexNet Publisher Licensing Toolkit Release Notes*.

Two build scenarios are presented based on whether you are implementing Secure Data Types in your application:

- Building the Application Without Secure Data Types
- Building the Application With Secure Data Types

Building the Application Without Secure Data Types

Follow the instructions in this section if you do not have Secure Data Types (SDTs) implemented. For implementations that include SDTs, proceed to Building the Application With Secure Data Types.

The procedure described in this section builds your FlexEnabled application for the recommended Microsoft Visual C++ environment. When reviewing these instructions, refer to the build rules for `lmflex.exe` in `<platform_dir>\makefile` for examples specific to your platform.

For other development environments, you might need to make modifications. For more information, see Development Environments other than Microsoft Visual C++.

Note • *Make sure that you have the command-line interface of the Microsoft Visual C++ development environment correctly configured. Consult the Visual C++ documentation for details about building from the command line.*

Task ***To build the application without SDTs***

1. Include the header files in the `<install_dir>\machind` directory.
2. Compile the application.
3. To build an application that has a static multithreading run-time, use the /MT switch. (See Using a Static or Dynamic Multithreaded Run-Time)
4. Statically link your application to FlexNet libraries and files as detailed in Table 7-1. (Static linking is recommended. See Dynamically Linking Your Application.)
5. Build the application.

Table 7-1 • Libraries and Files for Statically Linked Multithreaded Run-time

FlexNet Publisher Object File	FlexNet Publisher Client Libraries	MSVC++ Libraries
`lm_new.obj` (see Generating a New Version of lm_new for Each Build)	• `lmgr_trl.lib` ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.* • `libcrvs.lib` • `libsb.lib` • `activation\lib\libnoact.lib` • Either `lmgr_dongle.lib` (for dongle usage) or `lmgr_dongle_stub.lib` • `libredir_std.lib`	• `libcmt.lib` (for static multithreaded runtime) • Additional required libraries as listed in the `<platform_dir>/makefile` file

Example Build Instructions

This example is for applications developed in the Microsoft Visual C++ environment. General instructions are given. For specific details relating to the version of Visual Studio you are using, refer to Setting Up Microsoft Visual Studio for a FlexEnabled Application.

The example Win32 FlexEnabled application, `lmflex.c`, is used as an example throughout this section. This application is part of the FlexNet Publisher distribution. The source is located in the `machind` folder, `<install_dir>\machind`.

Instructions are given for:

- Compiling Your Application Using the Microsoft Visual C++ IDE
- Compiling Your Application Using the Microsoft Visual C++ Command Line

Compiling Your Application Using the Microsoft Visual C++ IDE

Task ***To compile your application with the Microsoft Visual C++ IDE***

1. From within the IDE, create a Win32 Application project that includes the C source file, `lmflex.c`, found in the `machind` folder.
2. Modify your build environment to include the path to the Licensing header file directory:

   `<install_dir>\machind`

3. Compile `lmflex.c` using the build mechanism supplied by the IDE.

The following steps describe which libraries and object files are needed to be linked into your application. A static linking model is presented.

Task ***To link your application with the IDE***

1. Add the `<platform_dir>` folder to the global library search path.
2. Set the following link-time options and libraries:
   - `/nodefaultlib`
   - FlexNet Licensing and system libraries listed in Table 7-1
3. Link the application using the build mechanism supplied by the IDE.

Compiling Your Application Using the Microsoft Visual C++ Command Line

Task ***To compile your application from the Microsoft Visual C++ command line***

1. Add **`/I "<install_dir>\machind"`** to your `nmake` makefile, batch file, or `cl` command line.
2. Compile the application. Add the following to your `nmake` makefile, batch file or enter it on the `cl` command line:

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

3. When you have successfully compiled your application, link with a command similar to the following:

   ```
   LINK /nologo /NODEFAULTLIB /MT /out:lmflex.exe lmflex.obj
                $(LIBS)
   ```

   where `$(LIBS)` is a list of Licensing client libraries. Consult Table 7-1 for details of the libraries. Link the application with a newly generated version of `lmnew.obj` as described in Generating a New Version of lm_new for Each Build.

   The `lmflex` linking command line for an /MT example looks like:

   `C:> LINK /nologo /NODEFAULTLIB /MT /out:lmflex.exe lmflex.obj lm_new.obj lmgr_trl.lib libcrvs.lib libsb.lib activation\lib\libnoact.lib libcmt.lib...`(see the contents of the file `<platform_dir>/makefile` for a complete list of libraries to include)

Using a Static or Dynamic Multithreaded Run-Time

FlexNet Licensing requires a multithreaded run-time environment. This can be provided in Visual C++ using either static or dynamic libraries and files.

Important • *Using static versions is recommended because the static libraries and files are more secure.*

Consider the following when making this decision:

- Static libraries are more secure than dynamic libraries. A hacker can replace a DLL on the system to find out the library and system calls used by FlexNet Licensing.
- Applications using static libraries are larger than the same application using dynamic libraries.
- Dynamic libraries provide a quick method of applying Microsoft updates. An end user of an application that uses a dynamic library can download an updated DLL to update the application. Applications using static libraries must be rebuilt to apply the update.
- Applications using dynamic libraries will not run if the DLLs are not on the system.

If you must use dynamic libraries, include the `/MD` switch. Then link to the run-time libraries, as listed in Table 7-2.

- **Using the IDE**—Modify the link-time command line to include the `/MD` switch.
- **Using the command line**—Include `/MD` on the `cl` command line.

Table 7-2 • Files and Libraries for Dynamic Multithreaded Run-Time

FlexNet Publisher Object File	FlexNet Publisher Client Libraries	MSVC++ Libraries
`lm_new_md.obj` (see Generating a New Version of lm_new for Each Build)	• `lmgr_trl.lib` ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.* • `libcrvs.lib` • `libsb.lib` • `activation\lib\libnoact.lib` • Either `lmgr_dongle.lib` (for dongle usage) or `lmgr_dongle_stub.lib` • `libredir_std.lib`	• `msvcrt.lib` (for dynamic multithreaded runtime) • Additional required libraries as listed in the `<platform_dir>/makefile` file

Dynamically Linking Your Application

As an alternative to static linking, you can link your application using a DLL model. The recommended method of implementing dynamic linking is to use Microsoft Visual C++ to create your own DLL that links in the FlexNet Licensing client library, for example `<platform_dir>/lmgr_trl.lib`. This technique effectively creates a DLL wrapper around the Licensing static client library.

Creating a DLL that Contains the Licensing Client Library

To create a DLL that contains the Licensing client library, you write a DLL that is statically linked to the Licensing client library and contains functions that call the Flexible API functions. The following are recommended when writing this DLL:

- Include several calls to Flexible API licensing functions within a single exported function within the DLL.

- Use external function names that do not reflect the licensing functionality. This makes it harder for hackers to identify that licensing functions are being used.
- Include other (non-licensing) functions used by the application in the DLL.

The following instructions outline this process.

Task ***To create a DLL that contains the Licensing client library***

1. Write a DLL containing your own functions to implement the set of licensing functions that your applications require.
2. Statically link your DLL to the appropriate Licensing client libraries. See Table 7-1.

Using a DLL that Contains the Licensing Client Library

Task ***To use a DLL that contains the Licensing client library***

1. Define `FLEXLM_DLL` when compiling your FlexEnabled application - Define FLEXLM_DLL.
2. Set lc_new_job Arguments for Dynamic Linking and TRL (not required if you are not using TRL)
3. Ship your DLL with your FlexEnabled application.

Define FLEXLM_DLL

When compiling your source code, you must make sure the `FLEXLM_DLL` compile-time switch is defined.

Table 7-3 • Defining FLEXLM_DLL

Location	Description
Command line	`cl /DFLEXLM_DLL`
In the source code	Add `#define FLEXLM_DLL` before any FlexNet Licensing headers are included.
In the IDE	From the Project menu, choose Settings and click the **C/C++** tab. Add FLEXLM_DLL to the Preprocessor Definitions.

Set lc_new_job Arguments for Dynamic Linking and TRL

The second argument to `lc_new_job` **must be** `NULL`, as in the following example.

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

Use the string `lc_new_job_arg2` as the second argument in non-DLL code. Then, you must statically link `lm_new.obj` into your application.

Development Environments other than Microsoft Visual C++

All FlexNet Licensing client libraries and object files in the distribution are shipped in a format compatible with Microsoft Visual C++. For development environments other than Microsoft Visual C++ (for example Visual Basic, C#, or a different C development environment such as Borland), your application is dynamically linked to the Licensing client routines using a DLL.

It is recommended that you create your own DLL as described in Creating a DLL that Contains the Licensing Client Library.

Building the Application With Secure Data Types

Follow the build instructions in this section if you have implemented Secure Data Types (SDTs) into your code. The instructions refer to the sample application `lmflexsdt.exe` to demonstrate the build process. When reviewing the instructions, refer to the build rules for `lmflexsdt.exe` in `<platform_dir>/makefile` for examples specific to your platform.

Task ***To build the application with SDTs***

1. Use commands similar to the following to compile the SDT implementation file, `sdtdata.c`, and your application code:

```
cl /DWIN32 $(CFLAGS_NOOPT) /Imachind /I. /Imachind\activation\include \
             /Imachind\sdt ..\machind\sdt\sdtdata.c
cl /DWIN32 $(CFLAGS_NOOPT) /Imachind /I. /Imachind\activation\include \
             /Imachind\sdt ..\examples\sdt\lmflexsdtdefs.c
cl $(CFLAGS_NOOPT) /Imachind /Imachind\activation\include \
              /Imachind\sdt ..\examples\sdt\lmflexsdt.c
```

Command	Description
`$(CFLAGS_NOOPT)`	Additional compiler flags required for your platform. Refer to the build rules for `lmflexsdt` in `<platform_dir>\makefile` for the flags specific to your platform.
`machind`	The machine independent directory, as described in Identifying the Toolkit Directories.
`sdtdata.c`	Component required when inserting SDTs in C code.
`lmflexsdtdefs``.c`	The SDT implementation file for the sample application `lmflexsdt.exe`. Replace this value with the path and name of your SDT implementation file. See Chapter 9, Implementing Secure Data Types (SDTs), for more information about the implementation file.
`lmflexsdt``.c`	The code file for the sample application `lmflexsdt.exe`. Replace this value with the path and name of the file containing your FlexEnabled application code.

2. When you have successfully compiled your application, link with a command similar to the following:

```
LINK /nologo /NODEFAULTLIB /OPT:NOREF /out:lmflexsdt.exe lmflexsdt.obj \
             sdtdata.obj lmflexsdtdefs.obj $(LMNEW_OBJ_CLIENT) \
```

```
$(CLIENTLIB) $(FNPLOADLIB) $(XTRALIB)
```

Command	Description
lmflexsdt.exe	The file name of the sample application `lmflexsdt.exe`, which uses SDTs. Replace this value with the file name of your FlexEnabled application containing SDTs.
lmflexsdtdefs.obj sdtdata.obj	SDT object files. `lmflexsdtdefs.obj` is the name of the object file generated for the sample implementation file `lmflexsdtdefs.c`. Replace this value with the name of your implementation object file.
$(LMNEW_OBJ_CLIENT)	The `lm_new.obj` file. This is a security component generated each time you build your production toolkit. It can be generated manually. See Security Considerations.
$(CLIENTLIB)	FlexNet Publisher client libraries: • `libcrvs.lib` • `libsb.lib` • `lmgr_trl.lib` ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients.*
$(FNPLOADLIB)	The `libFNPload.lib` used to load the library created using `preptool`.
$(XTRALIB)	Additional dongle and Windows system libraries: • `lmgr_dongle.lib` (for dongle usage) or `lmgr_dongle_stub.lib` (no dongle usage) • Windows system libraries (view the contents of the `<platform_dir>/makefile` file for a complete list)
$(LMSTDREDIR_LIB)	FlexNet Publisher client library: `libredir_std.lib`

3. Prepare the application for SDT usage. See Running preptool on Applications with SDTs.

Signing the Application

Mac OS X and Windows support the digital signing of executable code as well as other files. This enables both the operating system and the user to know the authenticity of files that are installed on the system.

Files provided within the version of the production toolkit that is used on Mac OS X or Windows are signed unless they are recompiled during use. Files that you create when you are creating a licensed product, including the licensed application, you should sign yourself. The following Table 7-4 for Mac OS X platforms lists files and indicates which are provided as signed files and which you need to sign. See Table 1-1, FlexNet Publisher Licensing Toolkit files, on page 27, for information relevant for Microsoft platforms.

For detailed information on signing, read the relevant manufacturer's documentation.

Table 7-4 • Code Signing on Mac OS X

File Name	Description	Code Signing
	FlexEnabled application	Recommended
	Vendor daemon	Recommended
FnpCommsSoap.dylib	SOAP communications component for transactions with FlexNet Operations	Signed
lmadmin	Web-based license server manager	Signed
lmgrd	Command-line license server manager	Signed
lmrand1	Random number generator used to generate `LM_SEEDS`	Signed
lmstrip	Utility used to obfuscate and add security to FlexEnabled applications	Signed
lmutil	Utilities used for license server management and administration	Signed

Testing the Application

Task ***To create a license file for your application***

1. Using a text editor, open `mycounted.lic` from the `<platform_dir>` directory.
2. Modify the `SERVER`, `VENDOR`, and `FEATURE` lines to include server information and information for the features that your application checks out. In place of a signature on each `FEATURE` line, enter **`SIGN=0`**. See *Programming Reference for License File–Based Licensing* for information about the allowed fields and syntax of a license file.
3. Save `mycounted.lic`.
4. Run `lmcrypt` to calculate the signatures and create a valid license file.

   ```
   lmcrypt mycounted.lic
   ```
5. Put your application's license file where your application expects to find it.
6. Start your application and perform a task that checks out one of your licenses.

Using lmstrip for Additional Security

The `lmstrip` utility obfuscates symbol names in compiled object files and executables—specifically targeting FlexNet Licensing external symbol names and internal symbol names. Obfuscating these symbol names has the following benefits:

- Obfuscation provides a level of security against attempts to reverse engineer your product in order to remove licensing.
- Obfuscation avoids symbol conflict issues at link time when two or more FlexNet Enabled libraries are linked together into a single executable. This is especially beneficial if your end product is a linkable library object. (See Linking with a Library that Already Uses FlexNet Licensing.)

The `lmstrip` utility is located in the `<platform_dir>` directory of the Toolkit.

lmstrip Variants

The FlexNet Publisher Licensing Toolkit includes one of the following two variants of the `lmstrip` utility, depending on the platform:

- **lmstrip for Windows**—This variant is included in all Windows platform toolkits.
- **lmstrip for UNIX (ELF Platforms)**—This variant is included in the following UNIX platforms that support the Executable and Linking Format (ELF) object file format:
  - x86/x64 Linux
  - Sun Sparc Solaris
  - x86/x64 Solaris

Note • *The* `lmstrip` *utility is not included in toolkits for other platforms.*

lmstrip for Windows: Functionality and Usage

On Windows platforms, the static libraries included with the Toolkit are already processed with `lmstrip` to remove all the internal licensing symbols. Only the external symbols remain in those libraries to aid in object linking to generate an executable.

Securing the Executable

Microsoft Visual Studio linkers do not typically preserve COFF symbols (unless the executable is linked with an option like /DEBUG or similar). Therefore, the licensing symbol information will not be present in the final executable, and there is usually no need to run `lmstrip` on the executable program generated from linking with the licensing libraries.

Securing the Object Library

If your end product is an object library that includes FlexNet Publisher Licensing objects, you can run `lmstrip` on the object library to obfuscate the external licensing symbols. This process secures the object library against license-related hacks and avoids potential symbol conflict issues. (See Linking with a Library that Already Uses FlexNet Licensing.)

Windows lmstrip Usage

```
lmstrip -t lmstrip-symbol-map.xml lmstrip-symbol-new.xml filename
```

- `lmstrip-symbol-map.xml`—Original symbol map file provided in the toolkit, containing all possible symbols for obfuscation across all supplied FlexNet Publisher libraries.
- `lmstrip-symbol-new.xml`—Updated symbol map file produced by `lmstrip` after operating on filename. Original symbol names from `lmstrip-symbol-map.xml` are retained, and obfuscated names are added.
- `filename`—Object file or static library to process.

Usage Example

If two third-party producers, producer1 and producer2 produce static libraries `p1.lib` and `p2.lib`, both of which have been linked to FlexNet Publisher's `lmgr_trl.lib`, and both link into the same destination executable, then at least one of the producers must run `lmstrip`. Assume producer1 runs `lmstrip`:

1. Compile producer1's source file `p1.c` to `p1.obj`. `p1.c` includes FlexNet Publisher's `lmclient.h`.
2. Link producer1's object file to FlexNet Publisher libraries, generating `p1.lib` (excluding Windows system libraries, for brevity).

