

*Programmer's
Reference
Manual*



Copyright 1999, Rainbow Technologies, Inc.

All rights reserved.

<http://www.rainbow.com>

All attempts have been made to make the information in this document complete and accurate. Rainbow Technologies, Inc. is not responsible for any direct or indirect damages or loss of business resulting from inaccuracies or omissions. The specifications contained in this document are subject to change without notice.

CONFIDENTIAL INFORMATION

The SentinelLM software protection system is designed to protect your software products from unauthorized use. The less information that unauthorized people have regarding your security system, the greater your protection. It is in your best interest to protect the information herein from access by unauthorized individuals. Please read the Developer's Agreement at the beginning of the developer's guide for safeguarding requirements.

Part Number 700554-001, Revision A

Software releases 7.0 and later

RAINBOW TECHNOLOGIES, INC.

50 Technology Drive, Irvine, CA 92618

Telephone: (949) 450-7300, (800) 852-8569

Fax: (949) 450-7450

RAINBOW TECHNOLOGIES LTD.

4 The Forum, Hanworth Lane, Chertsey, Surrey KT16 9JX, United Kingdom

Telephone: (44) 1932 579200

Fax: (44) 1932 570743

RAINBOW TECHNOLOGIES

122, Avenue Charles de Gaulle, 92522 Neuilly-sur-Seine Cedex, France

Telephone: (33) 1 41 43 29 02

Fax: (33) 1 46 24 76 91

RAINBOW TECHNOLOGIES GMBH

Lise Meitner Strasse 1, 85716 Unterschleissheim, Germany

Telephone: (49) 89 32 17 98 0

Fax: (49) 89 32 17 98 50

Additional offices in the United States, Australia, China, India, the Netherlands, Russia and Taiwan.
Distributors located worldwide.

SentinelLM is a trademark of Rainbow Technologies, Inc. Novell and NetWare are trademarks of Novell, Inc. Microsoft Windows, Microsoft Windows NT, Windows 95 and Windows 98 are trademarks of Microsoft Corporation. UNIX is a registered trademark, exclusively licensed through X/Open Company, Ltd. All other product names referenced herein are trademarks or registered trademarks of their respective manufacturers.

10 9 8 7 6 5 4 3 2 1 082099



SOFTWARE LICENSE AND DEVELOPER'S AGREEMENT

All Products (including developer's kits, Sentinel hardware keys, diskettes or other magnetic media, software, documentation and all future orders) are subject to the terms stated below. If you disagree with these terms, please return the Product and the documentation to Rainbow, postage prepaid, within three days of your receipt, and Rainbow will provide you with a refund, less freight and normal handling charges.

1. You may not copy or reproduce all or any part of the Product, except as authorized in item 2 below. Removal, emulation or reverse-engineering of all or any part of the Product constitutes an unauthorized modification to the Product and is specifically prohibited. Nothing in this license permits you to derive the source code of the software files that Rainbow has provided to you. Your software programs must be protected or licensed using a licensed and registered copy of this Rainbow Product. Rainbow provides no other warranty to any person, other than the Limited Warranty provided to the original purchaser of this Product.
- 2.a. You may make archival copies of the software files and you may modify and merge them into your software programs for the sole purpose of implementing the Product to protect and/or license your programs according to the Rainbow documentation provided with the Product. All software files remain Rainbow's exclusive property.
- b. Rainbow's Sentinel System Driver Software and other Rainbow software files listed in the "Licensee Redistribution Allowances" section (if it is defined in the Product's documentation) may be copied and distributed to your customers for the sole purpose of executing your protected or licensed software programs according to the Rainbow documentation provided with the Product.
- c. No license is granted to Licensee to sell, license, distribute, market or otherwise dispose of any software files or other component of the Product except when embedded in your software programs. Copies of your software programs must bear a valid copyright notice and must be distributed such that the object code for the Product cannot be extracted.
3. Rainbow warrants the Product and the magnetic media on which the software files are provided to be substantially free from significant defects in materials and workmanship under normal use for a period of twelve (12) months from the date of delivery of the Product to you. In the event of a claim under this warranty, Rainbow's sole obligation is to replace or repair, at Rainbow's option, any Product free of charge. Any replaced parts shall become Rainbow's property.
4. Warranty claims must be made in writing during the warranty period and within seven (7) days of the observation of the defect, accompanied by evidence satisfactory to Rainbow. Prior to returning any Product to Rainbow, you must obtain a Return Merchandise Authorization (RMA) number and

shipping instructions from Rainbow. Products returned to Rainbow shall be shipped with freight and insurance paid.

5. Except as stated above, there is NO OTHER WARRANTY, REPRESENTATION, OR CONDITION REGARDING RAINBOW'S PRODUCTS, SERVICES, OR PERFORMANCE, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Rainbow is not responsible for any delays beyond its control. Rainbow's entire liability for damages to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of Product that caused the damages or that are the subject matter of, or are directly related to, the cause of action. In no event will Rainbow be liable for any damages caused by your failure to perform your obligations, or for any loss of data, profits, savings, or any other consequential and incidental damages, or for any claims by you based on any third-party claim.

Licensee Redistribution Allowances

SentinelLM Licensees may release the Sentinel System Driver diskette for installation with their Sentinel-protected application and the Sentinel Client Activator and associated files. In addition, the Licensee may distribute the following commands, files, and related documentation: ainst.exe, commute.dat, dlt, echoid, echoid.dat, ipxecho.exe, lcommute, lcu, loadls.exe, lsapiw32.dll, lsdecode, lserv, lserv9x.exe, lservnt, lslic, lsmail.exe, lsmon, lspool, lsrvdown, lsusage, lswhere, prsclean, rlftool, timefix, wcommute, WlmAdmin, wrlftool, and the *SentinelLM System Administrator's Guide*.

International Quality Standard Certification



Certificate Number FM 30128

Rainbow Technologies, Inc. Irvine CA facility has been issued the ISO 9002 Certification, the globally recognized standard for quality, by British Standards Institution as of December 1994.

European Community Directive Conformance Statement



This product is in conformity with the protection requirements of EC Council Directive 89/336/EEC. Conformity is declared to the following applicable standards for electro-magnetic compatibility immunity and susceptibility; CISPR22 and IEC801. This product satisfies the CLASS B limits of EN 55022.

Contents

Preface	xv
The SentinelLM Manualsxv
About This Guide	xvi
Typographic Conventions	xvii
Syntax Conventions	xviii
Getting Help	xviii
Online Documentation	xviii
For Additional Help	xix
Contacting Rainbow Technical Support	xix
How to Report Problems	xxi
Chapter 1 - Introduction	1
Using the SentinelLM Application Library	1
Licensing on Stand-alone and Networked Computers.	3
Client API Example	3
Example	4
Language Interfaces Supported	5
Special Use of Win32 for Generating Tools.	5
Debugging Your Client Application.	6
Disabling Licensing	6
Chapter 2 - Protecting Your Application with the Application Library	9
Adding APIs to Your Source Code	9
Application Identification	10
Automatic License Server Detection.	10
Special Licensing Cases	12
Linking with the Correct Library	13
Windows Static Linked Libraries	13
Windows Dynamic Linked Libraries and Import Libraries	15

UNIX Libraries	15
Notes on Security	16
Protecting Against Time Tampering	17
Using a Custom Locking Code	17
Step 1 - Rebuilding License Server	17
Compiler Required	17
Files Required	18
Required Changes to Server Source Code	18
Steps to Rebuilding the License Server	19
Step 2 - Rebuilding echoid.exe	19
Compiler Required	20
Files Required for echoid.exe	20
Required Changes to echoid.exe	20
Steps to Rebuilding echoid.exe.	21
Step 3 - Modifying Client Application	21
Overall Process of Using a Rebuilt License Server and Rebuilt echoid.exe	22
Chapter 3 - SentinelLM Client API	23
Introduction	23
Basic Client Licensing Functions.	25
Quick Client Licensing Functions.	25
VLSlicense().	26
VLSdisableLicense()	29
Standard Client Licensing Functions	31
VLSinitialize().	31
LSRequest()	32
LSRelease()	36
VLScleanup()	37
LSUpdate()	38
Advanced Client Licensing Functions	41
VLSinitialize().	42
VLSrequestExt().	42
Challenge-response	45
VLSreleaseExt().	48
VLScleanup()	49
VLSbatchUpdate().	49