   ```
   lib p1.obj lm_new.obj lmgr_trl.lib lmgr_dongle_stub.lib libcrvs.lib libsb.lib libnoact.lib
   /OUT:p1.lib
   ```

3. `lmstrip p1.lib`

   ```
   lmstrip lmstrip-symbol-map.xml lmstrip-symbol-new.xml p1.lib
   ```

4. Retain `lmstrip-symbol-new.xml` for translating the stack trace if debugging is required.

Deprecated Windows lmstrip Usage

The following legacy usage of `lmstrip` has been deprecated and may be removed in a future release.

```
lmstrip [-e] [-i] [-v | -q] [-r]
    [-m[rw] [mapfile]] [-o string_file] filename
```

Task ***To display a list of all licensing internal and external symbol names to be obfuscated***

```
lmstrip -l
```

Task ***To display a list of non-licensing symbol names in* filename**

```
lmstrip -p filename
```

Task ***To suppress obfuscation and create a mapfile***

```
lmstrip -x mapfile
```

where:

Table 7-5 • lmstrip Windows inputs

Input	Description
`-e`	Do not obfuscate FlexNet Licensing external symbol names.
`-i`	Do not obfuscate FlexNet Licensing internal symbol names.
`-v`	Verbose mode. In addition to standard status information this option displays all symbol names, along with their new names, that have been obfuscated.
`-q`	Quiet mode. All output is suppressed.
`-mr [mapfile]`	Reads mapfile *`mapfile`*. Forces randomized names to be the same across invocations. Default mapfile name is `lmstrip.map`.
`-mw [mapfile]`	Writes mapfile *`mapfile`*. This file contains a lookup table of obfuscated symbol names that can be used in later invocations. Default mapfile name is `lmstrip.map`.
`-mrw [mapfile]`	Reads and updates mapfile *`mapfile`*. Default mapfile name is `lmstrip.map`.
`-o string_file`	A file containing one or more vendor-specified symbol names to obfuscate in `filename`. One symbol name per line of the form: ● `+symbol`–Adds the symbol to the list of vendor-specified symbols to be obfuscated. ● `-symbol`–Removes the symbol as an obfuscation candidate. `symbol` can be either a FlexNet Licensing symbol name or a vendor-specified symbol name.
`filename`	Name of the file to process.
`-p`	Displays the list of non-FlexNet Licensing symbol names in `filename`. Used in conjunction with `-mr mapfile`, symbols that have already been obfuscated are suppressed from the list.
`-x mapfile`	Do not obfuscate anything. This option just creates mapfile *`mapfile`*.
`-l`	Displays a list of all FlexNet Licensing internal and external symbol names to be obfuscated.

lmstrip for UNIX (ELF Platforms): Functionality and Usage

On ELF-based UNIX platforms that `lmstrip` supports, the static libraries included with the Toolkit are already processed with `lmstrip` to remove all the internal licensing symbols. Only the external symbols remain in those libraries to aid in object linking to generate an executable.

Securing the Executable

It is not necessary to use the `lmstrip` command on an executable on which the strip command has been run. Instead, on ELF-based UNIX platforms, use the UNIX `strip` command to remove symbol table information from the final executable.

Securing the Object Library

If your end product is a static object library that includes FlexNet Publisher Licensing objects, you can run `lmstrip` on the object library to obfuscate the external licensing symbols. This process secures the object library against license-related hacks and avoids potential symbol conflict issues. (See Linking with a Library that Already Uses FlexNet Licensing.)

Note • *On UNIX platforms,* `lmstrip` *is not currently supported for use with dynamically linked libraries.*

lmstrip Command Line Options

`lmstrip` *`[options] objects...`*

where `options` represents optional command-line arguments and `objects` represents a required specification of one or more object files (.o) or library archives (.a). The table, Command Line Options for `lmstrip`, describes the available options.

Table 7-6 • Command Line Options for lmstrip

Option	Description
-h [--help]	Prints help message.
--version	Prints program version number and exits.
-m [--map] arg	Path to the symbol map file. Default: `<current_dir>/lmstrip_symbol_map.xml`. This file is updated unless the -o option is specified.
-o [--output-map] arg	Path to an alternative output map file to write. Default: `<current_dir>/lmstrip_symbol_map.xml`.
-l [--list] arg	Path to a symbol list file to process. Default: `<current_dir>/lmstrip_symbol_map.xml`.
-p [--prefix] arg	Specifies a unique prefix to strengthen the obfuscation process. Recommended value: Your assigned vendor daemon name. Default: `0x`.
-s [--seed] arg	Supplies a specific seed to prime the obfuscation algorithm. Default: <current time>.

Usage Example: Basic

```
% lmstrip -m lmstrip_symbol_map.xml -o mysymbol_output_map.xml -l mysymbol_list.xml
    -p mycompany_ -s 20101231 mylibrary.a
```

Table 7-7 •

Element	Description
lmstrip_symbol_map.xml	The list of symbols to be obfuscated.
mysymbol_output_map.xml	The symbol mapping between the original and obfuscated string.
mysymbol_list.xml	The list of symbols present in the object file after obfuscation.
mycompany_	All obfuscated symbols will have the `mycompany_` prefix.
20101231	Used to prime the obfuscation. (Date is used in YYYYDDMM format.)
mylibrary.a	The static library archive file to be obfuscated.

Usage Example: Best Practice

```
% lmstrip mylibrary1.a mylibrary2.a
```

This example assumes the directory from which you invoke `lmstrip` contains the file `lmstrip_symbol_map.xml` that is included as part of the FlexNet Publisher Licensing Toolkit.

The above usage can be repeatedly invoked on different object files (or archive libraries) and will result in a consistent obfuscation scheme across all object files. This is possible because, by default, `lmstrip` updates the symbol map file that it reads in as input. Additionally, because the contents of the map file dictate the symbol obfuscation, a consistent obfuscation scheme will be used for all the object files across multiple invocations of `lmstrip`.

lmstrip Usage Notes

- `lmstrip` modifies an object file in place. Be sure to make a backup of the object file before running `lmstrip` on it.
- Always safely archive a copy of the output map file, as `lmstrip` uses a one-way obfuscation algorithm that cannot be easily reversed. The output map file contains a mapping of the in-clear symbols and their corresponding obfuscated names. This file is useful in cases where you need to debug a crash report from the field that includes a stack trace. If you lose this map file, there is no way to regenerate it after the fact (unless you used the -s option, as described below).
- The -s option specifies a seed value that primes the obfuscation algorithm. By using a hard-to-guess but internally predictable seed value, you will be able to generate a predictable obfuscated symbol map. For example, you may use the date of your product build as a seed value which can help you recover the symbol map in the future, if need be.
- The -p option specifies a prefix to the obfuscated symbols. You can use the vendor daemon name assigned to your product by Flexera as a globally unique string. This option is especially useful if your end product is a static library.
- `lmstrip` on Linux platforms is not Linux Standards Base (LSB) compliant.

Linking with a Library that Already Uses FlexNet Licensing

You can protect against symbol name conflict among multiple client licensing libraries by using `lmstrip` on each individual licensing library. This technique obfuscates the FlexNet Licensing symbol names locally to each library, eliminating Licensing symbol name conflict when two or more such libraries are linked into an application binary.

The resulting object file or library can be linked with a library that already has references to Licensing symbol names, regardless of the Licensing version. Both coexist successfully.

8

Distributing the FlexEnabled Application for License File–Based Licensing

To ensure that your end user can use your FlexEnabled product easily and successfully, use the information in this chapter as a guideline for distributing your FlexEnabled application.

FlexNet Publisher Components Your End User Requires

When you ship your FlexEnabled application, you need to include Licensing end-user components in your software installation kit, in addition to the files that you normally ship. Table 8-1 lists these additional end-user components.

Table 8-1 • FlexNet Licensing Components for End-User Distribution

Component	Description
Components Required on all Platforms	
• `lmadmin` (`lmadmin.exe` on Windows) or `lmgrd` (`lmgrd.exe` on Windows) • `vendor` (`vendor.exe` on Windows)	The license server manager and vendor daemons. Include these if you distribute either counted or served uncounted licenses.
• `lmutil.exe` and `lmtools.exe` (Windows only) • `lmutil` (other platforms)	The licensing toolkit utility programs. Used for management of `lmadmin`, `lmgrd` and license administration.

Table 8-1 • FlexNet Licensing Components for End-User Distribution

Component	Description
• MSI-compliant installer or a command-line tool similar to `installanchorservice.exe` (Windows) • `install_fnp.sh` (script provided with the toolkit) or an equivalent installer that performs the same actions (other platforms where present in the toolkit)	The FlexNet Licensing Service is a recommended component. If it is not included, TPM-related and virtualization features are not available.
Dongle driver installers	Drivers required if you support FLEXID hostids for dongles. See the *Driver Installation Guide for FlexNet ID Dongles* for details about the driver installers for the different platforms.
Additional Windows Components	
• `eventlogadd.reg` • `eventlogremove.reg` ***Note*** • *For* `lmadmin`*, these files are located in* `lmadmin\eventlog`*.*	REGEDIT4 scripts for providing Windows event-logging support

For `lmadmin`, provide `lmadmin\eventlog\FnplmLicenseServerMsgs.dll`
`VendorLicenseServerMsgs.dll`
For `lmgrd`, provide these files:
• `VendorLicenseServerMsgs.dll`
• `FnplmLicenseServerMsgs.dll`
	Resource-only message file DLLs for Windows event-logging
	`FnplmLicenseServerMsgs.dll supports both lmgrd and lmamin error messages, with specific functionality.`
	`VendorLicenseServerMsgs.dll is used for vendor daemon messages.`

In addition to the components in Table 8-1, your end users eventually need a license certificate. Depending on your internal policies, this can be shipped with your installation kit, or shipped separately.

Generating the License Certificate

The `lmcrypt` utility can be used to generate licenses for your end users. The source to `lmcrypt` is in the `machind` directory and can be used by you as the basis for your own license-generation utility

The `lmcrypt` utility generates the same signatures on all FlexNet Licensing-supported platforms for all Licensing versions, allowing you to create licenses for any supported platform on any other supported platform.

When you know the format of the license certificate feature definition lines that you need for your license model, you create a license certificate template. `lmcrypt` replaces the signature in this template, which can be entered simply as `SIGN=0`, with the correct signature. The license certificate is then ready to ship to the end user. To use `lmcrypt`, you need to either create your own license template or base it on an example license certificate available in the `examples/licenses` directory.

lmcrypt Usage

```
lmcrypt [files][-i infile] [-o outfile] [-maxlen n] [-e errfile]
       [-verfmt { 7_1 | 8_0 | 8_1 | 8_4 | 10_1 | 10_8 | 11_4 }]
        [-help]
```

If no input file is specified, or if specified as "–" or stdin, standard input is used. If no output file is specified, or if specified as "–" or stdout, standard output is used. *`files`* are read and written back in place. If no file arguments are specified, `lmcrypt` reads from stdin and writes to stdout. All signatures are recomputed.

`lmcrypt` works only on lines that have a vendor name matching the vendor's daemon name. If this is not possible, an error is produced, and the affected license line is left unaltered.

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

Task ***To demonstrate lmcrypt usage***

1. Copy an existing good license file to another name, for example, `newlicense`.
2. Edit `newlicense` and make any desired changes, such as changing the feature name, the number of licenses, or adding new features.
3. Save the file.
4. Change the existing signature on each `FEATURE` line to `SIGN=0`.
5. Type the following: **`lmcrypt newlicense`**. `newlicense` is then filled with correct signatures and is usable by an end user. Comments are passed through unaltered.

lmcrypt Error Returns

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

Input

```
FEATURE f1 demo 1.a50 01-jan-2017 uncounted HOSTID=08002b32b161 \
        SIGN=0
```

Error Reported

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

Delivering the License Certificate

It is recommended that the license certificate be delivered by email as an attachment. If delivered in the body of the email, the entire email message can be saved in the specified location; email headers and extraneous text are automatically ignored by FlexNet Licensing. Emailers can and do alter license certificates. We attempt to accommodate most emailer alterations, but not all are accommodated.

Considerations when emailing your license certificate include:

- **New line additions**—Emailers often insert new lines into text, such as a license certificate. This does not cause a problem; however, it is important that comments that appear between license certificate lines are prefixed with "#." Comments appearing before or after all lines do not require this (except the first line after the last feature definition line). Then, email messages can be saved with email headers intact, and this is a good way to recommend saving a license certificate.
- **Inadvertent addition of .txt extension to the license certificate name**—When saving a text file, either in the emailer, or with the various text editors on Windows, it is common that a `.txt` extension is appended, often with no warning to the user. The user needs to ensure that `.txt` is not appended to the license certificate name.
- **Quotes**—ASCII quotation marks are sometimes substituted with other special characters.
- **Word Format, Rich Text, and so on**—FlexNet Licensing recognizes license certificates saved as ASCII or UTF-8 encoded text, only. Do not save it in Word Format, Rich Text Format, or any other similar encoding.

Single License Server vs. Redundant License Servers

You have to help your end user decide how many license server machines to use, as the signatures are partially derived from the list of license server machine hostids in a license file. See *Programming Reference for License File–Based Licensing* for guidelines about setting up license servers.

If a user configures a three-server redundant server system, a copy of `lmadmin` or `lmgrd`, your vendor daemon, and the license files should be placed on the disk of *each* license server node.

Enabling Support for Windows Event-Logging

The FlexNet license server emits status and event information to the debug log at the end user's site. If supported, selected events are additionally logged to the Windows event log. The messages logged to the event log are a subset of those logged to the Licensing debug log and correspond to significant license server events for which the end user's license administrator may need to intervene.

You have the option of providing support to your end user for the logging of FlexNet license server events to the Windows event log. Providing support involves:

- Shipping the Message File DLLs

- Configuring the End User's Registry
- Optionally, Localizing the Event-Logging Message Files

Shipping the Message File DLLs

Central to Windows event-logging are the message file DLLs. Event-logging message file DLLs contain the messages that are logged to the Windows Event Log. Messages can be localized as appropriate for your end user.

Three message file DLLs are shipped in the licensing toolkit, and, as shipped, are in English:

- `VendorLicenseServerMsgs.dll`—Contains messages for vendor daemon–specific events.
- `FnplmLicenseServerMsgs.dll`—Contains messages for license server manager–specific events; supports `lmgrd`- and `lmadmin`-specific events .

Event-logging for vendor daemon–specific and license server manager–specific events can be controlled independently on the end user's license server machine.

These resource-only DLLs are provided in the `<platform_dir>` folder of the licensing toolkit. Ship these files, as is, or ship your localized version of them. Provide your end user guidelines for installing these files according to your vendor-specific installation model. See Localizing the Event-Logging Message Files for further information on localizing message file DLLs.

Configuring the End User's Registry

In order for the FlexNet license server (`lmadmin` or `lmgrd`) to log messages to the Windows event log, event-logging must be enabled via settings in the end user's registry. For your convenience, the licensing toolkit contains sample REGEDIT4 scripts that can be used for automating these registry entries.

For details of the `lmadmin` scripts, see the *License Administration Guide*.

For `lmgrd`, these scripts are located in the `<platform_dir>` folder:

- `eventlogadd.reg`—Template for installing event-logging support via registry entries on the end-user's license server machine. These registry entries are used for configuring event-logging.
- eventlogadd32on64.reg — Template for installing event-logging support via registry entries on the end-user's license server machine. These registry entries are used for configuring event-logging. Most importantly this registry is used when you want to log 32 bit server and vendor daemon event logs running on 64 bit OS.
- `eventlogremove.reg`—Template for uninstalling event-logging support.
- eventlogremove32on64.reg —Template for uninstalling event-logging support. Most importantly this registry is used when you want to remove the registry entries of 32 bit server and vendor daemon running on 64 bit OS.

These scripts, as shipped, are in a generic form. They reference the generic *<vendorname>* and *<path>* instead of real values. The following procedure suggests a way to incorporate the scripts into your installer.

Task ***To incorporate scripts into your installer***

1. Parameterize all occurrences of *<vendorname>* and *<path>* in both files.
2. To enable license server manager–specific event-logging at installation time, in `eventlogadd.reg` locate the key:

```
"LMGRD_EventLoggingMode"="DISABLE"
```

3. Set the mode to `ENABLE`. Alternatively, provide instructions to your end user for setting this registry key.
4. To enable vendor daemon–specific event-logging at installation time in `eventlogadd.reg`, locate the key:

   ```
   "<vendorname>_EventLoggingMode"="DISABLE"
   ```

5. Set the mode to `ENABLE`. Alternatively, provide instructions to your end user for setting this registry key.
6. At installation time, substitute the parameterized *<vendorname>* and *<path>* occurrences with the actual values at your end user's site.
7. Invoke the scripts as part of your installation to enable event-logging.

Localizing the Event-Logging Message Files

Event-logging message files contain description strings used when logging events to the Windows Event Log. Two message files are shipped in the licensing toolkit, and as shipped, are in English. Localize them as appropriate for your product requirements.

- `VendorLicenseServerMsgs.mc`—Contains description strings for vendor daemon-specific events. Localization recommended.
- `fnplmLicenseServerMsgs.mc`—Contains description strings for license server manager-specific events. Localization optional.

The structure of these files adheres to the syntax set up by Microsoft for Windows message files. Support for additional languages can be added. One message file can be localized to several languages. Consult Microsoft documentation for further details on organizing and localizing your message files.

After localizing the message files, you need to rebuild the current message file DLLs in order for your changes to take effect. The licensing toolkit provides a rule in `i86_n3/makefile` for building these files, using the standard Windows message file compiler, resource compiler, and linker. From the `i86_n3` folder, type the following in a DOS command window:

```
nmake buildmsgfile
```

This command copies the current DLLs to files with the extension `.dll_bak` and generates new DLLs. Consult your Microsoft documentation for further details on compiling message files.

For more information, see the event-logging and message file documentation in the Microsoft MSDN Library (msdn.microsoft.com).

Installing the License File

The are several alternatives for the way in which the FlexEnabled product finds the license file containing the correct license certificate for the feature in question. Some considerations are provided in the following section. For a complete list of steps involved, see Locating the License File in *Programming Reference for License File–Based Licensing*.

Vendor Considerations

It is recommended that the product specify a default license file directory relative to the installed product hierarchy, for example—

`install_path/license_file_path`

—where `install_path` is determined by the end user during installation and `license_file_path` is a directory. At run time, `install_path` is dynamically determined and the default license-file path is used by the checkout mechanism to find the right license.

The license file should be installed with a `.lic` suffix so the product can find it. The product does not need to know the exact name of the file, only that it ends with `.lic` and exists in the specified directory. A useful practice is to name the file with the generation date, in *yyyymmdd* format. For example, if the license is generated on July 10, 2016, it can be installed as:

`install_path``/licenses/20160710.lic`

The symbol @ should not be used in the path to the license file directory. Ensure that your end users have this information.

End User Considerations

Even though the FlexEnabled product specifies the location where it expects to find the license certificate, the end user can override this by setting the `LM_LICENSE_FILE` environment variable with a `license_file_list`, where `license_file_list` can consist of the following components:

- The full path to the license file
- A directory containing one or more license files with a `.lic` extension
- A *`port@host`* setting, where *`port`* and *`host`* are the TCP/IP port number and host name from the `SERVER` line in the license file. Alternatively, use the shortcut specification, *`@host`*, if the license file `SERVER` line uses a default TCP/IP port or specifies a port in the default port range (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 comma-separated list of three *port@host* specifiers denoting a three-server redundant configuration. For example, **port1@host1,port2@host2,port3@host3** specifies the three-server redundant configuration composed of **host1**, **host2**, and **host3**.

The product can also accept an environment variable named *VENDOR*`_LICENSE_FILE`, where *VENDOR* is the vendor daemon name, for example, `DEMO_LICENSE_FILE`. This environment variable's scope is limited to just the product from vendor *VENDOR*.

Note • *Some products do not recognize the `LM_LICENSE_FILE` environment variable. FlexEnabled Java products, in particular, do not recognize it.*

Installing the License Server

If your license lifecycle requires a license server (for counted or served-uncounted license models), you need to provide end users with an installer and instructions for installing either `lmgrd` or `lmadmin` as the license server manager. For more information about installer types and installation considerations, see Providing the License Server Installation Instructions.

What to Tell Your End Users

This section lists implementation decisions that you make which have a possible effect on end-user administration and the FlexEnabled product behavior. These decisions are presented in a worksheet format which can be used as a checklist when talking to your end user. Keywords are presented for each issue along with a references for additional information.

Up-to-date FlexNet Publisher Components

Make sure your end user uses the latest license server manager, `lmadmin` or `lmgrd`, and Licensing utilities, which are available from www.flexerasoftware.com. Additionally, new licensing toolkit enhancements may require your vendor daemon, product, or both to be rebuilt with a compatible version of the Licensing client library and distributed to your end users.

Figure 8-1 summarizes the rules for FlexNet Publisher component compatibility:

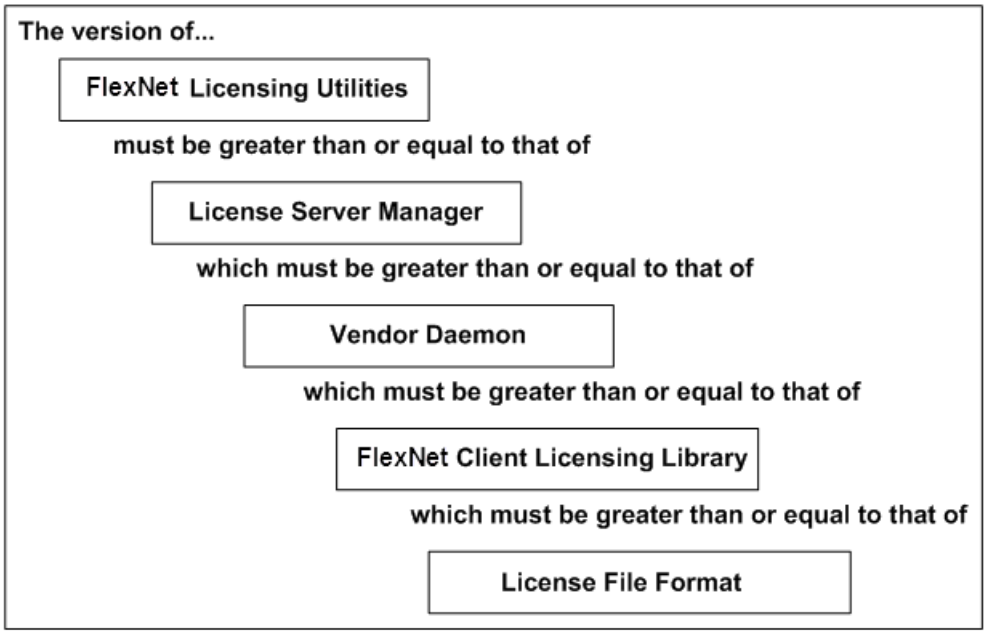

Figure 8-1: FlexNet Licensing Component Version Rules

Note • *`Lmadmin` can be used only with components with a version of 9.2 or higher.*

License Server Need

Notify your end user of your requirement for licenses to be served by a license server. This requirement is dictated by your particular license model.

License Server System Required?	Yes	No

Figure 8-2: License Server Query

See Also
Types of License Files in *Programming Reference for License File–Based Licensing*

Hostid Types

FlexNet Licensing supports architecture-specific hostids, special hostids, and vendor-defined hostids. Supply the end user with information as to which hostids you support for the feature and license server (as applicable) contexts, as well as the preferred method for determining the hostid.

- Which hostids are supported for product features?
- Which hostids are supported for the license server?
- How are hostids determined?

See Also
Hostids in *Programming Reference for License File–Based Licensing*
lc_hostid in *C/C++ Function Reference*

Default License File Location

The product can have an internal default path where it expects to find its license file.

Default license file location defined in the product?	Yes	No
If yes, location?		

Figure 8-3: Default License File Location

See Also
lc_checkout in *C/C++ Function Reference*

License File Environment Variable Enabled

The end user can override the default license file location set internally in the product by setting the `LM_LICENSE_FILE` environment variable. However, you can configure the FlexEnabled product so it does not recognize this variable.

LM_LICENSE_FILE environment variable recognized?	Yes	No

Figure 8-4: License File Environment Variable

See Also
LM_A_DISABLE_ENV in *C/C++ Function Reference*
`LM_LICENSE_FILE` environment variable in the *License Administration Guide*

License File Location in Registry Enabled

By default, the registry is automatically updated with the location of the license file which contains the desired license. You can disable this automatic update.

Automatic registry updating enabled?	Yes	No

See Also
LM_A_CKOUT_INSTALL_LIC in *C/C++ Function Reference*

Reading the Registry

By default, checkouts for features read the registry (Windows) or `$HOME/.flexlmrc` (UNIX), first, to determine the license file location. You can disable this behavior.

Registry reading enabled?	Yes:	No:

See Also
LM_A_APP_DISABLE_CACHE_READ in *C/C++ Function Reference*

License Finder (Windows Only)

On Windows platforms, if the product cannot find the license file, the user is presented with a dialog that asks for the license file location, the name of the system running the license server, or allows the user to fulfill the license through the internet. This functionality is enabled by default, but you can disable it.

License finder enabled?	Yes:	No:

See Also
LM_A_PROMPT_FOR_FILE (Windows Only) in *C/C++ Function Reference*
`FLEXLM_BATCH` environment variable in the *License Administration Guide*

License Denial/Checkout Failure

Several different scenarios exist for handling license request denials. You define the FlexEnabled product behavior in the event of a license request denial. Give the end user details specific to your FlexEnabled product.

How are denials handled?	
Notes:	

See Also
lc_checkout in *C/C++ Function Reference*

License Timeout

This policy sets the time after which an inactive license is reclaimed by the vendor daemon. You set the minimum timeout value in the product and need to notify the end user of the value.

Support for timeout implemented?	**Yes:**	**No:**
Minimum timeout value:		

See Also
ls_minimum_user_timeout in *C/C++ Function Reference*
`TIMEOUT` options file keyword in the *License Administration Guide*

License Lingering

This policy sets the amount of time the vendor daemon holds on to the license after the product checks it in or exits. To enable this functionality, set the linger value in the product.

Support for linger implemented?	**Yes:**	**No:**
Linger value:		

See Also
LM_A_LINGER in *C/C++ Function Reference*

Reconnection Strategy

In the event that daemon failure is detected, you need to implement a strategy in the FlexEnabled product for reconnection. Provide the end user with the details of this strategy.

Product behavior when daemon failure is detected?	
How many retries:	

See Also
Heartbeats in *Programming Reference for License File–Based Licensing*
LM_A_RETRY_COUNT and LM_A_RETRY_INTERVAL in *C/C++ Function Reference*

License Borrowing Interface

If you support license borrowing using the `BORROW` keyword, then you may have implemented a proprietary borrowing interface in the FlexEnabled product. If this is the case, provide the end user with details of the interface.

Borrowing interface implemented?	**Yes:**	**No:**
Usage notes:		

See Also
License Borrowing using the BORROW Keyword in *Programming Reference for License File–Based Licensing*
LM_A_BORROW_EXPIRE in *C/C++ Function Reference*

Early Return of Borrowed Licenses

You need to tell your user if you support the early return of borrowed licenses.

Early return supported?	**Yes:**	**No:**
Usage notes:		

See Also
License Borrowing using the BORROW Keyword in *Programming Reference for License File–Based Licensing*
ls_borrow_return_early in *C/C++ Function Reference*
`lmborrow` utility in the *License Administration Guide*

Duplicate Grouping

Duplicate grouping allows a vendor-defined combination of users, hosts, and displays to be considered in the same group such that license requests from any member only consume one license. You have the option of implementing this internally in the FlexEnabled product or on the feature definition line in the license certificate.

Duplicate grouping implemented?	**Yes:**	**No:**
What are the groups?		

See Also
lc_checkout in *C/C++ Function Reference*
LM_A_CHECKOUT_DATA in *C/C++ Function Reference*
DUP_GROUP in *Programming Reference for License File–Based Licensing*

Host Naming Convention

The FlexEnabled application can refer to the local host name either by its relative distinguished name or its fully qualified domain name (FQDN). This name is sent to the license server for validating host name specifications in the end-user options file.

Fully qualified domain names supported?	**Yes:**	**No:**

See Also
DUP_GROUP in *Programming Reference for License File–Based Licensing*
LM_A_HOST_OVERRIDE in *C/C++ Function Reference*
LM_A_USE_FQDN in *C/C++ Function Reference*

License Server Organization

Internal organization of license jobs within the FlexEnabled product affects the way the end user sets up license servers and the license files on those servers. For served licenses, all features checked out by a given license job need to be served by the same license server. You need to decide the organization of license jobs within the product. Provide your end user with the particulars of this organization before servers are set up at the end-user site.

Are multiple license jobs used?	Yes:	No:
Usage notes:		

See Also

lc_new_job in *C/C++ Function Reference*

Part 4
Other Development Tasks

This part of the Development Environment Guide includes the following chapters:

- Implementing Secure Data Types (SDTs)
- Providing the License Server Installation Instructions
- Setting Up Microsoft Visual Studio for a FlexEnabled Application

9

Implementing Secure Data Types (SDTs)

This chapter describes a method of adding security to a FlexEnabled application by binding it more securely to a prepared run-time component. (A FlexEnabled application to which additional licensing security is implemented is called a *tamper-resistant application*.) This binding uses Secure Data Types (SDTs) to encrypt strategic parameter values (usually constants or variables) within the code for a licensed feature in the application. The SDTs are linked to the licensed feature so that, if the feature's license is properly checked out, the run-time component decrypts the SDTs, allowing them to behave like their normal underlying data types. However, if the checkout is somehow bypassed, the SDTs are not decrypted properly, producing unexpected and random results when their underlying types are involved in computations.

A product-specific run-time component is created and bound to the FlexEnabled application that contains the SDTs. This method provides a defense against static and dynamic patching of the application code. Static patching modifies the executable code when it is not executing (that is, on disc), while dynamic patching modifies the code while it is running.

The run-time component can provide additional security by using Anti-Debugger Technology (ADT). This technology helps prevent the use of debuggers and system-tracing facilities while the executable is running, thereby make dynamic analysis more difficult. ADT operates at the operating-system level and can be turned off when necessary (for example, when products require debuggers). ADT protection is platform-specific, and its level of security varies between platforms.

Note • *SDT functionality described in this chapter is available for both trusted-storage and license file-based licensing. However, this functionality is not available in notr toolkits for either type of licensing.*

Implementation Overview

Implementing SDTs in your application involves two stages:

- Modifying the application—Add calls to load and unload the run-time component; and define and use SDTs in the application. At this stage, the code runs normally and can be debugged.
- Processing the application—Run the `preptool` utility on the binary to enable the use of the run-time component and to implement the SDTs. At this stage, the code runs properly only if valid licenses and the required SDT and run-time components are available.

It is recommended that you modify the application as early as possible in its development. The SDTs should be defined at the time that you are inserting features into your application code (using the Flexible API to insert feature-based license checkout calls). The correct choice and use of SDTs requires careful planning: the level of security they provide depends on how strategic the values you encrypt are in obtaining results from a feature and how difficult these values are to trace in the processing.

FlexNet Publisher supports both C++ and C in creating and implementing SDTs. However, SDTs coded in C++ are easier to implement and provide greater security. For clarity, this chapter provides separate instructions for coding SDTs in C++ and in C. When a product is coded in both C++ and C, SDTs, once defined, can be used in both environments. Ensure that you use the correct implementation and usage for the environment.

Operation of SDTs

An SDT is decrypted only when the feature associated with the SDT has a valid license. If a valid license is not checked out for the feature, subsequent use of any SDTs linked to that feature results in an incorrect decryption.

SDTs operate independently per feature. Therefore, if a user attempts to use a feature without a valid license, only the SDTs for that feature result in breaks. SDTs linked to other features still operate correctly.

The following sections illustrate the operation of SDTs.

SDT Behavior During a Valid Checkout

The following sequence shows how, when a valid license is checked out for feature linked to SDTs, the feature produces expected values:

1. SDTs for a feature are currently not use (that is, the feature is not in use).
2. When the user accesses the feature, a valid license is successfully checked out.
3. The run-time component correctly decrypts the SDTs linked to this feature, allowing the SDTs to run with expected values.
4. The feature license is checked back in.
5. SDTs return to a not-in-use state.

   The sample application `ezcalcsdtplus.exe` demonstrates STD behavior when a valid license is checked out the **-(subtract)** feature:

```
Current value:      0

Enter operator and operand (right hand number) e.g. + 5
  + to add
  - to subtract
  x to multiply
  / to divide
  ^ to raise to power
  a to average
  t to add (behavior of SDTs when valid license checkout circumvented) e.g. t 5
  c to clear
  q to clear

Or, just enter a number  >:-5

Checked out: SUBTRACT
Operation  : 0 - 5 = -5
Checked in : SUBTRACT
EZ Calculator
Current value:      -5
```

The following excerpt from `ezcalcstdplus.cpp` shows the application code that ensures license checkout for the feature (the CheckoutGuard class performs a valid checkout):

```
      void subtract( int value )
      {
          CheckoutGuard holder( m_job, "SUBTRACT" );
             std::cout << "Operation  : " << m_value << " - " <<
  value;
          FeatureSubtract_int32 current( m_value );
          FeatureSubtract_int32 entered( value );
          m_value = current - entered;
             std::cout << " = " << m_value << std::endl;
      }
```

SDT Behavior in the Absence of a Valid Checkout

A license for a feature can be incorrectly or invalidly checked out. For example, an invalid checkout can occur when the instructions to perform a checkout are patched in the activation application. When such a checkout occurs, any SDT linked to that feature operates with incorrectly decrypted results, causing the SDT to return unexpected or random values. The following steps show this sequence:

1. SDTs linked to a feature are currently not in use (that is, the feature is not in use).
2. The user successfully accesses the feature, bypassing license checkout.
3. SDTs linked to this feature do not decrypt correctly, resulting in incorrect or random values.