Client Configuration Functions	52
VLSsetContactServer()	53
VLSgetContactServer()	56
VLSsetServerPort()	57
VLSgetServerPort()	58
VLSinitMachineID()	58
VLSgetMachineID()	60
VLSmachineIDtoLockCode()	61
VLSgetServerNameFromHandle()	62
VLSinitServerList()	63
VLSgetServerList	64
VLSinitServerInfo()	65
VLSsetHostIdFunc()	65
VLSsetBroadcastInterval()	66
VLSgetBroadcastInterval()	67
VLSsetTimeoutInterval()	67
VLSgetTimeoutInterval()	68
VLSsetHoldTime()	69
VLSsetSharedId()	70
VLSsetSharedIdValue()	72
Local vs. Remote Renewal of Keys	73
VLSdisableLocalRenewal()	74
VLSenableLocalRenewal()	75
VLSisLocalRenewalDisabled()	75
VLSgetRenewalStatus()	76
VLSsetRemoteRenewalTime()	77
VLSdisableAutoTimer()	78
Client Query Functions	79
VLSgetClientInfo()	81
VLSgetHandleInfo()	83
VLSgetLicInUseFromHandle()	84
Feature Query Functions.	86
VLSgetFeatureInfo()	90
VLSgetVersions()	93
VLSgetFeatureFromHandle()	94
VLSgetVersionFromHandle()	95

VLSgetTimeDriftFromHandle()	96
VLSgetFeatureTimeLeftFromHandle()	97
VLSgetKeyTimeLeftFromHandle()	99
Client Utility Functions	100
VLSdiscover()	101
VLSaddFeature()	104
VLSaddFeatureToFile()	105
VLSdeleteFeature()	107
VLSgetLibInfo()	109
VLSshutDown()	110
VLSwhere()	112
Error Handling	113
VLSerrorHandle()	114
LSGetMessage()	115
VLSsetErrorHandler()	116
VLSsetUserErrorFile	117
Tracing SentinelLM Operation	118
Chapter 4 - License Code Generation API.	121
License Code Generation Functions	122
Basic Functions	126
VLScgInitialize()	126
VLScgCleanup()	127
VLScgReset()	128
Functions Which Retrieve or Print Errors	128
VLScgGetNumErrors()	129
VLScgGetErrorLength()	129
VLScgGetErrorMessage()	130
VLScgPrintError()	131
Functions for Setting the Fields in Code Struct	132
VLScgAllowAdditive()	135
VLScgSetAdditive()	136
VLScgSetCodeLength()	136
VLScgSetLicType()	138
VLScgAllowHeldLic()	139
VLScgSetHoldingCrit()	139

VLScgAllowStandAloneFlag()	140
VLScgAllowNetworkFlag()	141
VLScgSetStandAloneFlag()	141
VLScgAllowSharedLic()	142
VLScgSetSharedLicType()	142
VLScgAllowTrialLicFeature()	144
VLScgSetTrialDaysCount()	144
VLScgAllowLockMechanism()	145
VLScgSetClientLockMechanism()	145
VLScgSetServerLockMechanism1()	146
VLScgSetServerLockMechanism2()	147
VLScgAllowClockTamperFlag()	148
VLScgSetClockTamperFlag()	148
VLScgAllowOutLicType()	150
VLScgSetOutLicType()	150
VLScgAllowLicenseType()	151
VLScgSetLicenseType()	152
VLScgAllowCodegenVersion()	153
VLScgSetCodegenVersion()	153
VLScgAllowRedundantFlag()	154
VLScgSetRedundantFlag()	154
VLScgAllowMajorityRuleFlag()	155
VLScgSetMajorityRuleFlag()	156
VLScgAllowCommuterLicense()	157
VLScgSetCommuterLicense()	157
VLScgAllowLogEncryptLevel()	159
VLScgSetLogEncryptLevel()	159
VLScgAllowMultiKey()	160
VLScgSetKeyType()	160
VLScgAllowMultipleServerInfo()	162
VLScgAllowSecrets()	162
VLScgSetSecrets()	163
VLScgSetNumSecrets()	164
VLScgAllowVendorInfo()	165
VLScgSetVendorInfo()	165
VLScgAllowFeatureName()	166

VLScgSetFeatureName()	166
VLScgAllowFeatureVersion()	167
VLScgSetFeatureVersion()	168
VLScgAllowLockModeQuery()	169
VLScgSetClientServerLockMode()	169
VLScgAllowServerLockInfo()	170
VLScgSetServerLockInfo1()	170
VLScgSetServerLockInfo2()	171
VLScgAllowClientLockInfo()	172
VLScgSetClientLockInfo()	173
VLScgAllowKeysPerNode()	174
VLScgSetKeysPerNode()	174
VLScgAllowSiteLic()	175
VLScgSetSiteLicInfo()	175
VLScgSetNumSubnets()	176
VLScgAllowNumFeatures()	177
VLScgSetNumFeatures()	177
VLScgSetNumClients()	178
VLScgAllowNumKeys()	179
VLScgSetNumKeys()	179
VLScgAllowSoftLimit()	180
VLScgSetSoftLimit()	181
VLScgAllowKeyLifeUnits()	182
VLScgSetKeyLifetimeUnits()	182
VLScgAllowKeyHoldUnits()	183
VLScgSetKeyHoldtimeUnits()	183
VLScgAllowKeyLifetime()	185
VLScgSetKeyLifetime()	185
VLScgAllowKeyHoldtime()	186
VLScgSetKeyHoldtime()	186
VLScgAllowLicBirth()	187
VLScgSetLicBirthMonth()	188
VLScgSetLicBirthDay()	189
VLScgSetLicBirthYear()	189
VLScgAllowLicExpiration()	190
VLScgSetLicExpirationMonth()	191

VLScgSetLicExpirationDay()	191
VLScgSetLicExpirationYear()	192
VLScgAllowShareLimit()	193
VLScgSetShareLimit()	193
VLScgSetNumericType()	194
VLScgSetLoadSWLicFile()	195
License Generation Functions	196
VLScgGenerateLicense()	196
VLScgDecodeLicense()	197
License Meter Related Functions	198
VLScgGetLicenseMeterUnits()	198
VLScgGetTrialLicenseMeterUnits()	199
Trial License Related Functions	200
VLSgetTrialPeriodLeft()	200
Chapter 5 - Redundancy API	201
VLSaddFeature()	203
VLSaddFeatureExt()	205
VLSaddFeatureToFile()	206
VLSaddServerToPool()	208
VLSchangeDistbCrit()	209
VLSdelServerFromPool()	210
VLSdiscoverExt()	212
VLSgetDistbCrit()	215
VLSgetDistbCritToFile()	217
VLSgetFeatureInfoToFile()	219
VLSgetHostName()	220
VLSgetLeaderServerName()	221
VLSgetHostAddress()	223
VLSgetLicSharingServerList()	224
Chapter 6 - License Queuing API.	227
License Queuing Example Code.	227
License Queuing Functions	231
VLSqueuedRequest() and VLSqueuedRequestExt()	232
VLSgetQueuedClientInfo()	237

VLSremoveQueuedClient()	238
VLSremoveQueue()	240
VLSgetHandleStatus()	241
VLSupdateQueuedClient()	242
VLSgetQueuedLicense()	245
VLSinitQueuePreference()	247
Chapter 7 - Commuter License API	249
Commuter License Related Functions	249
VLSgetCommuterInfo()	249
VLSgetAndInstallCommuterCode()	250
VLSuninstallAndReturnCommuterCode()	252
Chapter 8 - Usage Log Functions	253
VLSchangeUsageLogFileName()	253
VLSgetUsageLogFileName()	254
Chapter 9 - Utility Functions	255
VLSscheduleEvent()	255
VLSdisableEvents()	256
VLSeventSleep()	256
Appendix A - Sample Applications	259
Sample Program Summary	259
Customization Samples	260
Appendix B - Customization Features.	261
Initializing the Server.	263
VLSserverVendorInitialize()	263
VLSeventAddHook()	263
Protecting Against Time Clock Changes	266
VLSconfigureTimeTamper()	267
VLSisClockSetBack()	268
Encrypting License Codes	269
VLSencryptLicense()	269
VLSdecryptLicense()	271
Encrypting Messages	272
VLSencryptMsg()	273

VLSdecryptMsg()	274
Changing the Default Port Number	276
VLSchangePortNumber()	276
Customizing the Host ID.	277
Creating the Custom Host ID Function	278
Registering the Custom Host ID Function on the Server	279
Registering the Custom Host ID Function on the Client.	279
Building the Server.	280
Creating an Updated Client ID Generator	280
Using a Customized Host ID.	280
Appendix C - Error and Result Codes for Client Functions	283
Client Function Return Codes	283
Appendix D - Error and Result Codes for License Generation Functions . .291	291
License Generation Function Return Codes	291
Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions	295
Return Codes.	295
Appendix F - Error and Result Codes for SentinelLM-Shell.	299
SentinelLM-Shell Return Codes	299
Appendix G - File Formats.	301
License Code File Format	301
Configuration File Format	302
Log File Format	306
Index	309

Preface

Thank you for choosing the SentinelLM™ licensing product to license your software. Read on for information on using the SentinelLM Application Library to add protection to your applications.

The SentinelLM Manuals

The SentinelLM product includes several manuals, all designed to work in conjunction with each other:

Manual	What's in it?	Who should read it?
<i>SentinelLM Quick Start Guide for Windows</i>	A quick tour of SentinelLM for Windows application developers.	Anyone who is new to SentinelLM or license management and wants a quick overview of SentinelLM features.
<i>SentinelLM Developer's Guide</i>	All the steps necessary to protect, package, and ship a stand-alone or network application protected with SentinelLM-Shell or the SentinelLM Application Library.	Developers using SentinelLM-Shell or the API option who are responsible for the overall process of protecting and shipping an application for a stand-alone or network computer.
<i>SentinelLM Programmer's Reference Manual</i>	Description of the SentinelLM Application Library.	Developers who are using the SentinelLM Application Library to protect their applications. This manual assumes you are familiar with the C programming language, although other language interfaces are available.

About This Guide

Manual	What's in it?	Who should read it?
<i>SentinelLM System Administrator's Guide</i>	Information for the end user of your protected application, including use of administrator commands and configuring and using a license server.	End users of your protected application who are responsible for administering the application and end user license management and who are familiar with system administration tasks.

About This Guide

This guide gives all the steps for planning for protecting your application, as well as for protecting, packaging, and shipping your protected application to your customers.

Chapter/Appendix	Description
<i>Chapter 1 - Introduction</i>	Shows how SentinelLM is put together.
<i>Chapter 2 - Protecting Your Application with the Application Library</i>	Provides instructions and information on client library functions and compiling applications.
<i>Chapter 3 - SentinelLM Client API</i>	Provides a complete reference of all client functions.
<i>Chapter 4 - License Code Generation API</i>	Explains how to generate license codes.
<i>Chapter 5 - Redundancy API</i>	Summarizes the redundancy functions.
<i>Chapter 6 - License Queuing API</i>	Explains the license queuing functions.
<i>Chapter 7 - Commuter License API Codes</i>	Summarizes the commuter license related functions.
<i>Chapter 8 - Usage Log Functions</i>	Explains the usage log functions.
<i>Chapter 9 - Utility Functions</i>	Summarizes functions for the UNIX platform.

Chapter/Appendix	Description
<i>Appendix A - Sample Applications</i>	Lists source code for the sample programs and utilities.
<i>Appendix B - Customization Features</i>	Lists the features that can be customized.
<i>Appendix C - Error and Result Codes for Client Functions</i>	Lists client function return codes.
<i>Appendix D - Error and Result Codes for License Generation Functions</i>	Lists license generation function return codes.
<i>Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions</i>	Lists return codes for redundancy, queuing and commuter functions.
<i>Appendix F - Error and Result Codes for SentinelLM-Shell</i>	Lists SentinelLM-Shell return codes.
<i>Appendix G - File Formats</i>	Summarizes all the environment variables.

Typographic Conventions

The following typographic conventions are used throughout this guide:

Convention	Purpose
<i>italic</i>	Used to signal a new term, for placeholders, variables, and file names, or for emphasis.
bold	Used for command-line options, utility, dialog box, and checkbox names.
<i>bold-italic</i>	Used for characters you type, such as <i>logon</i> .
courier	This font denotes syntax, prompts, and code examples.

Syntax Conventions

The following syntax conventions are used throughout this guide:

Convention	Purpose
[]	Square brackets enclose optional syntax.
...	Ellipses indicate that a clause can be repeated.
	A pipe indicates that only one of the syntax choices it separates may be used.
{ }	Curly braces indicate that one of the options they enclose <i>must</i> be used in actual syntax.

Getting Help

For an introduction to the SentinelLM product, please see the Windows online tutorial, discussed in the *SentinelLM Developer's Guide*. In addition to providing basic information on SentinelLM, the tutorial also runs demonstration programs that illustrate licensing techniques.

The *SentinelLM Quick Start Guide for Windows* also gives the Windows developer's hands-on experience using SentinelLM.

Another way of getting familiar with SentinelLM is to use the Windows SentinelLM-Shell, discussed in the *SentinelLM Developer's Guide*.

Online Documentation

For Windows computers, refer to the Adobe Acrobat versions of the SentinelLM manuals in the \Manuals directory in the SentinelLM directory. See the *SentinelLM Developer's Guide* for instructions on installing the Acrobat Reader on Windows computers.

For UNIX computers, refer to the manual pages in the *SentinelLM/man* directory. You may move these to the computer's current manual-page directory

(typically `/usr/man/man3` or `usr/local/man/man3`) or modify the `MANPATH` environment variable to include the `SentinelLM/man` directory. Adobe Acrobat versions of the SentinelLM manuals are also included in the `SentinelLM/man` directory. See the *SentinelLM Developer's Guide* for instructions on installing the Acrobat Reader on UNIX computers.

For Additional Help

If you have questions concerning the SentinelLM product, contact Rainbow Technologies Technical Support. (See the next section for details.)

Contacting Rainbow Technical Support

Rainbow Technologies is committed to supporting SentinelLM. If you encounter a problem we want to hear about it.

To contact us, please use one of the methods listed in the following table:

Corporate Headquarters North America & South America	
Internet	Rainbow Technologies North America http://www.rainbow.com
E-mail	techsupport@irvine.rainbow.com
Telephone	800 959-9954 (8:00 a.m. - 5:00 p.m. PST)
Fax	(949) 450-7450
Australia	
E-mail	Rainbow Technologies (Australia) Pty Ltd. techsupport@au.rainbow.com
Telephone	(61) 3 9820 8900
Fax	(61) 3 9820 8711

Contacting Rainbow Technical Support

China	
	Rainbow Information Technologies (China) Co.
E-mail	techsupport@cn.rainbow.com
Telephone	(86) 108 209 0680
Fax	(86) 108 209 0681
France	
	Rainbow Technologies
E-mail	techsupport@fr.rainbow.com
Telephone	(33) 1 41.43.29.02
Fax	(33) 1 46.24.76.91
Germany	
	Rainbow Technologies, GMBH
E-mail	techsupport@de.rainbow.com
Telephone	(49) 89 32 17 98 0
Fax	(49) 89 32 17 98 50
Taiwan	
	Rainbow Technologies (Taiwan) Co.
E-mail	techsupport@tw.rainbow.com
Telephone	(886) 2 27155522
Fax	(886) 2 27138220
United Kingdom & Eire	
	Rainbow Technologies, Ltd.
E-mail	techsupport@uk.rainbow.com
Telephone	(44) 1932 579200
Fax	(44) 1932 570743

How to Report Problems

If you are having problems with the SentinelLM software, please go the following Web site: www.rainbow.com/tech/support.html.

Contacting Rainbow Technical Support

Chapter 1 - Introduction

SentinelLM is a license toolkit used by developers to add network and/or stand-alone licensing to their applications. The main components of the license management system are a protected application, a license file containing one or more license codes that authorize the use of the program, and a license server to receive and act on authorization requests. Access to the license server is made possible by an Application Program Interface (API). API functions are implemented in the SentinelLM Client Library which is linked with the application. For stand-alone applications, the license server is replaced with code that perform equivalent functions but without network access. In either case, an application program uses the same API set. Thus, the same version of an application can be delivered to end users that will run in either network or stand-alone mode.

Using the SentinelLM Application Library

The SentinelLM Client Library is used to integrate SentinelLM into your client application. There are different integration styles that offer varying degrees of functionality.

- The **Quick-API** is for use in applications that require only one license for each instance of the program. It is the simplest of the three API sets, and only requires the addition of two function calls. The first initializes contact with the license server and automatically updates the license code. This call is made during program initialization. The other is made at the end of the program to disable licensing and return the license code.
- The **Standard-API** offers a full spectrum of licensing models including the licensing of multiple features in a single application. It requires

adding only four function calls. The program begins by initializing the client library and requesting an authorization code. At the end of the program, calls are made to release the license code and clean up the client library. This method provides greater control and flexibility to the developer.