   The sample application `ezcalcsdtplus.exe` demonstrates SDT behavior when license checkout for the feature is circumvented:

```
Current value:      3

Enter operator and operand (right hand number) e.g. + 5
  + to add
  - to subtract
  x to multiply
  / to divide
  ^ to raise to power
  a to average
  t to add (behavior of SDTs when valid license checkout circumvented) e.g. t 5
  c to clear
  q to clear

Or, just enter a number  >:t7

Checkout   : skipped for ADD
Operation  : 3 + 7 = -1470777420
EZ Calculator
Current value:      -1470777420
```

The following excerpt from `ezcalcstdplus.cpp` shows the application code that bypasses license checkout:

```
        void add_no_checkout( int value )
        {
              std::cout << "Checkout   : skipped for ADD" <<
   std::endl;
              std::cout << "Operation  : " << m_value << " + " <<
   value;
           FeatureAdd_int32 current( m_value );
           FeatureAdd_int32 entered( value );
           m_value = current + entered;
              std::cout << " = " << m_value << std::endl;
        }
```

Independence of Features

SDTs linked to different licensed features operate independently: breaking one SDT does not break SDTs linked to other features. Thus the following sequence produces the specified results:

1. Use of an SDT linked to feature “add”—no decrypt and undefined value for the SDT.
2. Successful checkout of licensed feature “add.”
3. SDTs that are linked to “add” feature do not decrypt and have undefined values.
4. Successful checkout of licensed feature “subtract.”
5. SDTs that are linked to “subtract” feature decrypt correctly.
6. Check-in of licensed feature “subtract.”

Mapping SDTs to Features

SDTs operate correctly only when there is a valid license for the matching feature. The mapping of SDTs to a feature requires the use of a *feature identity* and *feature alias*. The feature identity is an integer that identifies the feature within the SDT code. The feature alias is used within the application code to identify SDT types. The mapping of feature to a feature identity and feature alias is set up in a feature definition header file.

Feature identities:

- Must be unique within an executable.
- Must be within the range defined by the setting for `SDT_ENCODER_USER_LIMIT` in `sdtconfig.h`.

Choosing Constants and Variables as SDTs

The selection of constants and variables as SDTs is made primarily in relation to the function(s) within your application that you want to protect. Choose constants and variables that are critical to the correct operation of the function. You should ensure that the use of SDTs is not so trivial that the hacker can easily write machine code to replace the function or SDT operation.

The protection provided by SDTs operates in a different way for variables and constants. It is important that this difference is understood and that constants are used to obtain maximum security from using SDTs.

When an SDT is used for a constant, the value of the constant is encrypted during the compilation of the source code of the application. When an SDT is used for a variable, the variable value is encrypted during runtime. This is a potential avenue of attack for the hacker as he can see these values in the clear and hence potentially make some deductions about the encryption process. So to improve security, initialize an SDT variable from an SDT constant and perform a minimum of encrypted to plain value conversions.

For similar reasons, SDTs should not be used to protect a value that can be changed by user input and is repeatedly displayed. This allows the hacker to observe the encryption and decryption.

The implementation of SDTs in the code for the sample applications shipped with the your FlexNet Publisher toolkit (for example, `ezcalcsdtplus.cpp` for trusted-storage licensing or `lmflexsdt.c` for license file-based licensing) do not demonstrate effective security practices. They are included to demonstrate how to add SDTs, not to provide strategies for best SDT use cases.

Building and Testing Code That Includes SDTs

This section contains a recommended sequence for building and testing code that includes SDTs. It is strongly recommended that all code that uses SDTs is thoroughly tested when the SDTs are operational to ensure that the functionality that is required is implemented correctly.

1. If this is a build for a new application or a new version of an existing application, update seeds in `sdtconfig.h`.
2. Add SDTs and the required function calls to your code. The following is a typical flow of functionality:
   - **lc_flexinit_property_handle_create()**
   - lc_flexinit_property_handle_set()
   - lc_flexinit()

- lc_new_job()
- lc_set_att()
- **lc_checkout()**
- Insert SDT code (see SDTs for C++ or SDTs for C)
- **lc_checkin()**
- lc_free_job()
- lc_flexinit_cleanup
- lc_flexinit_property_handle_free()

3. Add the appropriate SDT components to your project and link to them (see Table 9-1).
4. Build your code with your debug libraries.
5. Use `preptool` to process your built code with `<debuggable value="true" />` set in the `<preferences>` element of the `preptool` configuration file. This turns off ADT checks and enables you to use debugging tools while testing the operation of the functionality in your code.

   For more information about running `preptool`, see Running preptool on Applications with SDTs.
6. Test the debug version of your code to ensure that all functionality operates correctly.

   Note • *Before commencing any tests in a trusted-storage licensing environment, ensure that the FlexNet Licensing Service has been installed on the system and is running.*
7. Build your code with release libraries.
8. Use `preptool` to process your release code with `<debuggable value="true" />` set in the `<preferences>` element of the `preptool` configuration file.

   This turns off ADT checks so that debuggers can be used during the test cycle.
9. Test the release version of your code to ensure that all functionality operates correctly.

Important • *When using SDTs, note the following:*

- *Ensure that any* **lc_checkout** *call is matched with one call to* **lc_checkin**. *If there are multiple calls to* **lc_checkin,** *then SDTs will fail to operate correctly.*
- *On Windows platforms, do not call* **flxActCommonLibraryCleanup** *or* **flxActCommonLibraryInit** *in* `DllMain`. *These APIs result in calls to the Microsoft Win32 FreeLibrary and LoadLibrary APIs. Calls to these Win32 APIs in* `DllMain` *are widely known to be associated with issues.*
- *The* `preptool` *utility, delivered with the FlexNet Publisher toolkit, is called as such. However, if a publisher provides its own utility, its name might be different.*

Guidelines for Using SDTs

The SDT system converts plain data types into objects, and as a result of this, there are a number of issues that the developer needs to be aware of.

Performance

The data values stored in SDT variables are encrypted, and each operation on the SDT value requires some level of processing of the encrypted data. This processing cycle incurs a performance impact when compared to performing operations on plain data types. Furthermore, the impact on performance increases when the product is protected (due to the greater level of encryption being applied), when compared to the unprotected product using SDTs. Therefore, avoid using SDTs in loops, which can cycle a high number of times.

Conversion Issues

Where unchecked or automatic-type conversions are used in a program (for example, within calls to functions that have variable argument counts such as `printf`), the change from plain type to secure-type object is not detected by the compiler. Therefore, if a variable is converted from a plain type to an SDT, and then used where such an automatic conversion can occur, such as `printf`, undefined behavior results.

Testing

Test SDT-protected variable usage (with the protection active) at regular intervals during development to ensure that performance overhead does not adversely affect the application behavior. For detailed instructions on testing, see Building and Testing Code That Includes SDTs.

Files Provided in the Toolkit for Implementing SDTs

The Table 9-1 lists the files that are provided in the toolkit and are used for implementing SDTs. This table is formatted so the files that you need to specify in your project or to edit are listed at the top of the table and highlighted in green. All other files are required but are included as required, without needing to be specified individually. All files are located in the `machind/sdt` folder.

Table 9-1 • FlexNet Publisher Licensing Toolkit Files for SDTs

File	Description
fnpsdtapi.h	Top-level header file that includes all the header files for SDTs.
sdtdata.c	Component required when inserting SDTs in C code.
sdttype.cpp	Component required when inserting SDTs in C++ code.
sdtconfig.h	Configurable parameters for SDTs. The encoding seeds should be changed for each release of a product.
sdtfeature.h	Functions used to define SDTs.

Table 9-1 • FlexNet Publisher Licensing Toolkit Files for SDTs

File	Description
fnpstdtypes.h	Standard types required for SDTs.
fnpplatformtypes.h	Platform-specific standard types.
sdtbase.h	Required for SDTs.
sdtdata.h	Required for SDTs.
sdtdefault.h	Required for SDTs.
sdtmarker.h	Required for SDTs.
sdtplatform.h	Required for SDTs.
sdttype.h	Required for SDTs.

sdtconfig.h

This file contains the following configuration information required for SDTs:

- **SDT seeds**—Use unique SDT seeds for each product and product release to create product-specific encoding of the SDTs. Edit these seeds before each release build.
- **Encoding**—Use the default `SDT_API_ENCODING_EXCLUDE_NONE`.
- **Plugin**—Use the default `Sdt::CDefaultPlugins`.
- **Encoder limit**—Specifies the maximum number of feature identities that can be defined. The default value is 64. It can be increased by multiples of 64.

Additional Function Calls Needed

When you implement SDTs in your application, you require the minimum **lc_new_job**, **lc_set_attr**, **lc_ checkout**, **lc_checkin**, and **lc_free_job** APIs. However, the application must also call the following initialization functions to load and unload the run-time component.

Note • *In a trusted-storage licensing environment, the run-time component is the same as the activation library.*

Table 9-2 • Initialization Functions

Function Name	Description
lc_flexinit	Loads the prepared run-time component and initializes the FlexNet Publisher Licensing Toolkit.

Table 9-2 • Initialization Functions

Function Name	Description
lc_flexinit_cleanup	Unloads the run-time component and performs any cleanup functions required.
lc_flexinit_property_handle_create	Creates a property handle.
lc_flexinit_property_handle_free	Frees an initialization property handle.
lc_flexinit_property_handle_set	Sets a type and value into the initialization property handle.
lc_flexinit_property_handle_get	Returns the value for a specified property.

The **lc_flexinit** and **lc_flexinit_cleanup** calls must be synchronized with the following:

- **Use of SDTs**—The application must call **lc_flexinit** before any SDTs are used in the code. In C++, this call must be made before construction of SDTs. For example the following code sample will fail because the SDT, `FeaturePrint_int32`, is declared before the call to **lc_flexinit:**

  ```
  FeaturePrint_int32 a(42);
  lc_flexinit();
  ```

- **License check-in**—The application must check in licenses for all features before calling **lc_flexinit_cleanup**.

In multithreaded environments you must ensure synchronization between threads. The following are cases where particular care is needed.

- **Executable exits**—When the FlexEnabled executable exits, the FlexNet code, either in the license server or the FlexEnabled application, automatically causes a check-in of features. This automatic process cannot be used when SDTs are in use. Your code must ensure that the features are checked in and the activation library is unloaded before your code exits.
- **Connection to license server breaks**—When a concurrent license is in use and the connection to the license server breaks, the code using the activation library must check in the licenses.

Note • *The functions performed by* **lc_flexinit** *and* **lc_flexinit_cleanup** *are also performed by the Activation API functions* **flxActCommonLibraryInit** *and* **flxActCommonLibraryCleanup**. *For more details see C/C++ Function Reference.*

Product with Multiple Executables

In the Windows environment, a product with multiple executables is typically an executable (`.exe`) file and DLLs that contain functions called by the executable. On a Mac, a product with multiple executables is typically an executable and bundles or frameworks included within the product package. For UNIX, a product with multiple executables is typically an executable and shared objects. This section refers to a *main executable* and *subsidiary executables*, such as DLLs or shared objects.

Multiple-Executable Scenarios

When SDTs are used, functionality can be split between the executable and one or more subsidiary executables. The following scenarios are possible:

- The check-out and check-in functions are in the main executable, and the SDTs are in subsidiary executables.
- The check-out and check-in functions and some SDTs are in the main executable, and additional SDTs are in subsidiary executables.
- The check-out and check-in functions and SDTs are in the subsidiary executables. No functionality exists in the main executable.

Create a Single Run-Time Component for All Executables

When SDTs are used, the same run-time component must process the check-out and check-in functions and the SDTs. Therefore, if these functions are split among executables, all these executables must access the same run-time component.

Task ***To create a single run-time component for use by all executables***

1. Add **lc_checkout** and **lc_checkin** functions to the appropriate executable.
2. Add SDTs to the appropriate executables.
3. Add **lc_flexinit** and **lc_flexinit_cleanup** to *every* executable that you modified in steps 1 and 2.
4. Run `preptool` to create a single run-time component for all the executables you modified in steps 1 and 2. (To do so, use a `preptool` configuration file that lists all the executables. For an example, see the *Working with the Software Publisher Utilities* in the *Programming Reference for Trusted Storage–Based Licensing*.)
5. Add the run-time component created in step 4 to your product so that it is available to all the executables that require it.

Note • *On Windows platforms, do not call* **flxActCommonLibraryCleanup** *or* **flxActCommonLibraryInit** *in* `DllMain`*. These APIs result in calls to the Microsoft Win32 FreeLibrary and LoadLibrary APIs. Calls to these Win32 APIs in* `DllMain` *are widely known to be associated with issues.*

Running preptool on Applications with SDTs

After you build an application containing SDTs, you must run the `preptool` utility on the code to enable the run-time component and to implement the SDTs.

`preptool` is not backward compatible with older versions of `preparchive*.yaa`. Ensure the version of `preptool` matches the version of `preparchive*.yaa` that is included in the toolkit.

Preparing Applications That Use Trusted-Storage Licensing

For more information about running `preptool` on applications that use trusted-storage licensing, see "Working with Software Publisher Utilities" in the *Programming Reference for Trusted Storage–Based Licensing*. For trusted storage licensing, `preptool` performs tasks in addition to preparing an application for SDT usage.

Preparing Applications That Use License File-Based Licensing

To enable the SDTs defined in an application that uses license file-based licensing, the `preptool` utility both prepares the application and generates a run-time component that handles the SDTs.

For more information about running `preptool`, see "Working with Software Publisher Utilities" in the *Programming Reference for Trusted Storage–Based Licensing*.

Requirements

Before proceeding with the `preptool` process for applications using license file-based licensing, note the following:

- If `lmstrip` needs to be run, do so *before* running `preptool`.
- If the application needs to be digitally signed, do so after you run `preptool`.

Use Cases

Using Trusted-Storage Licensing Toolkits to Implement SDTs in Applications with License File-Based Licensing

For publishers who have purchased a trusted-storage licensing toolkit but want to implement SDTs in an application that uses license file-based licensing, do the following:

1. In the `preptool` configuration file, specify `publisher.xml` as the activation settings file:

   ```
   <activationsettings path="ActivationSettings\publisher.xml" isserver="false"/>
   ```

   The file `publisher.xml` is a full-featured activation settings file issued with the trusted-storage licensing toolkit. However, it is used to prepare applications that use trusted-storage licensing and those that use license file-based licensing and contain SDTs.

2. Run `preptool` using the `-sdtonly` option in the command-line. (Refer to the `makefile` in your toolkit for an actual example of this command run for `lmflexsdt`.)

   ```
   preptool -sdtonly -v <prepconfig>
   ```

   or

   ```
   preptool -t -v <prepconfig>
   ```

   where

   - `-sdtonly` implements the SDTs. (You can substitute `-t` as the switch for -stdonly.) Use this option to implement SDTs in license file-based licensing only.

- *prepconfig* is the name of the `preptool` configuration file that points to `publisher.xml`.

Note • *You can also use the* `-p <preparchive_path>` *or* `-C <publisher_preptool>` *options with the* `preptool` *command. See the "Working with Software Publisher Utilities" chapter in the* Programming Reference for Trusted Storage–Based Licensing *for more information about these options.*

Using Licence File-Based Licensing Toolkits to Implement SDTs in Applications

Publishers who have purchased rights to license file-based licensing only can create applications that use only license file-based licensing. Do the following to run `preptool` on such an application that contains SDTs:

1. In the `preptool` configuration file, specify `sdt_publisher.xml` as the activation settings file:

   ```
   <activationsettings path="ActivationSettings\sdt_publisher.xml" isserver="false"/>
   ```

   The file `sdt_publisher.xml` is the activation settings file reserved for preparing SDTs in applications created with license file-based licensing toolkits only.

   Note • *You cannot use* sdt_publisher.xml *when preparing a file-based licensing application with a trusted-storage licensing toolkit. In this case, you must use* publisher.xml.

2. Run `preptool` using the `-sdtonly` option in the command-line. (Refer to the `makefile` in your toolkit for an actual example of this command run for `lmflexsdt`.)

   ```
   preptool -sdtonly -v <prepconfig>
   ```

   or

   ```
   preptool -t -v <prepconfig>
   ```

   where

   - `-sdtonly` implements the SDTs. (You can substitute `-t` as the switch for `-stdonly`.) Use this option to implement SDTs in license file-based licensing only.
   - *prepconfig* is the name of the `preptool` configuration file that points to `sdt_publisher.xml`.

   Note • *You can also use the* `-p <preparchive_path>` *or* `-C <publisher_preptool>` *options with the* `preptool` *command. See "Working with Software Publisher Utilities" in the Programming Reference for Trusted Storage–Based Licensing for more information about these options.*

SDT Sample Files Included in Toolkits

FlexNet Publisher toolkits provide the following sample files to show how SDTs are built and used in applications.

For Trusted Storage–Based Licensing

Trusted-storage licensing toolkits provide the following sample files to demonstrate how SDTs are built and used in applications that use trusted-storage licensing. The files are classified in two tables as C++- based or C-based examples.

Table 9-3 • Sample Files Using C++

Samples	Toolkit Location	Description
ezcalcsdtplus.exe (Windows) **ezcalcsdtplus (UNIX)**	`<platform_dir>`	Application that was created in C++ language and uses SDTs.
ezcalcsdtplus_libFNP.dll (Windows) **ezcalcsdtplus_libFNP.so (UNIX)**	`<platform_dir>`	Run-time component for `ezcalcsdtplus` created by `preptool`.
ezcalcsdtplus.cpp	`<install_dir>\examples\ezcalc`	Code for the `ezcalcsdtplus` application.
sdtdefs.c **sdtdefs.h**	`<install_dir>\examples\ezcalc`	SDT feature definition header and implementation files.
ezcalcsdtplus.xml **ezcalcsdtplus_unix.xml** **ezcalcsdtplus_mac.xml**	`<install_dir>\examples\ezcalc`	`preptool` configuration files for the `ezcalcsdtplus` application for Windows, UNIX, and Mac platforms, respectively.