- The **Advanced-API** provides all the capabilities of the Standard-API plus additional server-side customization features. The Microsoft LSAPI defines a family of functions together with their parameters and return codes for use with applications running with a license server. A subset of LSAPI is included in the Advanced-API set, and is compliant with that standard. The additional functions that augment the Standard-API to form the Advanced-API can be grouped into one of several categories as follows:
 - Client Configuration functions, which allow an application to retrieve or change default values for such settings as port number, server name, broadcast interval, timeout interval, etc.
 - Client Query functions, which obtain a snapshot of the current status of the license server and the features it licenses.
 - Feature Query functions, which retrieve feature licensing information from the license manager such as name and version.
 - Client Utility functions, which provide client library capabilities such as retrieving the host names running SentinelLM, the names of the computers running the license server, and other facilities useful to certain specialized applications
 - Error handling functions, which make possible turning error handling on and off, registering custom error handlers, and printing error messages.

Licensing on Stand-alone and Networked Computers

Typically, your customer installs your application on one or more computers or on a file server that is connected to the network. They designate one computer on which the license server will run (the computer need not be the file or application server). To obtain a license authorization, the client applications communicate with the license server over the network as soon as they start up. Only when a valid license code is issued does the application actually run. Applications do not have to be network-aware. SentinelLM handles all network communication with the license server.

Stand-alone licensing is usually used with non-networked PCs running Windows. You can ship a single copy of your software to all your customers even if some of them have networking and some do not. By simply providing a different type of license code, you activate your software to run in stand-alone mode or in network mode.

Client API Example

This section describes and gives an example of how to integrate the SentinelLM client library functions into your application software. The example is independent of the platform on which it is run; i.e., it will execute either under Windows or UNIX. The purpose of the example is to illustrate the straightforward manner in which an application can be protected using SentinelLM.

The first call is **VLSInitialize()** and is made during program initialization. It has no parameters and will return a status of LS_SUCCESS upon successful completion. Once that has been done, you may proceed with your application.

The next function to call is **LSRequest()** which takes several parameters. These include the *PublisherName* which identifies your company, *FeatureName* which identifies your product, and *Version* which specifies the version number of that product. This information is contained in the license code, and must match before authorization to run the program can be given.

Client API Example

If you intend to license your application without separate feature sets, only one call to **LSRequest()** is needed. However, if you are planning to charge for different features, each feature will require a separate license, and one **LSRequest()** call will be required for each feature. The features will need different names for identification, and a separate version number may be associated with each one.

Note The license will be updated automatically for you at 80% of the lifetime of the license. A call to **LSUpdate()** is not necessary.

Once the application knows that the user has finished using a particular feature, it calls **LSRelease()** to return the license authorization to the license pool so other programs can use it. Finally, after all licenses have been released and the program is ready to terminate, a call is made to **VLScleanup()** to inform the library that any resources that it has allocated may be released.

Example

```
{
    LS_HANDLE    handle;

    /***** First Call, Initialize the client library *****/
    if (VLInitialize())
    {
        printf("Unable to initialize license server library.\n");
        VLScleanup();
    };

    /***** Second Call: Request a license *****/
    if (LS_SUCCESS != LSRequest (LS_ANY, PUBLISHER_NAME,
                                FEATURE_NAME, VERSION, NULL, NULL,
                                NULL, &handle))

    {
        printf("Unable to obtain a license.\n");
        VLScleanup();
    };

    printf("Successfully Obtained a license.\n");

    /***** Third Call: Return the license *****/
    (void) LSRelease(handle, LS_DEFAULT_UNITS, NULL);

    /***** Last Call: Clean Up *****/
```

```
    VLSleanup();  
}
```

Language Interfaces Supported

Different language interfaces are supported by SentinelLM to allow you to incorporate SentinelLM Application Library calls in applications coded in different programming languages. Among the language interfaces supported are Microsoft Visual C/C++, Microsoft Visual Basic, PowerBuilder, Borland C, and Delphi. Check the *\Intrface* directory in the SentinelLM directory, for the latest language interfaces.

Other interfaces are available, and will continue to become available over time. Contact your Rainbow representative for information on new interfaces and specific versions supported. If your application does not use one of the supported interfaces, see the *SentinelLM Developer's Guide* for information on using the SentinelLM-Shell, which encloses your application in a protective shell without modifying your application.

Special Use of Win32 for Generating Tools

Persons using the license generating capability of Sentinel LM are advised that the program to generate licenses is protected by one of Rainbow's hardware keys. Therefore, the program must be run under Windows, even when generating licenses to be used under UNIX. More generally, all users of the SentinelLM system are encouraged to install the Windows version of SentinelLM first in order to familiarize themselves with all of its features. This is recommended even if its eventual intended use is for UNIX environments.

Debugging Your Client Application

The SentinelLM Client Library has been written to intercept and log four different levels of events. The values for the different events in increasing order are:

```
VLS_TRACE_KEYS  
VLS_TRACE_FUNCTIONS  
VLS_TRACE_ERRORS  
VLS_TRACE_ALL
```

Any value implicitly includes logging not only its own event class, but the event classes associated with all lower values as well. A fifth value, **VLS_NO_TRACE**, is the default, and turns off all logging activity.

A developer can activate one of these levels by inserting a call to **VLSsetTraceLevel()** somewhere in the client code (see function description on See “Tracing SentinelLM Operation” on page 118.). The trace level will not be set until the function is called, making it possible to limit logging to certain portions of the client code only. A developer may choose to place more than one such call in the client code, and use different trace levels with each call in order to generate different logging profiles based upon what code is being executed.

To activate the logging feature, the SentinelLM server must be started using two switches as shown in the following example for Windows 95/98:

```
lserv9x stet -f my_trace.log
```

If the name of the log file is not fully qualified, the file will be created in the directory in which the client code is executing. The log file will be overwritten each time the client code is restarted.

Disabling Licensing

The macro **NO_LICENSE** in the *lserv.h* file can be set to completely disable licensing for debugging. This replaces all SentinelLM function calls with void

statements. Don't forget to re-enable licensing before preparing your application for shipment.

Disabling Licensing

Chapter 2 - Protecting Your Application with the Application Library

This chapter contains instructions and detailed information on:

- Client library functions
- Compiling your application

Using the SentinelLM Application Library to embed protection calls in your application source code provides the maximum amount of control, and allows you the most flexibility in using licensing models. This chapter contains information on using the SentinelLM Application Library to protect your application. For a full discussion of the SentinelLM Application Library calls, refer to other chapters in this book.

Adding APIs to Your Source Code

Once you determine which licensing model you are going to support, you can start to implement the code. In most cases, API calls remain the same for different licensing options. Licensing options are encoded in the license code so your program can adapt to future changes. Let's first take a look at how to quickly implement a sample program.

The first call you want to make in your application during its initialization is **VLSInitialize()**.

It has no parameters and will return a `LS_SUCCESS` status upon success. You should proceed with your application after this call.

The next function you want to call is **LSRequest()**.

This API takes several parameters. *PublisherName* identifies your company. *FeatureName* identifies your product and *version* identifies the version number for that product. This information must match what you give the license code generator when you generate a license code for a user.

Application Identification

Each successful request returns a handle which identifies the dialog set up between the licensed application and the license server. This handle should be used in all dialog or connection library calls.

This architecture enables a licensed application to set up multiple connections with the license server and request multiple licenses. The license server treats each request independently.

If you are going to license your application without separate feature sets, you will only need to call **LSRequest()** once. However, if you are planning to license and charge based on features, you will need to call **LSRequest()** once for each feature. These features will need to have a different name for identification. Each feature can have a version associated with it.

If you choose to implement license queuing, you may want to use the **VLSqueuedRequest()** call instead. Use the *requestFlag* parameter to control normal and queued license requests. For details, see “Chapter 6 - License Queuing API” on page 227.

Automatic License Server Detection

If you provide no information to SentinelLM on the location of a license server, a SentinelLM-licensed application uses a broadcast mechanism to determine the existence of an active SentinelLM license server on the local subnet, and automatically establishes a dialog with the first license server with a license for the given feature and version.

You can prevent a network broadcast and instead direct the application to specific license servers in the following ways:

- If you set the **LSFORCEHOST** environment variable to a particular license server, SentinelLM initiates contact with that license server only. **LSFORCEHOST** overrides the **LSHOST** environment variable or the *LSHOST/lshost* file.
- If no **LSFORCEHOST** environment variable is set, SentinelLM looks for an **LSHOST** environment variable or *LSHOST* (or *lshost*) file, which contains a list of one or more license servers. Example: **LSHOST** = server1;server2;server3 where serverX can be *hostname*, IP or IPX address of the license server. If SentinelLM cannot find an **LSHOST** environment variable or *LSHOST/lshost* file, or if it cannot find the license servers specified in that variable or file, SentinelLM uses its broadcast mechanism to find any license server on the local subnet which contains the desired feature/version.

When there are multiple SentinelLM license servers with different license files, licensed applications may query the wrong license server for permission to run. If a licensed application contacts a license server that does not have any free licenses, the application will not receive a license and other non-redundant license servers that have available licenses for the feature/version will not automatically be contacted. The SentinelLM client library will return an error, and/or the application will terminate.

This situation can be avoided by using the SentinelLM client library call, **VLSDiscover()**, to locate all of the SentinelLM license servers on the local subnet, and query each of them individually for a license. You will need to call **VLSetContactServer()** to initiate contact with each license server. Another option is to use the **LSHOST** environment variable or the *LSHOST/lshost* file. Using **VLSDiscover()** may be preferable in that it protects end users from having to set environment variables or be concerned with additional files.

Although SentinelLM uses the broadcast mechanism, network impact is minimal. It is used only on the first **LSRequest()** call and only on the local subnet. It is optimized to use minimal bandwidth.

If you are using the combined stand-alone and networked mode library (dual mode), The **LSRequest()** API will first try to look for a stand-alone license. If a stand-alone license does not exist on the client machine, it will perform a broadcast on the network for a license server. Your application should check the return code and continue to execute if **LSRequest()** returns **LS_SUCCESS**. Once **LSRequest()** is called, the client library will automatically renew the license acquired before it expires. This frees the application from worrying about renewing the license on a rigid time schedule. However, it is recommend that you call **LSUpdate()** periodically to make sure that the license renewal is successful and the license server is still up and running. **LSUpdate()** is not required for stand-alone licensing but there are no side effects from including it so your application works in both stand-alone and networked mode.

Note If you choose to call **LSUpdate()** to manually renew the license, you must call **LSUpdate()** within the lifetime of the license. Be absolutely certain to call **VLSdisableLocalRenewal()** after **VLSinitialize()**, but before **LSRequest()**.

The licensing is done once these functions are called and your application can proceed with its normal functionality.

After your application decides that a particular feature is no longer required by the user, it can call **LSRelease()** to release the license back to the license pool so other programs can use it.

When your application quits, you should call **VLSCleanup()** to let the client library take care of releasing any resources it allocates.

Special Licensing Cases

There might be cases where you want to take advantage of built-in support for special licensing options. For example, a shared license allows more than one application/component to share the same license. This is useful for logically grouping similar features which you do not intend to charge the user for separately. For more details, refer to **VLSsetSharedId()** and **VLSsetSharedIdValue()** in “Chapter 3 - SentinelLM Client API” on page 23.

Another example of special licensing is the held license. If your program is short-lived, you can use **VLSsetHoldTime()** to set the checkout time for a license. This allows users to reclaim a license when running a short-lived, frequently used application, such as compilers.

You may want to manually update the license yourself. To do so, you need to call:

- **VLSInitialize()**
- **VLSdisableLocalRenewal()**
- **LSRequest()**
- **LSUpdate()** (You will need to create your own timer to insure the update occurs prior to the license lifetime expiring.)
- **LSRelease()**
- **VLSCleanup()**

Linking with the Correct Library

Both dynamic linked libraries and static linked libraries are available for 32-bit Windows applications. We recommend using the combined stand-alone and network (dual mode) library if possible. This allows your application to request a license either on a stand-alone computer or from a remote license server.

Windows Static Linked Libraries

In addition to using the correct static libraries, you must also link the following libraries (which are included in your Windows development environment) into your application: *wsock32.lib*, *rptcrt4.lib*, *netapi32.lib*, *shell32.lib*, *ole32.lib*, *oleaut32.lib*, *uuid.lib*, *odbc32.lib*, *odbccp32.lib*, *wsock32.lib*, *rpcrt4.lib*, and *netapi32.lib*. Please see the *samples32.mak* make file in the SentinelLM `\demo\MsvcDev\Samples` directory for details on how to link your application with the SentinelLM client library.

Linking with the Correct Library

Note The libraries in the following tables are only available if you have purchased the appropriate options (i.e., Developer option).

In the static libraries folder, you will find the following files:

Table 2-1: Windows Static Libraries

Library	Description
Isapiw32.lib	Dual network and stand-alone client library for Windows applications. This library allows you either to access the stand-alone license locally or acquire a license from a remote license server over the network.
Issrv32.lib	This library is the same as <i>Isapiw32.lib</i> .
Isclws32.lib	The network client library for Windows applications. This library allows your application to acquire licenses via network only.
Isnnet32.lib	The stand-alone client library for Windows applications. This library allows you to acquire stand-alone licenses on a local computer only.

Windows Dynamic Linked Libraries and Import Libraries

Table 2-2: Windows Dynamic Libraries and Import Libraries

Library	Description
<i>Isapiw32.dll</i>	Dual network and stand-alone client library for 32-bit Windows applications. This library allows you either to access the stand-alone license locally or acquire a license from a remote license server over the network.
<i>Isapiw32.lib</i>	This library is the import library for <i>Isapiw32.dll</i> , (Microsoft format).
<i>Issrv32.dll</i>	This library is the same as <i>Isapiw32.dll</i> .
<i>Isclws32.dll</i>	The network client library for 32-bit Windows applications. This library allows your application to acquire licenses via network only. If you copy this library to <i>Isapiw32.dll</i> , you may use the <i>Isapiw32.lib</i> import library supplied with the installation.
<i>Isnnet32.dll</i>	The stand-alone client library for 32-bit Windows applications. This library allows you to acquire stand-alone licenses on a local computer only. If you copy this library to <i>Isapiw32.dll</i> , you may use the <i>Isapiw32.lib</i> import library supplied with the installation.

UNIX Libraries

You can choose one of three libraries to link with:

- *libls.a*—The network licensing client library, not relevant for stand-alone licensing.
- *libnonet.a*—Library for stand-alone mode licensing. Does not have any network awareness at all. Does not require a license server in order to run.
- *liblssrv.a*—Integrates the functionality of *libls.a* and *libnonet.a*. At run-time, it switches to either *libls.a* behavior or *libnonet.a* behavior, depending upon the environment variable, LSHOST. If LSHOST is set to NO-NET or no-net, the linked application will go into stand-alone mode, otherwise it will stay in network mode.

libls.a and *libnonet.a* will result in smaller executables but are more limited and less flexible in functionality and behavior than *liblssrv.a*.

To specify the library best for you, edit the *Makefile* in the *examples* directory of the *SentinelLM* shipment. Change the value of the macro, `LICENSE_LIBS`. By default, it specifies the library *libls.a* to link with, via `-lls`. Change it to `-lnonet` or `-llssrv`.

Now you are ready to compile and link a licensed application. Try relinking the sample applications and examples in the *examples* directory.

Notes on Security

SentinelLM uses proprietary, advanced anti-hacking techniques to safeguard against malicious attempts to alter its intended mode of use.

SentinelLM uses proprietary encryption and decryption algorithms for all network communication to guard against wire tapping. All messages are time-stamped to thwart attempts at replaying encrypted messages in response to authorization requests. Critical licensing information required by the license server is encrypted to the network licenses by a separate set of encryption algorithms.

You can add an additional layer of security with your own encryption and decryption algorithms to the network licenses.

In addition to customizing encryption algorithms you can use the challenge-response mechanism to authenticate client-server communications. See “Chapter 3 - SentinelLM Client API” on page 45 and *SentinelLM Developer’s Guide* (“License Code Generator chapter”) for details.

Finally, developers can strengthen their legal position if their license agreement includes the following statement:

“Removal, emulation, or reverse engineering of all or any part of this product or its protection constitutes an unauthorized modification to the product and is specifically prohibited. Nothing in this license statement permits you to derive the source or

assembly code of files provided to you in executable or object formats.”

Such language closes major loopholes in the copyright laws of many nations.

Protecting Against Time Tampering

Software-based license protection schemes may break down if the end user changes the system time. The SentinelLM license server is configured to detect tampering of the system clock.

The SentinelLM license server will verify at start up and periodically thereafter, whether the system clock has been altered. If it detects evidence of such tampering, it discards licenses with an expiration date. You also have the option of implementing your own functionality to detect system clock changes. Please see “Appendix B - Customization Features” on page 261.