Table 9-4 • Sample Files Using C

Samples	Toolkit Location	Description
ezcalcsdt.exe (Windows) **ezcalcsdt (UNIX)**	`<install_dir>\examples\ezcalc`	Application that was created in C language and uses SDTs.
ezcalcsdt_libFNP.dll (Windows) **ezcalcsdt_libFNP.so (UNIX)**	`<platform_dir>`	Run-time component for `ezcalcsdt` created by `preptool`.
ezcalcsdt.c	`<install_dir>\examples\ezcalc`	Code for the `ezcalcsdt` application.
sdtdefs.c **sdtdefs.h**	`<install_dir>\examples\ezcalc`	SDT feature definition header and implementation files.

Table 9-4 • Sample Files Using C

Samples	Toolkit Location	Description
ezcalcsdt.xml **ezcalcsdt_unix.xml** **ezcalcsdt_mac.xml**	`<install_dir>\examples\ezcalc`	`preptool` configuration files for the `ezcalcsdt` application for Windows, UNIX, and Mac platforms, respectively.

For License File-Based Licensing

License file-based licensing toolkits provide the following sample files to show how SDTs are built and used in applications that use license file-based licensing.

Table 9-5 • Sample Files for License File-Based Licensing

Samples	Toolkit Location	Description
lmflexsdt.exe (Windows) **lmflexsdt (UNIX)**	`<platform_dir>`	Application that uses SDTs. ***Note*** • *In order to build and prepare this application, refer the* makefile *in your toolkit. Run the following command from* `<platform_dir>`*:* **`nmake SDT=1 (Windows)`** **`make SDT=1 (UNIX)`**
lmflexsdt_libFNP.dll (Windows) **lmflexsdt_libFNP. so (UNIX)**	`<platform_dir>`	Run-time component for `lmflexsdt` created by `preptool`.
lmflexsdt.c	`<install_dir>\examples\sdt`	Code for the `lmflexsdt` application.
lmflexsdtdefs.c **lmflexsdtdefs.h**	`<install_dir>\examples\sdt`	SDT feature definition header and implementation files.
lmflexsdt_win.xml **lmflexsdt_unix.xml** **lmflexsdt_mac.xml**	`<install_dir>\examples\sdt`	`preptool` configuration files for the `lmflexsdt` application for Windows, UNIX, and Mac platforms, respectively.

SDTs for C++

SDTs used within C++ provide classes that behave like plain integers, but store the actual value of the integer in a coded form. These secure SDT classes behave like the corresponding plain types. The following code fragment, taken from the sample file `ezcalcsdtplus.cpp`, illustrates the use of SDTs within code. Note that it does not include the code required to define or implement the SDT `FeatureDivide_int32`.

Example Using a Plain Type

```
void divide( int value )
    {
        CheckoutGuard holder( m_job, "DIVIDE" );
        int current( m_value );
        int entered( value );
        m_value = current / entered;
    }
```

Example Using the SDT, FeatureDivide_int32

```
void divide( int value )
    {
        CheckoutGuard holder( m_job, "DIVIDE" );
        FeatureDivide_int32 current( m_value );
        FeatureDivide_int32 entered( value );
        m_value = current / entered;
    }
```

The process of adding SDTs to your code is in outline as follows:

- Create a feature definition header file that contains a define for each licensed feature that requires binding to the activation library.
- Create a feature implementation file and add it once to *each* module that uses SDTs.
- Use the SDTs within your code.
- Add `sdttype.cpp` to your project and link your executable to it.

Note • *The activation library must be executing while SDTs are in use otherwise the SDTs do not operate correctly. See Additional Function Calls Needed for details.*

Creating the Feature Definition Header File

Each feature that you want to protect by binding it to the activation library must be defined in a header file. The following code fragment is an example taken from the feature definition header file, `sdtdefs.h`, included in the toolkit. This feature definition header file is used by the example FlexEnabled application `ezcalcsdtplus.cpp`.

```
#include "fnpsdtapi.h"

SDT_FEATURE_DEFINE( "ADD", 0, FeatureAdd );
SDT_FEATURE_DEFINE( "SUBTRACT", 1, FeatureSubtract );
SDT_FEATURE_DEFINE( "MULTIPLY", 2, FeatureMultiply );
```

```
SDT_FEATURE_DEFINE( "DIVIDE", 3, FeatureDivide );
SDT_FEATURE_DEFINE( "POWER", 4, FeaturePower );
SDT_FEATUR_DEFINE( “AVG”, 5, FeatureAvg );
```

This defines a feature identity and feature alias for each of the six licensed features in the `ezcalc` application. These license features are in the application, for example the `ADD` feature is used in the **add** function as shown in this code fragment from `ezcalcsdtplus.cpp`:

```
void add( int value )
    {
        CheckoutGuard holder( m_job, "ADD" );
    }
```

Note • *The feature identity to feature mapping defined in a feature definition header file is specific to the modules that the feature definition header file is used with. Thus feature identities can be used in other feature definition header files to apply to other features. Also, different feature identities can be mapped to the same feature in separate feature definition header files.*

Creating the Feature Implementation File

The SDTs defined in the header file created in the previous section must be implemented by being added to an implementation file. The following code is an example taken from the feature implementation file, `sdtdefs.c`, included in the toolkit. This feature implementation file is used by the example FlexEnabled application `ezcalcsdtplus.cpp`.

```
#include "sdtdefs.h"

SDT_FEATURE_IMPLEMENT_BEGIN()
    SDT_FEATURE_IMPLEMENT_C( FeatureAdd, "ADD" )
    SDT_FEATURE_IMPLEMENT_C( FeatureSubtract, "SUBTRACT" )
    SDT_FEATURE_IMPLEMENT_C( FeatureMultiply, "MULTIPLY" )
    SDT_FEATURE_IMPLEMENT_C( FeatureDivide, "DIVIDE" )
    SDT_FEATURE_IMPLEMENT_C( FeaturePower, "POWER" )
    SDT_FEATURE_IMPLEMENT_C( FeatureAvg, "AVG" )
SDT_FEATURE_IMPLEMENT_END()
```

Each of the license features is separately implemented using the feature alias and feature name as parameters. Note that this example file uses the C macro: it would be more common to use the C++ macro, SDT_FEATURE_IMPLEMENT.

SDTs operate independently in each executable or dynamic library to which they are added. Thus an implementation file must be added to each module in which they are used and each module must be separately processed using the `preptool`.

Using SDTs in Code

Task ***To use SDTs in your code:***

1. Modify your code:
   a. Include the feature definition header file.
   b. Include the feature implementation file. A feature implementation file must be added to *each* module that uses SDTs

c. Insert a call to **lc_flexinit** in your code so that it loads the activation library before any code that constructs or uses SDTs is executed.

d. Use the secure SDT classes instead of plain types for crucial parameters.

e. Insert a call to **lc_flexinit_cleanup** in your code so that it unloads the activation library once SDTs are no longer being used.

2. Test your code.

   You can use debuggers at this stage to test the operation of your code. The SDTs inserted in the code are not operational.

3. Process the compiled executable with the `preptool`.

   You can only use debuggers at this stage if the your settings for the activation library allow them. The SDTs inserted in the code are operational.

For each feature defined in the feature definition header file a set of secure classes that use the following naming convention are available for use:

`FeatureAlias_type`

Thus the following feature definition and feature implement lines allow the use of the SDTs `FeatureDivide_int32`, `FeatureDivide_uint32`, and other SDTs:

```
SDT_FEATURE_DEFINE( "DIVIDE", 3, FeatureDivide );
SDT_FEATURE_IMPLEMENT_C( FeatureDivide, "DIVIDE" )
```

Table 9-6 lists all the types that are supported:

Table 9-6 • C++ Secure Types

Type	Equivalent to
`int8`	8-bit signed integer
`uint8`	8-bit unsigned integer
`int16`	16-bit signed integer
`uint16`	16-bit unsigned integer
`int32`	32-bit signed integer
`uint32`	32-bit unsigned integer

The following operations are supported on C++ SDTs: =, +=, -=, *=, /=, %=, &=,\=, ^=, ..=, ,,=, +, -, *, /, %, &, |, ^, <<, >>, <, <=, = =, >=, >, !=, ++ (pre-increment), -- (pre-decrement), ++ (post-increment), -- (post-decrement).

C++ SDT Macro Reference

The following describes the macros used to define SDTs in your code. For an example of SDTs within code, see `ezcalcsdtplus.cpp` in `<install_dir>\examples\ezcalc`.

SDT_FEATURE_DEFINE

This macro defines a set of C++ classes that can be used instead of the equivalent plain types.

`SDT_FEATURE_DEFINE(`*`Name, FeatureIdentity, FeatureAlias`*`)`

Parameters

Name

The feature name as used by the Flexible API, that is, `lc_checkout(job, feature, version, num_lic, flag, code, dup_group)`.

FeatureIdentity

An integer identity used within the activation library to identify the feature to which this SDT is mapped. It must be unique within the executable and in the range `0` (zero) to one less than the maximum defined in `sdtconfig.h` (`SDT_ENCODER_USER_LIMIT - 1`).

FeatureAlias

An alias by which the SDTs for this feature are referenced when using the SDTs within the application code.

Discussion

To use this macro you must include the following header file:

```
machind/sdt/sdtfeature.h
```

This macro is used within the feature definition header file to define each feature that you want to bind to the activation library. It creates a set of C++ classes that can be used instead of the equivalent plain types for processes that are crucial to the operation of the feature. After the SDT implementation is completed, any process that uses the secure C++ classes completes successfully only when a valid license is available; that is, a checkout request for the feature has completed successfully and a check-in request has not occurred.

See Table 9-6 for a list of the C++ classes.

Note • *This macro must be used in conjunction with other macros: for a code example that demonstrates this see SDTs for C.*

SDT_FEATURE_DEFINE_CONST

This macro encrypts a literal constant value as an SDT.

`SDT_FEATURE_DEFINE_CONST(`*`FeatureAlias, Value`*`)`

Parameters

FeatureAlias

An alias by which the SDTs for this feature are referenced when using the SDTs within the application code.

Value

The literal constant value to be encrypted.

Discussion

To use this macro you must:

- Include `sdtfeature.h`.
- Define the SDT class used for this value (SDT_FEATURE_DEFINE).

This macro is used in the application code to encrypt critical constants and increase the security of the code in which it is used. SDT constants provide stronger security because the value is encrypted at compile time and the plain value is only decrypted when it is used at runtime.

Note • *This macro must be used in conjunction with other macros: for a code example that demonstrates this see SDTs for C.*

SDT_FEATURE_GET_NAME

This macro returns the feature name corresponding to the alias used when defining an SDT.

`SDT_FEATURE_GET_NAME( FeatureAlias)`

Parameters

FeatureAlias

An alias by which the SDTs for this feature are referenced when using the SDTs within the application code.

Discussion

Use this macro for diagnosing problems.

SDT_FEATURE_IMPLEMENT

This macro implements SDTs that are linked to a licensed feature.

`SDT_FEATURE_IMPLEMENT( FeatureAlias)`

Parameters

FeatureAlias

An alias by which the SDTs for this feature are referenced when using the SDTs within the application code.

Discussion

To use this macro you must:

- Declare the SDT in the feature definition header file (SDT_FEATURE_DEFINE).

- Include the feature definition header file in the module where this macro is being used.
- Mark the ends of the list of declarations (SDT_FEATURE_IMPLEMENT_BEGIN and SDT_FEATURE_IMPLEMENT_END).

This macro is used within the feature implementation file to implement SDTs linked to a licensed feature. The licensed feature is identified by the alias used in the SDT declaration. Each declaration in the feature definition header file must have a matching implement feature macro in the feature implementation file. The implementation file must be added to each module in which the SDTs are used.

Note • *This macro must be used in conjunction with other macros. For a code example that demonstrates this, see the file ezcalcsdtplus.cpp in* `<install_dir>\examples\ezcalc`.

SDT_FEATURE_IMPLEMENT_BEGIN

This macro marks the beginning of a list of entries that implement SDT features.

```
SDT_FEATURE_IMPLEMENT_BEGIN( )
```

Discussion

This macro is used in the feature implementation file to mark the beginning of a list of entries that implement SDT features defined in feature definition header file.

Note • *This macro must be used in conjunction with other macros: for a code example that demonstrates this see SDTs for C.*

SDT_FEATURE_IMPLEMENT_END

This macro marks the end of a list of entries that implement SDT features.

```
SDT_FEATURE_IMPLEMENT_END( )
```

Discussion

This macro is used in the feature implementation file to mark the end of a list of entries that implement SDT features defined in an feature definition header file.

Note • *This macro must be used in conjunction with other macros. For a code example that demonstrates this, see* `<install_dir>\examples\ezcalc\ezcalcsdtplus.cpp`.

SDTs for C

SDTs used within C provide C developers with a sparse version of the automatic protection provided by the SDT classes available in C++. The use of SDT within C is more complex to implement than in C++: an alternative is to use C++ SDT classes and place C functions into C++ compilations units.

SDTs within C provide a data structure that contains the encrypted SDT information and a set of macros that operate on the SDT values.

The process of adding SDTs to your code is in outline as follows:

1. Create a header file that contains a define for each licensed feature that requires binding to the activation library.
2. Create an feature implementation file and add it once to each module.
3. Use the SDTs within your code.
4. Add `sdtdata.c` to your project and link your executable to it.

Note • *The activation library (run-time component) must be executing while SDTs are in use otherwise the SDTs do not operate correctly. See Additional Function Calls Needed for details.*

Creating a Feature Definition Header File

Each licensed feature that you want to protect by binding it to the activation library must be defined in a header file.

Creating a Feature Implementation File

The SDTs defined in the header file created in the previous step must be implemented by being added to an implementation file.

SDTs operate independently in each executable or dynamic library to which they are added. Thus, an implementation file must be added to each module in which they are used and each module must be separately processed using `preptool`.

Using SDTs in Code

Task ***To use SDTs in your code:***

1. Modify your code:
   a. Include the feature definition header file.
   b. Include the feature implementation file.
   c. Insert a call to **lc_flexinit** in your code so that it loads the activation library before any code that uses SDTs is run.
   d. Use the secure SDT classes instead of plain types.
   e. Insert a call to **lc_flexinit_cleanup** in your code so that it unloads the activation library once SDTs are no longer being used.
2. Test your code.

   You can use a debugger at this stage to test the operation of your code. The SDTs inserted in the code are not operational.

3. Process the compiled executable with the `preptool`. For more information about `preptool`, see Running preptool on Applications with SDTs.

   You can only use a debugger at this stage if the your settings for the activation library allow them. The SDTs inserted in the code are operational.

C SDT Macro Reference

SDT_FEATURE_DECLARE_VAR

This macro declares a secure variable.

`SDT_FEATURE_DECLARE_VAR( FeatureAlias, VariableName)`

Parameters

`FeatureAlias`

An alias by which the SDTs for this feature are referenced when defining the SDT variables used within the application code.

`VariableName`

The variable name for the secure variable.

Discussion

Task ***To use this macro:***

1. Include `sdtfeature.h`.
2. Define the feature alias and identity (SDT_FEATURE_DEFINE) in the feature definition header file

This macro is used within the feature definition header file to declare secure variables.

Task ***To use a secure variable in your code you must:***

1. Include the feature definition header file in your module.
2. Create an feature implementation file and include it in your module.
3. Insert a call in your code that loads the activation library.
4. Process the compiled executable with the `preptool`.
5. Use the following macros to manipulate the secure variables:
   - SDT_FEATURE_SET
   - SDT_FEATURE_GET
   - SDT_FEATURE_SET_CONST

- SDT_FEATURE_*operator_name*

After the implementation of the SDTs is completed, any process that uses a secure variable completes successfully only when a valid license is available, that is, a checkout request for the feature has completed successfully and a check-in request has not occurred).

SDT_FEATURE_DEFINE

This macro defines a secure variable that can be used instead of the equivalent plain type.

`SDT_FEATURE_DEFINE( Name, FeatureIdentity, FeatureAlias)`

Parameters

Name

The feature name as used by the Flexible API (for example, `lc_checkout(job, feature, version, num_lic, flag, code, dup_group)`.

FeatureIdentity

An integer identity used within the activation library to identify the feature to which this SDT is mapped. It must be unique within the executable and in the range 0 (zero) to one less than the maximum defined in `sdtconfig.h` (`SDT_ENCODER_USER_LIMIT - 1`)

FeatureAlias

An alias by which the SDTs for this feature are referenced when using the SDTs within the application code.

Discussion

To use this macro, include `sdtfeature.h`.

This macro is used within the feature definition header file to define each licensed feature that you want to bind to the activation library. It is used in conjunction with SDT_FEATURE_DECLARE_VAR to declare secure variables that use the secure long type instead of an ordinary long type for processes that are crucial to the operation of the licensed feature. You must use the following macros to manipulate the secure variables:

- SDT_FEATURE_SET
- SDT_FEATURE_GET
- SDT_FEATURE_SET_CONST
- SDT_FEATURE_*operator_name*

After the implementation of the SDTs is completed (including the use of the `preptool`), any process that uses a secure variable completes successfully only when a valid license is available (a checkout request for the feature has completed successfully and a check-in has not occurred).