Using a Custom Locking Code

A custom locking code requires the following components:

1. A rebuilt license server that uses the custom ID function. For example, *lserv9x* or *lservnt*.
2. A rebuilt *echoid.exe* that uses the same custom ID function as the license server.
3. A modified client application.

Step 1 - Rebuilding License Server

Compiler Required

A Microsoft Visual C++ 6.0 compiler is required.

Note It is possible to use other compilers, but instructions below are for Microsoft Visual C++ compiler. Please contact Rainbow if you are using another compiler and require assistance.

Files Required

The following files are required in rebuilding the license server:

Table 2-3: Files required to rebuild the license server

File Name	Description
<i>ServerInit.cpp</i>	C++ source file containing re-definition of VLServerVendorInitialize() function.
<i>CustomHostID.cpp</i>	C++ source file containing custom locking code definition.
<i>CustomHostID.h</i>	C++ include file containing custom locking code prototype.
<i>Ismainwa.c</i>	C source file containing entry point to license server application.
<i>Iserv.h</i>	SentinelLM include file installed during SentinelLM installation.
<i>Iserv95.res</i>	Resource file for license server application.
<i>Iserv9x.lib</i>	SentinelLM static library installed during SentinelLM installation.
<i>Iserv9x.dsp</i>	Microsoft Visual C++ 6.0 project file.
<i>Iserv9x.dsw</i>	Microsoft Visual C++ 6.0 workspace file.

Note *Iserv95.res*, *Iserv9x.lib*, *Iserv9x.dsp*, and *Iserv9x.dsw* files can be slightly different for NT version.

Required Changes to Server Source Code

A SentinelLM license server with custom locking code will differ from a default license server because the **VLServerVendorInitialize()** function is “overloaded” so that it will call the function **VLSetHostIdFunc()**. The **VLServerVendorInitialize()** function is called during server startup for both

default license servers and custom license servers, but the default version does not call **VLSsetHostIdFunc()** function.

The **VLSsetHostIdFunc()** function accepts as a parameter the name of the function which will return the custom locking code. This locking code must be calculated in a consistent long value; not a random value. You are free to implement any algorithm in order to produce the locking code, as long as the algorithm generates a reproducible value.

The **VLServerVendorInitialize()** function is automatically called during server startup. However, for servers that initialize custom locking code, **VLServerVendorInitialize()** function is overloaded (redefined) to call **VLSsetHostIdFunc(functionName)**. *functionName* is the name of the custom locking code function and *GetCustomLockCode* is the name of the custom locking code function, both described above. *GetCustomLockCode* is provided only as an example name.

Steps to Rebuilding the License Server

1. Obtain a zip file from Rainbow Technologies that contain all the necessary files. Please see “Files Required” on page 18. Unzip the zip file into a directory of your choice.
2. Open the workspace file corresponding to the customized license server project. For example, if you have a 9x license server, then you will need to open the *lserv9x.dsw* project file.
3. Modify the source code. See “Required Changes to Server Source Code” on page 18.
4. Choose **Rebuild All** from the Build menu.

Step 2 - Rebuilding *echoid.exe*

In order to add a license locked to a custom criteria, a rebuilt *echoid.exe* is also required. The rebuilt *echoid.exe* will be used to produce a fingerprint relative to the custom locking code function. This fingerprint can then be used to generate locked licenses that utilize the custom locking criteria.

Compiler Required

A Microsoft Visual C++ 6.0 compiler is required.

Note It is possible to use other compilers, but instructions below are for Microsoft Visual C++ compiler. Please contact Rainbow if you are using another compiler and require assistance.

Files Required for *echoid.exe*

The following files are required in rebuilding *echoid.exe*:

Table 2-4: Files required to rebuild *echoid.exe*

File Name	Description
<i>CustomHostID.c</i>	C source file containing custom locking code definition.
<i>CustomHostID.h</i>	C include file containing custom locking code prototype.
<i>echoid.c</i>	C source file containing logic for generating fingerprints. Notice, this file will be modified to call the custom locking code function.
<i>Isngen.h</i>	SentinelLM include file installed during SentinelLM installation.
<i>Iserv.h</i>	SentinelLM include file installed during SentinelLM installation.
<i>Isapiw32.lib</i>	Import library for Win32 run-time <i>DLL</i> that is installed during SentinelLM installation.
<i>echoid.dsp</i>	Microsoft Visual C++ 6.0 project file
<i>echoid.dsw</i>	Microsoft Visual C++ 6.0 workspace file

Required Changes to *echoid.exe*

Rebuilding *echoid.exe* only requires a slight modification to the source code. Before calling **VLGetMachineID()**, call **VLSetHostIdFunc(functionName)**, where *functionName* is the name of the custom locking code function.

Again, using *GetCustomLockCode* as the name of the custom lock code function, the sequence of function calls will be as follows:

- rest of *echoid* source
- **VLSsetHostIdFunc**(*GetCustomLockCode*)
- **VLSgetMachineID**()
- rest of *echoid* source

Steps to Rebuilding *echoid.exe*

1. Obtain a zip file from Rainbow Technologies that contain all the necessary files. Please see “Files Required for *echoid.exe*” on page 20. Unzip the zip file into a directory of your choice.
2. Open the workspace file corresponding to the customized license server project.
3. Modify the source code. See “Required Changes to Server Source Code” on page 18.
4. Choose **Rebuild All** from the Build menu.

Step 3 - Modifying Client Application

The client application should also make a call to **VLSsetHoldIdFunc**(). This function call needs to be performed before a license request is issued. In doing this, a developer guarantees that both the client-locked licenses and server-locked licenses will be handled. Also, the client application will not be adversely affected by this function call if the default license server, rather than the custom license server, is used. Please see “Required Changes to Server Source Code” on page 18.

Overall Process of Using a Rebuilt License Server and Rebuilt *echoid.exe*

1. Decide on an algorithm for generating custom locking code. Notice, this locking code needs to be a reproducible long value.
2. Rebuild license server. See “Step 1 - Rebuilding License Server” on page 17.
3. Rebuild *echoid.exe*. See “Step 2 - Rebuilding *echoid.exe*” on page 19.
4. Edit *echoid.dat* so that the custom locking criteria is a criteria mask. This step may not be needed if custom locking criteria mask is the default mask in the rebuilt *echoid.exe*.
5. Execute the rebuilt *echoid.exe*.
6. Generate server-locked licenses with the fingerprint obtained from the rebuilt *echoid.exe* as the primary criteria.
7. Add licenses to the rebuilt license server via **lslic** or via the license server configuration file *lservrc*.
8. Modify the client application and rebuild it. See “Step 3 - Modifying Client Application” on page 21.
9. Execute the client application.

Chapter 3 - SentinelLM Client API

Introduction

Using the SentinelLM client API, the following integration styles of varying complexity are supported:

- The simplest style requires adding only two function calls to the application program. During program initialization, a call is made to **VLslicense()** to initialize contact with the license server and automatically update the license code. Then, during program termination, a call is made to **VLsdisableLicense()** to disable licensing and return the license code. Any additional communication required with the license server is automatically handled by the client library.
- A style providing greater flexibility requires the use of four different calls within the application program. During program initialization, calls are made to **VLsinitialize()** to initialize the client library and then to **LSRequest()** to request an authorization license code. **VLsinitialize()** or **VLsqueuedRequest()** should be called only once. During program termination, calls are made to **LSRelease()** to release the authorization license code and then to **VLScleanup()** to clean up the client library. **VLScleanup()** should be called only once.
- The full featured function interface is recommended when using advanced licensing features. This interface is compliant with the industry LSAPI standard. This style uses the API calls described in the intermediate style above, but is augmented by calls to other library functions.

This chapter describes all the function calls available in the SentinelLM Application Program Interface (API), which includes the industry standard, LSAPI. All function calls, return codes, and data types that begin with the LS prefix are part of the LSAPI standard. The APIs that begin with the VLS prefix are the SentinelLM extensions that make licensing easier and more powerful.

All function calls return the status code, `LS_SUCCESS`, if successful or a specific error code indicating the reason for failure otherwise. For more information about applicable error codes, see “Error Handling” on page 113.

On Win32 and UNIX computers, there are three sets of client libraries:

- **Stand-alone:** For stand-alone operation without requiring a network license server. The functions not supported in the stand-alone client library are actually present but do not perform any meaningful action. You do not need to make any source code changes when moving from a SentinelLM network client library to a stand-alone client library.
- **Network:** For any operation requiring a network license server.
- **Integrated:** For both stand-alone and networked operations. We recommend you link with this library if you would like to support both stand-alone and network license management. Even if you are not sure if you need to support both, you may still consider using this library for future expansion. Applications linked with this client library can obtain stand-alone licenses from a local file or network licenses from a network license server. There are special control flags enabling developers to customize the behavior of choosing between stand-alone and network libraries.

Multiple license codes can be requested within an application for a feature and feature version. Each license code must be released and updated separately as the license server treats these license codes as separate clients. A handle that uniquely identifies the license code will be returned for each **LSRequest()** call using the argument, *lshandle*. This handle is also used in other SentinelLM function calls.

All of the SentinelLM client libraries are thread safe. However, license handles may not be shared or passed from one thread to another. We recommend

spawning a thread (or using the main application thread) and perform all SentinelLM functions for that single thread.

Available license code generation function calls can be separated into the following categories:

- Basic client licensing functions
- Challenge-response
- Client configuration
- Client query
- Feature query
- Client utility
- Error handling
- Tracing SentinelLM operation
- Redundancy
- Queuing

Basic Client Licensing Functions

Quick Client Licensing Functions

The following table summarizes the quick client functions:

Table 3-5: Quick Client Licensing Functions

Function	Description
VLSlicense()	Performs single-call licensing.
VLSdisableLicense()	Disables single-call licensing.

*VL*License()

VL*License*()

Client	Server	Static Library	DLL
✓		✓	✓

Initializes contact with the license server requires authorization and automatically updates the license.

Syntax

```
LS_STATUS_CODE VLLicense (  
    unsigned char    *featureName,  
    unsigned char    *version,  
    LS_HANDLE        *lshandle;
```

Argument	Description
<i>featureName</i>	Name of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 24 characters.
<i>version</i>	<i>Version</i> of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 11 characters.
<i>lshandle</i> (out)	This handle must be used to release this license code by calling VLSdisableLicense() . Space must be allocated by the caller.

Note Length limitations exist on feature name and version depending on the type of license you want to issue to your customer. See the *SentinelLM Developer's Guide* for details.

Description This function obtains an authorization license using **LSRequest()** and then automatically updates the license after 80% of the license lifetime has passed, using the **LSUpdate()** function. This function uses timers (SIGALRM on UNIX) to update a license periodically. You should not update that license yourself using **LSUpdate()** or any other license renewal function. When you wish to release the license (terminate the automatic updates), you must use the API function **VLSdisableLicense()** which removes the timer, and releases the

license. If you release the license using **LSRelease()** and your application continues to run, the timer will keep trying to renew an invalid license since it does not know that you have released the license yourself.

On UNIX, since there is only one timer available to each running application, there will be a conflict if your application wishes to use timers *and* use **VLSlicense()** at the same time. To accommodate multiple simultaneous uses of a single timer, the SentinelLM API provides a generalized version of the timer functions.

From one instance of an application, you can call **VLSlicense()** only once. **VLSlicense()** can automatically update only a single handle. Subsequent calls to **VLSlicense()** will fail.

Note This function is available on most UNIX platforms. This function may not be available on platforms that do not support a timer event or a time signal.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL
VLS_CALLING_ERROR	<i>lshandle</i> is NULL. Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_NO_LICENSE_GIVEN	Invalid handle specified. Handle is already released and destroyed from previous license operations.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for requested feature to grant a license.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
LS_LICENSE_EXPIRED	License has expired.

*VL*License()

VLS_APP_NODE_LOCKED	Requested feature is node locked, but request was issued from an unauthorized machine.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_ELM_LIC_NOT-ENABLE	The license was converted using the license conversion utility (from a 5.x license), but the DLT process is not running.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_MAJORITY_RULE_FAILURE	Majority rule failure prevents token from being issued.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_BAD_SERVER_MESSAGE	Message returned by license server could not be understood.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
LS_NONETWORK	Failed to initialize Winsock wrapper. (Only applicable if using network-only library.) Generic error indicating network failure.

VLSdisableLicense()

VLS_INTERNAL_ERROR	Failure occurred in setting timer. (Timer is only attempted to be set if timer is available for platform and if license requires timer for updates.)
--------------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSdisableLicense()

Client	Server	Static Library	DLL
✓		✓	✓

This function disables single-call licensing and returns the license code.

Syntax LS_STATUS_CODE VLSdisableLicense (
 LS_HANDLE **lshandle*);

Argument	Description
<i>lshandle</i>	The handle which had been obtained earlier by a call to VLSlicense() .

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>lshandle</i> is an ambiguous handle; it is not exclusively active or exclusively queued.
VLS_ALL_UNITS_RELEASED	All units have already been released.
VLS_RETURN_FAILED	Generic error indicating that the license could not be returned.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.

VLSDisableLicense()

VLS_NO_SERVER_RESPONSE	Communication with license server timed out.
VLS_BAD_SERVER_MESSAGE	Message returned by server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_INTERNAL_ERROR	An error occurred with respect to the serialization/customization of SentinelLM files.

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

Standard Client Licensing Functions

The following table summarizes the standard client functions:

Table 3-6: Standard Client Licensing Functions

Function	Description
VLSInitialize()	Initializes the client library.
LSRequest()	Requests an authorization license code.
LSRelease()	Releases an authorization license code.
VLScleanup()	Called when finished using the client library.
LSUpdate()	Called periodically to renew the license code and inform the license server that it is alive.

VLSInitialize()

Client	Server	Static Library	DLL
✓		✓	✓

Initializes the client library.

Syntax LS_STATUS_CODE VLSInitialize (void);

This function has no arguments.

Description This call must be made before any SentinelLM function can be called.

Note Applications that call the UNIX standard-C library function, **fork()**, generally follow this call with an **exec()** function call to re-initialize all global values. For some applications, however, this may be undesirable. In such cases, issue the call before the first **LSRequest()** call and after each **fork()** call. This call is not necessary for applications that do not use **fork()** or **exec()** after forking. Calling this function unnecessarily does not have any negative side effects.

LSRequest()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
LS_NONETWORK	Failed to initialize Winsock wrapper. (Only applicable if using network-only library.)

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **VLScleanup()**

LSRequest()

Client	Server	Static Library	DLL
✓		✓	✓

Requests an authorization license code from the license server.

Syntax LS_STATUS_CODE LSRequest (
 unsigned char *licenseSystem,
 unsigned char *publisherName,
 unsigned char *featureName,
 unsigned char *version,
 unsigned long *unitsReqd,
 unsigned char *logComment,
 LS_CHALLENGE *challenge,
 LS_HANDLE *lhandle;

Argument	Description
<i>licenseSystem</i>	Unused. Use LS_ANY as the value of this variable. LS_ANY is specified to indicate a match against installed license systems.

Argument	Description
<i>publisherName</i>	A string giving the publisher of the product. Limited to 32 characters and cannot be NULL. Company name and trademark may be used.
<i>featureName</i>	Name of the feature for which the licensing code is requested. May consist of any printable characters and cannot be NULL. Limited to 24 characters.
<i>version</i>	Version of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 11 characters.
<i>unitsReqd</i>	The number of licenses required. The license server verifies that the requested number of units exist and may reserve those units, but no units are actually consumed at this time. The number of units available is returned. If the number of licenses available with the license server is less than the requested number, the number of available licenses will be returned using <i>unitsReqd</i> . If <i>unitsReqd</i> is NULL, a value of 1 unit is assumed.
<i>logComment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.
<i>challenge</i>	The challenge structure. If the challenge-response mechanism is not being used, this pointer <i>must</i> be NULL. The space for this structure must be allocated by the calling function. The response to the challenge is provided in the same structure, provided a license was issued, i.e., provided the function LSRequest() returned LS_SUCCESS. For details of the challenge-response mechanism and how to use it effectively, see page 45.
<i>lshandle</i>	The handle for this request is returned in <i>lshandle</i> . This handle must be used to later update and release this license code. A client can have more than one handle active at a time. Space for <i>lshandle</i> must be allocated by the caller.

Description This function is used by the application to request licensing resources to allow the product to execute. If the valid license is found, the challenge-response is computed (if applicable) and LS_SUCCESS is returned. The challenge-response

LSRequest()

is computed if a non-NULL value is passed for the *challenge* argument. At minimum, the *PublisherName*, *ProductName*, and *Version* strings are used to identify matching license(s). When the application has completed execution, it must call **LSRelease()** to release the license resource.

If the number of units required is greater than the number of units available, then **LSRequest()** will not grant the license.

Every client should complete this call successfully before commencing execution of the application or the feature.

If the default error handler is not used, the client application must check the code returned by the **LSRequest()** call and should continue only if LS_SUCCESS is returned. The default error handler will exit the application on error.