SDT_FEATURE_GET

This macro returns the plain (decrypted) value of a SDT.

`SDT_FEATURE_SET( VariableName)`

Parameters

VariableName

The SDT variable name as defined using `SDT_FEATURE_DECLARE_VAR`.

Discussion

This macro can only be used with secure variables that have been declared using `SDT_FEATURE_DECLARE_VAR`.

SDT_FEATURE_IMPLEMENT_BEGIN

This macro marks the beginning of a list of entries that implement SDT features.

```
SDT_FEATURE_IMPLEMENT_BEGIN( )
```

Discussion

This macro is used in the feature implementation file to mark the beginning of a list of entries that implement SDT features defined in feature definition header file.

SDT_FEATURE_IMPLEMENT_C

This macro implements SDTs that are linked to a licensed feature.

```
SDT_FEATURE_IMPLEMENT_C( FeatureAlias, Name)
```

Parameters

FeatureAlias

An alias by which the SDTs for this feature are referenced when defining the SDT variables used within the application code.

Name

The feature name as used by the Flexible API, `lc_checkout(job, feature, version, num_lic, flag, code, dup_group)`.

Discussion

Task ***To use this macro:***

1. Declare the SDT in the feature definition header file (SDT_FEATURE_DEFINE).
2. Include the feature definition header file in the module where this macro is being used.
3. Mark the ends of the list of implemented features (SDT_FEATURE_IMPLEMENT_BEGIN and SDT_FEATURE_IMPLEMENT_END).

This macro is used within the feature implementation file to implement SDTs linked to a licensed feature. The licensed feature is identified by the alias used in the SDT declaration. Each declaration in the feature definition header file must have a matching implement feature macro in the feature implementation file. The implementation file must be added to each module in which the SDTs are used.

SDT_FEATURE_IMPLEMENT_END

This macro marks the end of a list of entries that implement SDT features.

```
SDT_FEATURE_IMPLEMENT_END( )
```

Discussion

This macro is used in the feature implementation file to mark the end of a list of entries that implement SDT features defined in an feature definition header file.

SDT_FEATURE_*operator_name*

This set of macros are used to perform operations on secure variables.

SDT_FEATURE_*operator_name*(*VariableName*, *ParameterName*)

Parameters

operator_name

The name of the operation to be performed. It can be either a numeric operator or a logical operator.

Numeric operators: ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, SHL, SHR
Logical operators: LT, LTE, EQ, GTE, GT

VariableName

The name of the secure variable that contains the data value operated on.

ParameterName

The name of the secure variable that contains the parameter used in the operation.

Discussion

These macros can only be used with secure variables. See SDT_FEATURE_DECLARE_VAR for details of how to declare and use secure variables.

SDT_FEATURE_SET

This macro sets the value of a secure variable.

SDT_FEATURE_SET(*VariableName*, *Value*)

Parameters

VariableName

The SDT variable name as defined using SDT_FEATURE_DECLARE_VAR.

Value

The name of the plain data variable from which the value is obtained or the value itself.

Discussion

This macro can only be used with secure variables that have been declared using SDT_FEATURE_DECLARE_VAR.

SDT_FEATURE_SET_CONST

This macro sets a constant value for a secure variable.

SDT_FEATURE_SET(*VariableName, Value*)

Parameters

VariableName

The SDT variable name as defined using SDT_FEATURE_DECLARE_VAR.

Value

The name of the plain data variable from which the constant value is obtained or the constant value itself.

Discussion

This macro can only be used with secure variables that have been declared using SDT_FEATURE_DECLARE_VAR.

10

Providing the License Server Installation Instructions

If your license lifecycle requires a license server (for counted or served-uncounted license models), you need to supply instructions for installing the license server. You can have your end user install either `lmadmin` or `lmgrd` as their license server manager. The following chapter provides methods and considerations for this installation.

This chapter is split into the following sections:

- Installing the License Server lmadmin
- Installing the License Server lmgrd

Installing the License Server lmadmin

Provide the end user with an `lmadmin` installer in one of these formats:

- Downloaded lmadmin Installer
- lmadmin Installer Wrapped in Your Own Installation Program
- Your Own lmadmin Installer

The following sections outline considerations for providing each of these `lmadmin` installer types. However, for details on how to start up and manage the `lmadmin` license server, refer to the *License Administration Guide*.

Important Notes About lmadmin Installation

Note the following:

- The `lmadmin` installer is based on InstallAnywhere, which is a Java-based program. Therefore, to run the installer, a JRE must be installed. If the JRE is not already present on the machine, the end user must install it separately, because it is not bundled with the `lmadmin` installer. For the required minimum version, refer to the *Release Notes*, section *System Requirements for lmadmin*.

***Note** • InstallAnywhere is the leading multi-platform development solution that delivers a professional and consistent cross-platform installation experience for desktop and server applications.*

- To run `lmadmin` on Windows platforms, the relevant Microsoft Visual C++ Redistributable must be installed. The `lmadmin` installer provides the option to install this package during the installation process. For the required minimum version, refer to the *Release Notes*, section *System Requirements for lmadmin*.
- On Windows platforms, it is best practice to install `lmadmin` and the vendor daemon to a subdirectory of `ProgramFiles (x86)` for the 32-bit `lmadmin` or `ProgramFiles` for the 64-bit `lmadmin` . The default path is `C:\Program Files (x86)\FlexNet Publisher License Server Manager` for the 32-bit `lmadmin` and `C:\Program Files\FlexNet Publisher 64-bit License Server Manager` for the 64-bit `lmadmin`. This location is also referred to as *<lmadmin_install_dir>*.

  Data written by `lmadmin` at runtime should be written to a subdirectory of `ProgramData` (in folders `cache`, `conf`, `licenses`, and `logs`). The default path is `C:\ProgramData\FLEXlm\lmadmin\`. This location is also referred to as *<lmadmin_runtimedata_dir>*.

  (Windows services typically have execute permissions only against applications in a `Program Files` subdirectory, and write access only to `Program Data`.)
- On Linux platforms, users should not install `lmadmin` under root directories or run `lmadmin` as Root.
- Do not run the 32-bit `lmadmin` and the 64-bit `lmadmin` on the same system. If you are upgrading from the 32-bit `lmadmin` to the 64-bit `lmadmin`, you must stop and deinstall the 32-bit `lmadmin` first.

Downloaded lmadmin Installer

The simplest method of providing an `lmadmin` installer is to have end users download and run the appropriate `lmadmin` installer from the FlexNet Publisher website. This program provides a user-interface configured with pre-filled defaults that the user can accept or modify as needed to meet their site's requirements. However, the end user can also run this downloaded installer in console mode, described next.

Option to Run the Installer in Console Mode

On AIX, Linux, Solaris, or Windows platforms, end users can run the `lmadmin` installer in console mode. This mode enables them to install `lmadmin` from the command line, providing an alternative to the user-interface-based method.

The following command runs the installer in console mode:

```
<lmadmin_installer> -i console
```

where `<lmadmin_installer>` is the name of the `lmadmin` installation program.

The console installation process prompts the user for information needed to complete the installation.

lmadmin Installer Wrapped in Your Own Installation Program

Another option is to wrap the `lmadmin` installer within your own installation program, giving you some control over the installation. For example, you can run the installer in user-interface or in silent mode and customize installation options.

Note • *End users must have Java Runtime Environment installed to run your installation program. See the previous section, Important Notes About lmadmin Installation.*

User-Interface-Based Installation

When you wrap the `lmadmin` installer in your own installation program, you can run the installer with the standard user-interface defaults; or you can run it with a properties file or command-line variables that define your own user-interface defaults. While the end user can still modify the default values in the installer's user interface, you can make the installation process easier for a specific user by providing defaults that the user is likely to accept.

Use a Properties File to Customize User-Interface Defaults

If you have several `lmadmin` installer defaults to modify, run the installer with a properties file that defines these changes.

Task ***To run the lmadmin installer with a properties file to customize user-interface defaults***

1. Create a properties file as described in Properties File for the lmadmin Installer.
2. Edit the properties in the file as needed.
3. Within your installation program, use a command similar to this example to run the `lmadmin` installer:

   ```
   lmadmin-x64_n6-<version>.exe -f lmadmin_inst_<version>.properties
   ```

   where

   - `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
   - `-f` is the switch that tells the installation to read from the specified properties file.
   - `lmadmin_inst_<version>.properties` is the name of the properties file.

Use Variables to Customize User-Interface Defaults

If you have one or few installation defaults to modify, simply include the variables for those properties in the execution command line. For a description of useful command-line installation variables, see the later section, lmadmin Command-Line Installation Variables.

To run the `lmadmin` installer with variables to customize pre-filled values in the user interface, use a command similar to this example:

```
lmadmin-x64_n6-<version>.exe -DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License
Server Manager"
```

where

- `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
- `-DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License Server Manager"` is the variable that redefines an option value (in this case, the location of the installation directory on a Windows platform). You can specify multiple variables. (Substitute `"c:\Program Files (x86)"` if you are using the 32-bit `lmadmin` installer.)

Silent Installation

When you wrap the `lmadmin` installer in your own installation program, you have the option to run the installer in silent mode with or without a properties file or command-line variables that configure the installation. Because the silent-mode installation involves no user interaction, you are assured that installation options are set according to your specifications. However, having no user interaction can cause some problems as well. The following procedures and examples show you how to run the installer in silent mode, how to customize options during silent installations, and how to work around certain issues encountered in silent installations.

Use Standard Defaults in Silent Mode

To run the installer in silent mode, using standard FlexNet Publisher defaults, use a command similar to one of these examples:

`lmadmin-x64_n6-<version>.exe -i silent` (Windows 64-bit)
`lmadmin-x64_lsb-<version>.bin -i silent` (Linux 64-bit)
`lmadmin-sun64_u10-<version>.bin -i silent` (Sun 64-bit)

where

- `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
- `-i silent` is the switch that tells the installer to run in silent mode.

Use a Properties File to Customize Installation Options in Silent Mode

If you have several `lmadmin` installer defaults to modify, run the installer with a properties file that defines these changes.

Task ***To run the lmadmin installer in silent mode with a properties file to customize user-interface defaults***

1. Create a properties file as described in Properties File for the lmadmin Installer.
2. Edit the properties in the file as needed.
3. Within your installation program, use a command similar to this example to run the `lmadmin` installer:

   ```
   lmadmin-x64_n6-<version>.exe -f lmadmin_inst_<version>.properties -i silent
   ```

   where

   - `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
   - `-f` is the switch that tells the silent installation to read from the specified properties file.
   - `lmadmin_inst_<version>.properties` is the name of the properties file.
   - `-i silent` is the switch that tells the installer to run in silent mode.

Use Variables to Customize Installation Options in Silent Mode

If you have one or few installation options to modify, simply include the `lmadmin` installation variables for these options in the execution command line. For a description of useful command-line installation variables, see the later section, lmadmin Command-Line Installation Variables.

To run the FlexNet `lmadmin` installer with variables to customize installation options, use a command similar to this example:

```
lmadmin-x64_n6-<version>.exe -DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License
Server Manager" -i silent
```

The following information is used in the command:

- `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
- `-DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License Server Manager"` is the variable that redefines an option value (in this case, the location of the installation directory on a Windows platform). You can specify multiple variables.
- `-i silent` is the switch that tells the installer to run in silent mode.

Ensure Availability of the Log File Should the Silent Installation Fail

A silent installation fails to complete if the end user does not have permissions to create a folder in the default or given path. However, because no files are installed in this situation, the user has no way to view the log file to determine the reason for the installation failure. Therefore, to ensure that you can always review the log file, change its default location to one to which the end user has access.

Important • *The default location for the installation log file is in the* `/tmp/uninstall/Logs` *directory for non-Windows platforms and directly under* `C:\` *on Windows platforms. (In releases prior to 11.10.1, the log file was created directly in the* `lmadmin` *installation directory.) Be sure to inform your end user of the exact location of the installation log file.*

To change the log-file location, use the `-DINSTALL_LOG_DESTINATION` variable in the command line, as in this example:

```
lmadmin-x64_n6-<version>.exe -DINSTALL_LOG_DESTINATION=c:\ -i silent
```

You can also change the log-file name using the `-DINSTALL_LOG_NAME` variable in the command line, as in this example:

```
lmadmin-x64_n6-<version>.exe -DINSTALL_LOG_DESTINATION=c:\
-DINSTALL_LOG_NAME=installer_debug.txt -i silent
```

For a description of these variables and their default values, see the later section, lmadmin Command-Line Installation Variables.

Note • *You can also update the INSTALL_LOG_DESTINATION and INSTALL_LOG_NAME options in the properties file; then run the silent installation with the properties file.*

Ensure That Previous Installation Is Not Over-Written During Silent Installation

No user interaction occurs during a silent installation. Consequently, a silent installation provides no prompt that allows the end user to specify an alternate installation location should the silent-installation location already contain a previous version of `lmadmin`. The silent installation simply overwrites the existing files. If you do not want to overwrite a previous installation, use the `-DUSER_INSTALL_DIR` variable to change the location of the new installation, as shown in this example:

```
lmadmin-x64_n6-<version>.exe -DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License Server Manager <version>" -i silent
```

For a description of this variable and its default value, see the later section, lmadmin Command-Line Installation Variables.

Note • *You can also update the* `USER_INSTALL_DIR` *option in the properties file; then run the silent installation with the properties file.*

Your Own lmadmin Installer

The third option for providing an `lmadmin` installer is to create your own installer. When you do this, you can wrap command-line `lmadmin` options within your program to enable end users to customize the installation. For a description of common options to include in your `lmadmin` installer, see Command-Line lmadmin Options for Installers.

Properties File for the lmadmin Installer

The installation properties file contains the values for options in the `lmadmin` installer user interface. When installing `lmadmin` from the command line, you can use the properties file to set up new pre-filled defaults (which are user-modifiable) in the installer user interface or, for a silent installation, to define the installation options.

To create a properties file, use a command similar to this example. This command installs `lmadmin` and saves the option values you specify in the installer user interface to a properties file whose path and name you define in the command:

```
lmadmin-x64_n6-<version>.exe -r c:\lmadmin_inst_properties\lmadmin_inst_<version>.properties
```

where

- `lmadmin-x64_n6-<version>.exe` (or appropriate platform-related name) is the installer name.
- `-r` is the switch that creates a properties file from the options you are about to specify in the user interface of the `lmadmin` user interface.
- `c:\lmadmin_inst_properties\lmadmin_inst_<version>.properties` is the complete path and file name of the properties file.

Note • *When creating the properties file, you must specify a complete path and file name for the file. Additionally, the extension must be* `.properties`.

lmadmin Command-Line Installation Variables

When wrapping the `lmadmin` installer in your own installation program, you can use command-line variables (instead of creating or updating a properties file) to customize individual installation options. For the regular user-interface version of the installer, use these variables to define new pre-filled (but user-modifiable) values in the installer interface. For a silent installation, use these variables to define installation options.

Format of Variables

In the execution command for the `lmadmin` installer, format each variable as *`VARIABLE=value`*, as shown in this example:

```
lmadmin-x64_n6-<version>.exe -DUSER_INSTALL_DIR="c:\Program Files\FlexNet Publisher 64-bit License
Server Manager" -DINSTALL_LOG_DESTINATION=C:\
```

Common Variables for the lmadmin Installer

The following tables lists the common variables that you can include in the command line for running the `lmadmin` installer:

Table 10-1 • Command-Line Variables for lmadmin Installer

Variable	Description
-DUSER_INSTALL_DIR	Enter a new path in which to install the license server. On Windows, the default is `C:\Program Files\FlexNet Publisher 64-bit License Server Manager` (on 64-bit systems). ***Important*** • *For a silent installation, you must enter an absolute path for this value.* ***Note*** • *If a silent installation is laying down files in the same location as a previous installation, it automatically writes over existing files. To avoid overwritten files, use this variable to install the files in a different location.*
-DUSER_RUN_TIME_ FOLDER_PATH	(For Windows only) Enter a new path in which to install the run-time folders. By default, `lmadmin` folders that are updated during run time should be installed under `ProgramData`, to allow the `lmadmin` service, which has Local Service privileges and is of start-up type “Automatic”, to modify these files. The default is `C:\ProgramData\FLEXlm\lmadmin`.

Table 10-1 • Command-Line Variables for lmadmin Installer

Variable	Description
-DINSTALL_LOG_DESTINATION	Enter a new path for the installation log file. **Non-Windows platforms** By default, the installation process generates this file in the `/tmp/uninstall/Logs` directory. **Windows platforms** In a default Windows installation, the installation process generates this file directly under `C:\`. Especially if you are providing a silent `lmadmin` installation, use this variable to place the log file in an administrator-accessible location different from the installation directory. In this way, the License Administrator always has access to the log to determine what went wrong with the installation. ***Note*** • *Prior to* `lmadmin` *11.10.1, the installation log file was created directly under the* `lmadmin` *installation directory.*
-DINSTALL_LOG_NAME	Enter a new name for the log file. By default, the name is `FlexNet_Publisher_License_Server_Manager_Install_MM_dd_yyyy_hh_mm_ss.log` on non-Windows platforms and `installer_debug.txt` on Windows platforms.
-DPREVIOUS_INSTALLATION_PATH	Enter the absolute path of the installation directory containing the data that you want to import to this installation. In non-Windows installations and default installations of `lmadmin` before release 11.14.1.1 on Windows, this includes vendor daemons, license files, and configuration information. In default installation of `lmadmin` version 11.14.1.1 or later, this directory contains `lmadmin.exe` and its related DLLs, and vendor daemons. It does not contain data that is modifiable during run time (such as configuration information, licenses, or logs).
-DUSER_PREV_INSTL_RUN_TIME_PATH	(For Windows only) Enter the absolute path of the run-time directory containing the files that are modifiable during run time (the `cache`, `conf`, `logs`, and `licenses` folders) you want to import to this installation.
-DRUN_SERVICE	(For Windows only) Enter **1** to run the license server as a service. The default is 0 (to not run the server as a service).
-DSERVICE_NAME	(For Windows only) Enter a new name for the service. The default name is `lmadmin`.

Table 10-1 • Command-Line Variables for lmadmin Installer

Variable	Description
-DDELAY_SEC	(For Windows only) Enter the number of seconds to delay between the time you start the service and the time it actually begins running. This delay is helpful when a FlexNet ID dongle is used to lock the license server to a machine (that is, when the FLEXID is used on the SERVER line). The license server can sometimes fail to start when a system reboots because the license server loads before the dongle device driver has a chance to load properly.
-DHTTP_PORT_NUMBER	Enter the TCP/IP port that the Web server uses to listen for communication with clients who are connecting with the `lmadmin` interface. By default, this value is `8090`.
-DLIC_PORT_NUMBER	Provide the license server manager port. To set a specific port, enter a positive integer. To have FlexNet Publisher assign a default port, enter **`0`** (zero). The default port is always the first available value in the range of 27000 through 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.*
-DRESTART_RETRIES	Specifies the maximum number of times that `lmadmin` attempts to restart the vendor daemon if the vendor daemon goes down. Default value is 10 times.
-DCACHE_TIMEOUT	Specifies the time out value in seconds to update the count of activated licenses to be displayed on web UI dashboard. Default value is 600 (seconds). Allowed range is 30 to 5400 seconds.
-DIMPORT_INSTALL_1= ”Import Installation files”	Specifies whether you want to import configuration from an existing `lmadmin` installation. If yes, specify “Import Installation files” for this command-line variable.
-DALERT_TIME_INTERVAL	Specifies the time interval in seconds to update alerts on web UI dashboard. Default value is 600. Allowed range is 60 to 5400 seconds.
-DVCREDIST	(For Windows only) In order for `lmadmin` to run, the Microsoft Visual C++ Redistributable Package must be installed. For the required minimum version, refer to the *Release Notes*, section *System Requirements for lmadmin*. Specify the value 1 to install the Redistributable Package in silent mode: `-DVCREDIST=1` (default value: 0; the Redistributable Package will not be installed).

Other Command-Line Variables

For a list of other command-line variables, view the contents of the `installvariables.properties` file found in the `uninstall` directory of a current `lmadmin` installation. To create a command-line variable based on a property listed in this file, precede the property name with *-D*. For example, to create a variable from `HTTP_PORT_NUMBER` in the properties file, enter `-DHTTP_PORT_NUMBER=`*`value`* in the command line.

Command-Line lmadmin Options for Installers

If you are creating your own `lmadmin` installer from scratch (that is, you are not incorporating the `lmadmin` installer version in your custom installation), wrap `lmadmin` command-line options into your installer to configure how `lmadmin` will run. The following `lmadmin` options are specifically targeted for installer configuration. To view all available `lmadmin` options, run `lmadmin -help` from the command line. For a description of all available options, see the *License Administration Guide*.

Table 10-2 • Command-Line lmadmin Options for the Installer

Option	Description
-importInstallation ***old installation directory***	Imports configuration information from the specified existing `lmadmin` installation directory or from a specified location for the license server configuration files (optional use of `-config` and/or `-configDir`). This option does not start `lmadmin`. This argument can be combined with `-config` and `-configDir`.
–installService ***service_name***	(For Windows only) Creates a Windows service (with the name you provide) to run the license server manager. ***Note •*** *After executing the command to install the license server manager as a Windows service, it is not started automatically. You must start the service for the first time using the Windows Services Console.*
–delay ***nn***	(For Windows only) Sets the number of seconds (nn) to delay between the time you start the service and the time it actually begins running. This delay is helpful when a FlexNet ID dongle is used to lock the license server to a machine (that is, when the FLEXID is used on the SERVER line). The license server can sometimes fail to start when a system reboots because the license server loads before the dongle device driver has a chance to load properly. ***Note •*** *Use this option with the `-installService` argument.*
-removeService	(For Windows only) Uninstalls the Windows service with the name you specified. Make sure you stop the service before removing it.
-config ***configuration_file_name***	Defines the name of the license server configuration file to use when starting the license server manager. By default, the path and file name is `<lmadmin_install_dir>/conf/server.xml` on non-Windows platforms, and `<lmadmin_runtimedata_dir>\conf\server.xml` on Windows platforms.

Table 10-2 • Command-Line lmadmin Options for the Installer

Option	Description
-configDir ***configuration_directory***	Defines the directory where the license server configuration files are located. By default, this path is `<lmadmin_install_dir>/conf` on non-Windows platforms, and `<lmadmin_runtimedata_dir>\conf` on Windows platforms. ***Note*** • *In installations where the* `conf` *folder is not installed in the same location as* `lmadmin.exe`*, you must use the* `-configDir` *command for all operations that involve updates to files in the* `conf` *folder (for example,* `server.xml`*).*
-licPort ***license_server_port***	Configures the license server manager port. By default, this number is the first available value in the range of 27000 through 27009. To set a specific port, enter a positive integer. To have FlexNet Publisher assign a default port, enter **`0`** (zero). ***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.*
-webPort ***http_port***	Configures the TCP/IP port that the Web server uses to listen for communication with clients who are connecting with the `lmadmin` interface. By default, this value is `8090`.
-webSecurePort ***https_port***	Configures the TCP/IP port (five characters maximum) that the `lmadmin` web server uses to listen for HTTPS (HTTP-over-SSL) communication. If you change the port, you must stop and restart the license server. By default, this value is **`0`** (no HTTPS communication). ***Note*** • *Specify the* `-webSecurePort` *option only if you are configuring HTTPS communication with the* `lmadmin` *web server interface.*

Table 10-2 • Command-Line lmadmin Options for the Installer

Option	Description
-defaultAdminGroup	(For Windows only) Adds the specified Windows Active Directory group as an `lmadmin` Administrator. The License Administrator can then log in to the `lmadmin` user interface initially using any Active Directory user ID (and its associated password) belonging to this group. To identify this group, use the format *`domain\groupname`*, where *domain* is a valid Active Directory domain to which the machine running `lmadmin` has a trusted relationship, and *`groupname`* identifies a valid account within that domain. This value is not case-sensitive and can include up to 32 characters. Using the `lmadmin` user interface, you can change this account's role as an `lmadmin` administrator or remove the account from the `lmadmin` user-interface user list. However, you cannot manage the account as an Active Directory account. ***Note •*** *Use this argument sparingly. Its main purpose is to provide initial access to the* lmadmin *interface.*
-defaultAdminUser	(For Windows only) Adds the specified Windows Active Directory user as an `lmadmin` Administrator. The License Administrator can then log in to the `lmadmin` user interface initially using the Active Directory user ID and password. To identify this user, use the format *`domain\username`*, where *domain* is a valid Active Directory domain to which the machine running `lmadmin` has a trusted relationship, and *`username`* identifies a valid user within that domain. This value is not case-sensitive and can include up to 32 characters. Using the `lmadmin` user interface, you can change this account's role as an `lmadmin` administrator or remove the account from the `lmadmin` user-interface user list. However, you cannot manage the account as an Active Directory account. ***Note •*** *Use this argument sparingly. Its main purpose is to provide initial access to the* lmadmin *interface.*

Installing lmadmin as a Service (Windows Platforms)

On Windows platforms, `lmadmin` can be installed as a service which automatically starts up at system initialization time. This can be configured if the option **Run as a service** (in the installer) is selected.

When `lmadmin` is configured as a service, the process creates the key as follows:

- 64-bit `lmadmin`: `HKEY_LOCAL_MACHINE\SOFTWARE\FNPlm\License Manager\`
- 32-bit lmadmin
  - 32-bit OS: `HKEY_LOCAL_MACHINE\SOFTWARE\FNPlm\License Manager\`
  - 64-bit OS: `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\FNPlm\License Server\`

Optionally, you can add the `EventTimeout` value to this key to configure how long the `lmadmin` service waits for the vendor daemon to start (without the `EventTimeout` registry value, `lmadmin` waits for 60 seconds.) For more information, see the later section, Configuring the Service Wait Time for Vendor Daemon Startup.

When `lmadmin` is installed as a service, the following best practices should be followed; otherwise, the service might not start:

- `lmadmin` and the vendor daemon must be installed to a subdirectory of `Program Files (x86)` (32-bit `lmadmin`) or `Program Files` (64-bit `lmadmin`).
- Data written by `lmadmin` at runtime must be written to a subdirectory of `ProgramData` (in folders `cache`, `conf`, `licenses`, and `logs`).

(Windows services typically have execute permissions only against applications in a `Program Files` subdirectory, and write access only to `Program Data`.)

Installing the License Server lmgrd

The following sections outline some considerations when providing the end user with `lmgrd` as your license server manager.

Installing lmgrd and Vendor Daemons

Under Windows, best practice is to install `lmgrd` and the vendor daemon in a subdirectory of `Program Files` for a 64-bit license server or `Program Files (x86)` for a 32-bit license server. Data that is written by executables at runtime (the debug log and report log) should be written to a subdirectory of `ProgramData`. This helps to ensure that the license server has sufficient privileges, and to pass compliance tests, such as those run by the Microsoft Attack Surface Analyser tool. (Windows services typically have execute permissions only against applications in a `Program Files` subdirectory, and write access only to `ProgramData`.)

At run time, `lmgrd` automatically searches for the vendor daemon in prescribed locations according to the following sequence:

1. The `vendor_daemon_path` parameter from the license file's `VENDOR` line.
2. The `lmgrd` process' current directory.
3. The `$PATH` environment variable in the `lmgrd` process environment.
4. The directory where the `lmgrd` executable resides.

Note • *For trusted-storage licensing on standard platforms, the vendor daemon's activation library component, vendor_libFNP.dll (Windows platforms) or vendor_libFNP.so (UNIX platforms), must be installed in the correct location as specified in the vendor daemon's preptool configuration file when the vendor daemon executable was prepped. Generally, this is the same location as the vendor daemon executable. See "Preptool Configuration File: Examples" in Programming Reference for Trusted Storage–Based Licensing for further details. For notr platforms, the activation library component, vendor_libFNP_notr.so, can be installed in any location consistent with platform-specific conventions for shared libraries.*

Starting lmgrd at System Startup Time (UNIX Platforms)

lmgrd can optionally be configured to automatically start up at system initialization time. See the *License Administration Guide* for more details.

Note • *For trusted-storage licensing, the* LD_LIBRARY_PATH *environment variable in the* lmgrd *process environment must include the path to the vendor daemon's activation library component, vendor_libFNP.so.*

Installing lmgrd as a Service (Windows Platforms)

On Windows platforms, lmgrd can be installed as a service which automatically starts up at system initialization time. This can be configured in one of two ways:

- Manually, using LMTOOLS. This procedure is discussed in the *License Administration Guide*.
- Automatically, by your application installer using the installs.exe utility.

When lmgrd is configured as a service, the process creates the key in the HKEY_LOCAL_MACHINE\SOFTWARE\FNPlm\License Manager\ path for a 32-bit OS and in the HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\FNPlm\License Server path for a 64-bit OS. Optionally, you can add the EventTimeout value to this key to configure how long the lmgrd service waits for the vendor daemon to start. (Without the EventTimeout registry value, lmgrd waits for 60 seconds.) For more information, see the section Configuring the Service Wait Time for Vendor Daemon Startup.

Installing lmgrd as a Service Using installs.exe

The installs.exe utility is provided by the licensing toolkit in the <platform_dir> directory. installs.exe can be used as a command-line utility to manually install the service, or it can be embedded in your end-user installation procedure to automatically install the service on behalf of the user at installation time.

The following shows the basic command syntax:

```
installs  -c license_file_path \
          -e lmgrd_location \
          -l log_file_path \
          -n service_name \
          [-r service_name] \
          [-k lmgrd parameters]
```

You provide the following:

Table 10-3 • installs.exe Inputs

Inputs	Description
-c *license_file_path*	Location of the license file. Path names with embedded spaces must be enclosed in double quotes.

Table 10-3 • installs.exe Inputs

Inputs	Description
`-e` *`lmgrd_location`*	Location of `lmgrd.exe` on your system. Path names with embedded spaces must be enclosed in double quotes. It is best practice is to install `lmgrd.exe` in a subdirectory of `Program Files (x86)` (for a 32-bit `lmgrd`) or `Program Files` (for a 64-bit `lmgrd`).
`-l` *`log_file_path`*	Location for the debug log file. Path names with embedded spaces must be enclosed in double quotes. The default location for the debug log file, when `lmgrd` is started as a service, is the current working directory path. To specify a different location, specify a fully qualified path. It is best practice to set the location of `debug_log_path` to a subdirectory of `ProgramData`, because this location has user-write permission by default, which is needed when the license server runs as a service with LocalService permission.
`-n` *`service_name`*	The service name. If not specified, ”`FlexNet License Manager`” is used by default. Service names with embedded spaces must be enclosed in double quotes.
`-r` *`service_name`*	Remove `service_name` service.
`-k` *`lmgrd parameters`*	Used to pass additional command line arguments to `lmgrd` each time it is started as a service. A string of parameters with embedded spaces must be enclosed in double quotes. One or more of the following can be specified: ● `-local`—Restricts the `lmdown` command to be run only from the same machine where `lmgrd` is running. ● `-x lmdown`—Disables the `lmdown` command (no user can run `lmdown`). ● `-x lmremove`—Disables the `lmremove` command (no user can run `lmremove`). See the *`lmgrd` Command-Line Syntax* section in *Programming Reference for License File–Based Licensing* for more information on these three arguments. Other `lmgrd` parameters are not supported here.

After `installs.exe` is run successfully, `lmgrd` is installed as a Windows service and will be started automatically each time the system is booted. If you want to customize the `installs` program, the source code is included in the file `machind\installs.c`.

Configuring the Service Wait Time for Vendor Daemon Startup

In situations where a local client itself is service, the client service should wait until the Windows Service Control Manager (SCM) indicates the `lmgrd` service is in the `Running` state before issuing a checkout request. To ensure the checkout request is successful, the `lmgrd` service waits until the vendor daemon has started and signaled a synchronization object, before the service itself signals to the SCM that it is in the `Running` state. While waiting to be signaled, the `lmgrd` service reports the `Start Pending` state to the SCM. The default amount of time the `lmgrd` service waits is 60 seconds, but this can be configured if you add the `EventTimeout` registry value to the `HKEY_LOCAL_MACHINE\ SOFTWARE\FLEXlm License`

Manager\<service name> key. The value is in milliseconds, meaning, for example, that you specify 400 seconds as ***400000***. Should the lmgrd service reach the EventTimeout time without being signaled by the vendor daemon, the service shuts down, and the SCM reports state of the lmgrd service as Stopped.

lmgrd Installation Examples

Task ***To install 64-bit lmgrd as a service***

Enter a similar command:

```
installs -e "C:\Program Files\FlexLM Licensing Server\lmgrd" \
         -c "C:\Program Files\FlexLM Licensing Server\sample.lic \
         -l C:\ProgramData\FNP_DIR\debug.log \
         -n "Myvendor License Manager"
```

where:

- C:\Program Files\FlexLM Licensing Server\lmgrd is the location of the lmgrd executable.
- C:\Program Files\FlexLM Licensing Server\sample.lic is the full path name for the license file.
- C:\ProgramData\FNP_DIR\debug.log is the full path name for the debug log file.
- "Myvendor License Manager" is the service name.

Task ***To remove the service named "Myvendor License Manager" installed in the previous example***

Enter a similar command:

```
installs -r -n "Myvendor License Manager"
```

Task ***To install 64-bit lmgrd as a service and to configure the*** *Lmgrd* ***startup parameter -x lmremove***

Enter a similar command:

```
installs -e "C:\Program Files\FlexLM Licensing Server\lmgrd" \
         -c "C:\Program Files\FlexLM Licensing Server\sample.lic \
         -l C:\ProgramData\FNP_DIR\debug.log \
         -n "Myvendor License Manager" \
         -k "-x lmremove"
```

Now, each time lmgrd starts up, it is invoked with the –x lmremove parameter.

Starting lmgrd Manually

Before running lmgrd manually, ensure that *either* of the following is true:

- The vendor daemon is in the same directory as lmgrd or in one of the directories specified in the lmgrd process's $PATH environment variable.

- For license file–based licensing, the end user has edited the license file to include the path to the vendor daemon on the `VENDOR` line.

Additionally, for trusted storage–based licensing on UNIX platforms, the `LD_LIBRARY_PATH` environment variable must include the path to the vendor daemon's activation library component, `vendor_libFNP.so`.

Unix Platforms

Task ***To start lmgrd manually on Unix platforms***

Trusted storage–based licensing:

`%setenv LD_LIBRARY_PATH` *`vendor_daemon_so`*

`%lmgrd -c` *`license_file_list`* `-l [+]`*`debug_log_path`*

License file–based licensing:

`lmgrd -c` *`license_file_list`* `-l [+]`*`debug_log_path`*

Windows Platforms

Task ***To start lmgrd manually on Windows platforms***

Use `LMTOOLS` or `lmgrd -c` *`license_file_list`* `-l [+]`*`debug_log_path`*

where `license_file_list` is the full path to the license file or a delimited license search path and `debug_log_path` is the path to the debug log file. Prepending `debug_log_path` with the + character appends logging entries. Use the -l option before other options to log all debugging information to `debug_log_path`.

It is best practice to set the location of `debug_log_path` to a subdirectory of `ProgramData`, because this location has user-write permission by default, which is needed when the license server runs as a service with LocalService permission. See the *License Administration Guide* for descriptions of the messages in the debug log file.

`lmgrd.exe` must be located in a subdirectory of `Program Files (x86)` (for a 32-bit `lmgrd`) or `Program Files` (for a 64-bit `lmgrd`).

A

Setting Up Microsoft Visual Studio for a FlexEnabled Application

As with many Microsoft Visual Studio projects, creating a FlexEnabled application requires particular project settings and additional header files and libraries. For a standard FlexEnabled C or C++ application, you need to specify the directories containing the FlexNet Publisher header files and libraries.

Task ***To set up Microsoft Visual Studio for a FlexEnabled application:***

1. Open Microsoft Visual Studio and change to the Solution Explorer view.
2. Right-click the project icon and select **Properties** (or select Properties from the Project menu). This displays your project's Property Pages.

   Note • *To avoid having to repeat some steps, select **All Configurations** from the **Configuration** menu at the top of the panel.*

3. Ensure that **Character Set** is set to **Use Multi-Byte Character Set**.
4. Specify the additional include directories where the FlexNet Publisher API headers files are found.
   a. Expand the C/C++ category in the left pane and select the first sub-item, called **General**.
   b. In the **Additional Include Directories** setting, browse for the `machind` subdirectory in your FlexNet Publisher toolkit directory.

      Note • *These instructions are only for a FlexEnabled application project. Other project types require different directories and libraries: for example, if you are building an activation utility, you should also browse for* `machind\activation\include`.

5. Specify the directories where the FlexNet Publisher libraries are found.
   a. Expand the Linker category in the left pane and select the first sub-item, again called **General**.

b. In the **Additional Library Directories** setting, browse for the following directories, located in your FlexNet Publisher toolkit directory:

- `i86_n3`
- `i86_n3\activation\lib`

6. Set linker properties by selecting the **Input** item under the **Linker** category.

a. Set **Ignore All Default Libraries** to **Yes**.

b. In the **Additional Dependencies** setting, specify which libraries to include. For a complete list of required libraries and files, see the contents of the `<platform_dir>\makefile` or `makefile.act`. Note the following exceptions:

- Include `msvcrt.lib` in the debug build only.
- Include `libcmt.lib` in the release build only.
- If your application uses trusted storage, also include the following libraries for *both* debug and release builds: `libFNPload.lib`, `FNP_Act_Installer.lib`, and `libact.lib` (which replaces `libnoact.lib`).

Note • *Because different builds use different libraries, if you selected **All Configurations** from the **Configuration** list, you should change the setting back to reflect the type of release you are building.*

Additionally, refer to the `<platform_dir>\makefile` or `makefile.act`. file to verify which system library dependencies should display in the **Inherited Values** list.

Including the FlexNet Publisher Header Files

In your source code, the FlexNet Publisher header files you need to include are the following:

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

Adding a preptool Build Step for Trusted Storage–Based Applications

Any FlexEnabled application (or activation utility) that uses trusted storage must be processed with `preptool` in order to create the (vendor-specific and product-specific) activation library DLL.

The command is `preptool` *`prepconfig.xml`*, where *`prepconfig.xml`* is the name of a configuration file that describes the file being prepped, the name of the DLL to create, whether to turn on anti-debugger technology, and so on. The `preptool` executable also needs access to `publisher.xml` and the `preparchive_version.yaa` file. For several examples, see the XML files in the `examples\activation` directory of the toolkit directory.

Because `preptool` needs to be run after every build, it can be useful to define a post-build event that runs `preptool`. Assuming you've copied `preptool.exe`, its configuration file, the `.yaa` file, and `publisher.xml` to the build directory (or specified alternative locations for everything), use the following instructions to define the post-build step.

Task ***To define the post-build event that runs preptool:***

1. From the **Project** menu, select **Properties** to display your project's **Property Pages** dialog.
2. In the **Build Events** category, select **Post-Build Event**.
3. In the **Command Line** field, type something similar to the following:

   ```
   cd "$(TargetDir)"
   preptool AppNameConfig.xml
   ```

4. In the **Description** field, type something similar to the following:

   ```
   Processing application with preptool...
   ```

   The description appears in the build output window when you build the project.

Building an MFC Application

An MFC project has its own required libraries and so on, so the build settings are a little different.

Task ***To build an MFC application:***

1. Open Microsoft Visual Studio and change to the Solution Explorer view.
2. Right-click the project icon and select **Properties** (or select Properties from the Project menu). This displays your project's Property Pages.
3. Expand the **Linker** category and select **Input**, leaving **Ignore All Default Libraries** with its default value **No**.
4. Depending on whether you are building a debug build or a release build, set the **Ignore Specific Default Libraries** setting to the appropriate library:
   - For a debug build, `libcmt.lib`.
   - For a release build, `msvcrt.lib`.
5. In the **Additional Dependencies** setting, specify the additional libraries to include. For a complete list of required libraries and files, see the contents of the `<platform_dir>/makefile` or `makefile.act`.

   If your application uses trusted storage, be sure to include the following libraries for *both* debug and release builds: `libFNPload.lib`, `FNP_Act_Installer.lib`, and `libact.lib` (which replaces `libnoact.lib`).

   Note • *Because different builds use different libraries, if you selected* ***All Configurations*** *from the* ***Configuration*** *list, you should change the setting back to reflect the type of release you are building.*

Setting Up Other Application Types

Activation Utilities

If you are creating an application activation utility or a server activation utility, you need to add `machind\activation\include` to the list of locations for header files.

As with a FlexEnabled application, the libraries are contained in the folders `i86_n3` and `i86_n3\activation\lib`.

Activation utilities need to be processed with `preptool`, so you may want to set up a post-build event as described in the section, Adding a preptool Build Step for Trusted Storage–Based Applications.

Manual License Generator Applications

For a Manual License Generator application, point to the `<installdir>\machind\activation\include\responsegen` directory for responsegen header files, and `<installdir>\examples\activation\responsegen\` for example usage of the responsegen API.

For the library files, point to the `<installdir>\<platform_dir>\activation\lib\responsegen` directory and include the `responsegen.lib` library.

You also need `responsegen.dll` in the application's directory at run time.

Glossary

Term	Meaning
activatable license	A license right in trusted storage that can be activated to a FlexEnabled client. See also license group.
activated feature line	See feature definition line.
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.
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.

Term	Meaning
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.
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 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.
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.

Term	Meaning
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.
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.

Term	Meaning
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.
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.

Term	Meaning
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.
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.

Term	Meaning
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.
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.
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.

Term	Meaning
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. Flexera 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.
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.

Term	Meaning
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.
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.

Term	Meaning
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. Flexera 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.
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.
windback tolerance	An acceptable amount of windback to a system clock that won't trigger windback detection. A certain tolerance is normally allowed to account for time zone differences or Daylight Saving Time.

Index

Symbols

.NET development environment 107

A

activatable license 179
activated feature line, See feature definition line
activation 179
    library 84, 88
    short code 188
Activation API 40, 179
activation application 180
Activation Borrow API 40
activation client 180
Activation Licensing Components
    Server Activation Utility 56
activation request 180
activation response 180
activation server 180
activation settings file 65, 66
    publisher.xml 18
    sdt_publisher.xml 65, 66
activation specification record 19, 180
    creating 81
    tool for creating 35
activation transaction 180
    repair 94
    return 94
activation utility 89
ADT, Anti-Debugger Technology 132
alternative development environment
    overview 107
anchor 180
anchoring 20
application activation utility 82
ASR, See activation specification record 19
asrgen 35

B

binding 20, 132, 180
borrow reclaim count
    resetting 58
    setting 58
borrowing 181
    license rights 58
borrowing interface 127
    license 127
building FlexEnabled application
    UNIX 97
    Windows 102
building production Licensing toolkit
    UNIX 23
    Windows 25
bulk entitlement 181

C

C environment
    UNIX 23
    Windows 25
changing vendor daemon name 18, 64, 66
checkin 181
checkout 181
client system 181
composite transaction 181
concurrent license 181
config response 181
configuration files, Licensing toolkit 32

configuring, Licensing toolkit 15
considerations
end-user-side 122
vendor-side 121
consumer 181
counted license 181
customizing
Licensing toolkit 14

D

daemon, vendor 189
debug log file 181
default license file location 124
demo license, See trial license
denial/checkout failure, license 125
deployed FlexNet Licensing files
eventlogadd.reg 117
eventlogremove.reg 117
lmgrd 116
lmtools.exe 116
lmutil 116
MvsnLicenseServerMsgs.dll 117
vendor daemon 116
VendorLicenseServerMsgs.dll 117
deployment
installing lmgrd as service 171
starting lmgrd manually 173
starting lmgrd on boot 171
duplicate grouping 128, 182
dynamic linking 105

E

editing lm_code.h 16, 63
embedded activation, See local activation
enabled
license file environment variable 124
license file location in registry 125
encryption 15, 63, 64
encryption seed 182
ENCRYPTION_SEEDs 63
end user 182
end-user license administrator, See license administrator
end-user machine, See client system
end-user-side considerations 122
enterprise 182
entitlement 182, 185
entitlement ID 182
environment variable 124
environment variable enabled, license file 124
event logging
configuring registry 120
enabling support 119
message file DLLs 120
eventlogadd.reg 120
eventlogremove.reg 120
expiring license 182

F

failure, license denial/checkout 125
feature 183
feature bundle 183
feature definition line 179, 183
FEATURE/INCREMENT line, See feature definition line
file
installing the license 121
license 184
options 186
report log 187
file environment variable enabled, license 124
file-based activation, See manual activation
finder (Windows only), license 125
FlexEnabled application 183
FlexEnabled client 183
Flexible API 37, 133, 149, 183
FLEXLM_DLL 106
FlexNet Agent 183
FlexNet Licensing
client DLL 106
components 123
FlexNet Licensing Service 87, 92, 137
installing (Linux) 49
installing (OS X) 50
FlexNet Operations 183
floating license 184
fulfill 184
fulfillment fecord 81
fulfillment record 184

G

generated components, Licensing toolkit 33
generating licenses using lmcrypt 117
grouping duplicate license requests 128

H

heartbeats 184
automatic 71, 97
hop count 184
hostid 184
informing end-users about 124
hostid type 184
hybrid license 184

I

installing
- FlexNet Licensing Service (Linux) 49
- FlexNet Licensing Service (OS X) 50
- license certificate 121
- lmadmin as a service 169
- lmgrd as service 171
- vendor daemon 170

installs.exe 171
integration decisions
- server configuration 119

interface, license borrowing 127
internet activation, See programmatic activation

K

keyboard activation, See short-code activation

L

l_n36_buf error 99
libcmt.lib (/MT) 103
libcrvs.a 99
libcrvs.lib 76, 78, 103, 105, 108
libsb23.a 99
libsb23.lib 76, 78, 103, 105, 108
license
- counted 181
- finder 125
- lingering 126
- served 188
- timeout 126
- unserved 189

license administrator 184
license borrowing interface 127
license denial/checkout failure 125
license file 90, 184
- default location 124
- delivering 119
- generating with lmcrypt 118
- installing the 121
- location 125
- updating registry 125

license finder 125
license generator 184
license group 185
license model 185
license rights 185
- borrowing 58
- local 186

license search path 185
license server 185
- installation instructions 158
- installing 93, 122
- installing lmadmin as a service 169
- installing lmgrd 170
- lmadmin command-line installation variables 163
- lmadmin command-line options 167
- lmadmin installer properties file 163
- lmadmin silent installation 161
- lmadmin user-interface-based installation 160
- organization 128
- requirement 94

license server manager 90, 185
license server manager, definition 185
license server system 185
license sharing 186
license-file list, See license search path
licensing client 186
Licensing Service API 40
Licensing toolkit
- components 32
- configuration files 32
- configuring 15
- customizing 14
- generated components 33
- source code 43
- standard components 36
- testing 32
- testing on UNIX 44

licensing utilities
- asrgen 35
- lmcrypt 35
- tsreset 35

lingering, license 126
linking
- dynamic 105
- static 104

lm_code.h 14, 32, 63
- editing 16, 63
- ENCRYPTION_SEEDs 63
- LM_SEEDs 16
- LM_STRENGTH 17
- TRL_KEYs 16, 64
- usage 16
- VENDOR_KEYs 16, 63
- VENDOR_NAME 16, 63

lm_new_md.obj 105
lm_new.o 15, 16, 99
lm_new.obj 16, 103
LM_SEEDs 16
LM_STRENGTH 17
- LM_STRENGTH_113BIT 17
- LM_STRENGTH_163BIT 17
- LM_STRENGTH_239BIT 17
- LM_STRENGTH_DEFAULT 17

lmadmin 90, 186
command-line installation variables 163
command-line options for installer 167
installing as a service 169
properties file for installer 163
silent installation 161
user-interface-based installation 160
lmcrypt 15, 35, 117
lmflex.c 103
lmgr_trl.lib 76, 78, 105
lmgr.lib 76, 78, 105
lmgrd 90, 186
installing 170
installing as service 171
starting on boot 171
lmstrip 110
and other FlexNet Licensing client libraries 115
overview 110
local activation 186
local license rights 186
localizing message files 121
location in registry enabled, license file 125
log file
debug 181
report 187
lsvendor.c 63

M

Mac 71, 98
machineid 186
makefile for toolkit 15
manual activation 186
Manual Response Generator 186
message files
localizating 121
shipping DLLs 120
Microsoft event logging, localizing 121
Microsoft Visual C++, building with 25
model, license 185
msvcrt.lib 67
msvcrt.lib (/MD) 105
multithread environment 104
multithreaded application 71, 97

N

newlink GlossaryC 181
notr 10, 70

O

options file 186

P

package 186
part number 186
permanent license 187
PIC libraries 71, 98
PIE support 24
position-independent executables, see PIE support 24
PREPARCHIVE 24
preptool 79, 92, 132, 137, 148, 170
preptool configuration file 19, 65, 67
product 187
programmatic activation 187
properties file, for lmadmin installer 163
publisher id 187
publisher.xml 15, 18, 32, 65, 66

Q

quorum 187

R

ranlib 30
reconnection strategy 126
registry enabled, license file location 125
regular entitlement, See simple entitlement
rehost 187
repair request 187
repair response 187
report log file 187
responsegen 187
return request 187
return response 187
return transaction 188

S

sdt_publisher.xml 32, 65, 66
SDT, Secure Data Types 132
sdtconfig.h 71
SDTs 40, 71, 73, 77, 99, 107
C++ 146
Choosing 136
Feature Definition header file 146
Feature Implementation file 147
Files required for 138
Guidelines 138
In multiple executables 140
Mapping to features 132, 136
Multithreaded environment 140
Testing code 136
Secure Data Types 40, 71, 73, 77, 99, 107, 132

served license rights 188
server
    license 185
server need 94
server redundancy 128
shared libraries 98
shared objects 71
shipping message file DLLs 120
short code 188
short-code activation 188
signature 188
silent installation, lmadmin 161
simple entitlement 188
single-action transaction 188
source code, Licensing toolkit 43
SSL Certificate 92
standard components, Licensing toolkit 36
static linking 104
strategy
    reconnection 126
suite 188
supersede 188

T

tamper-resistant applications, See Secure Data Types
tamper-resistant licenses 97
    security considerations 70
testing your application
    in license file-based licensing 109
    in trusted storage-based licensing 80
three-server redundancy 188
timeout, license 126
toolkit makefile 15, 17, 32, 64
triad 188
trial license 188
TRL 16, 70, 97
TRL keys 62
TRL_KEYs 16, 64
TRL, See tamper-resistant licenses 97
Trusted Configuration file 19
trusted configuration file 188
Trusted Storage 52, 84
trusted storage 188
trusted storage configuration 189
tsreset 35
type, hostid 184

U

uncounted license
    uncounted 189
unserved license rights 189
unserved licenses 189
upgrade 189
upgrading 62
upsell 189

V

Vendor Certificate Generator 189
Vendor Daemon 90
vendor daemon 15, 189
    installing 170
vendor keys 15, 62
vendor name 15, 63, 189
    changing 18, 64, 66
VENDOR_KEYs 16, 63
VENDOR_NAME 16, 63
VendorLicenseServerMsgs.dll 120
VendorLicenseServerMsgs.mc 121
vendor-side considerations 121
Visual Basic 107

W

windback 189
windback detection 189
windback tolerance 189
Windows event logging 42
    deployed files 119