Note If queuing is desired, you must use **VLSQueuedRequest()** instead of **LSRequest()**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>lshandle</i> is NULL. <i>challenge</i> argument is non-NULL, but cannot be understood. Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL.
VLS_NO_LICENSE GIVEN	<i>unitsReqd</i> is zero <i>lshandle</i> is not a valid handle. License is only available at license server that does not match mode settings, e.g. network license available when stand-alone mode, etc.
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for requested feature to grant license.

LS_LICENSE_EXPIRED	License is expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license expired or trial license usage exhausted.
VLS_APP_NODE_LOCKED	Requested feature is node locked, but request was issued from unauthorized machine.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_VENDORIDMISMATCH	The vendor identification of requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_MAJORITY_RULE_FAILURE	Majority rule failure prevents token from being issued.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER-FILE	No license server has been set and unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_INTERNAL_ERROR	Failure occurred in setting timer. (Timer is only attempted to be set if timer is available for platform and if license requires timer for updates.)

LSRelease()

VLS_ELM_LIC_NOT_ENABLE	The license was converted using the license conversion utility (from a 5.x license), but the DLT process is not running.
------------------------	--

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also “Challenge-response” on page 45, **VLSsetTimeoutInterval()**, **VLSqueuedRequest()**.

LSRelease()

Client	Server	Static Library	DLL
✓		✓	✓

Requests that the license server release licenses associated with a handle.

Syntax LS_STATUS_CODE LSRelease (
 LS_HANDLE *lshandle*,
 unsigned long **units_consumed*,
 unsigned char LSFAR **log-comment*;

Argument	Description
<i>lshandle</i>	The handle returned by the corresponding LSRequest() .
<i>units_consumed</i>	Number of units released.
<i>log_comment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.

Description Releases the license(s) associated with *lshandle*, allowing them to be immediately used by other requesting applications. For a shared license, all client applications must release their licenses before the license server marks the license as available.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>lshandle</i> is an ambiguous handle; it is not exclusively active or exclusively queued.
VLS_RETURN_FAILED	Generic error indicating that the license could not be returned.
VLS_ALL_UNITS_RELEASED	All units already released.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLScleanup()

Client	Server	Static Library	DLL
✓		✓	✓

Cleans up the client library.

Syntax LS_STATUS_CODE VLScleanup (void);

This function has no arguments.

LSUpdate()

Description After all SentinelLM calls are done and before exiting, you must call this function. This function may not be called if the application is being protected using the Quick-API. Calling **VLScleanup()** after calling **VLSdisableLicense()** can produce unpredictable results.

Returns The status code, LS_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **VLSinitialize()**

LSUpdate()

Client	Server	Static Library	DLL
✓		✓	✓

Once an authorization license has been received, the client must call **LSUpdate()** periodically to renew its license and inform the license server that it is alive, if automatic renewal is disabled.

Syntax

```
LS_STATUS_CODE LSUpdate (  
    LS_HANDLE      lshandle,  
    unsigned long   *unused1,  
    long           *unused2,  
    unsigned char   *unused3,  
    LS_CHALLENGE   *unused4;
```

Argument	Description
<i>lshandle</i>	This must be the handle previously returned by the corresponding LSRequest() call.
<i>unused1</i>	Unused. Use LS_DEFAULT_UNITS as the value.
<i>unused2</i>	Unused. Use NULL as the value.
<i>unused3</i>	Use NULL as the value.
<i>unused4</i>	Use NULL as the value.

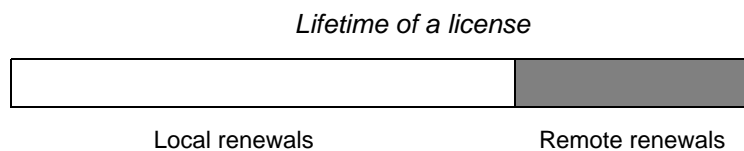
Description Currently the client library defaults to automatic (local) license renewal. You do not need to call this API unless you disable local license renewals.

If local license renewals are disabled, the client must call **LSUpdate()** periodically to renew its license and inform the license server of its continued need for a license. However, you should do this only in rare cases where renewals are critical or the system load is uncertain.

If you do call **LSUpdate()** manually to verify the client is still attached to the license server, you must disable local renewals before calling **LSUpdate()**.

Local Vs. Remote License Renewal

In order to reduce network traffic and communication overhead, SentinelLM checks whether the license lifetime is close to expiration, and contacts the license server only if it is about to expire. Otherwise, it returns the success code. This is called local renewal. There is no appreciable overhead in renewing a license too frequently, and non-timer based renewal schemes can use this feature to their advantage.



That part of the lifetime of a license which results in the renewal of the license by the license server is called the remote renewal period. Its default value is 80% of the license lifetime. However, for best results, the use of timers to optimally control the frequency of renewal calls is recommended.

Note Auto timers will not work in a Win32 console application.

Timer-based renewal schemes are not required to use local renewals at all. The period of the timer can be such that **LSUpdate()** calls occur only during the remote renewal period of the license.

The SentinelLM API also provides the function, **VLDisableLocalRenewal()**, which forces all future **LSUpdate()** requests to go to the license server.

LSUpdate()

VLSgetRenewalStatus() provides information on whether the last successful update was local or remote. See page 73 for these and other related function calls.

Note that **LSUpdate()** is a signal-safe function, so that it can be called from signal handlers and can be interrupted by other signal handlers without any known ill effects. Other functions are not guaranteed to be signal-safe.

Returns

The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>lshandle</i> is a queued handle. Cannot use LSUpdate() to update a queued handle. <i>challenge</i> argument is non-NULL, but cannot be understood.
VLS_NO_LICENSE GIVEN	Generic error indicating that license was not updated.
LS_LICENSETERMINATED	Cannot update the license because the license has already expired.
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_LICENSE_EXPIRED	License has expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license expired or trial license usage exhausted.
VLS_FINGERPRINT_MISMATCH	Client-locked; locking criteria does not match.
VLS_APP_NODE_LOCKED	Feature is node locked, but the update request was issued from an unauthorized machine.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_VENDORIDMISMATCH	The vendor identification of requesting application does not match the vendor identification of the feature for which the license server has a license.

VLS_INVALID_DOMAIN	The domain of the license server is different from that of client.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_BAD_SERVER_MESSAGE	Message returned by license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_ELM_LIC_NOT_ENABLE	The license was converted using the license conversion utility (from a 5.x license), but the DLT process is not running.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also VLSbatchUpdate(), VLSsetRemoteRenewalTime(), VLSgetRenewalStatus(), VLSdisableLocalRenewal(), VLSenableLocalRenewal(), VLSisLocalRenewalDisabled(), VLSsetTimeoutInterval()

Advanced Client Licensing Functions

The following table summarizes the advanced client functions:

Table 3-7: Advanced Client Licensing Functions

Function	Description
VLSinitialize()	Initializes the client library.
VLSrequestExt()	Requests an authorization license.
VLSreleaseExt()	Releases an authorization license.

VLInitialize()

Table 3-7: Advanced Client Licensing Functions (Continued)

Function	Description
VLScleanup()	Called when finished using the client library.
VLSbatchUpdate()	Updates several license codes at once.

VLInitialize()

See “VLInitialize()” on page 31.

VLRequestExt()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLRequestExt (  
    unsigned char    *licenseSystem,  
    unsigned char    *publisherName,  
    unsigned char    *featureName,  
    unsigned char    *version,  
    unsigned long    *unitsReqd,  
    unsigned char    *logComment,  
    LS_CHALLENGE    *challenge,  
    LS_HANDLE        *lshandle,  
    VLSserverInfo    *serverInfo;
```

Argument	Description
<i>licenseSystem</i>	Unused. Use LS_ANY as the value of this variable.
<i>publisherName</i>	A string giving the publisher of the product. Limited to 32 characters. Company name and trademark may be used.
<i>featureName</i>	Name of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 24 characters.

Argument	Description
<i>version</i>	Version of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 11 characters.
<i>unitsReqd</i>	The number of licenses required. If the number of licenses available with the license server is less than the requested number, the number of available licenses will be returned using <i>unitsReqd</i> . If <i>unitsReqd</i> is NULL, a value of 1 unit is assumed.
<i>logComment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.
<i>challenge</i>	The <i>challenge</i> structure. If the challenge-response mechanism is not being used, this pointer <i>must</i> be NULL. The space for this structure must be allocated by the calling function. The response to the <i>challenge</i> is provided in the same structure, provided a license code was issued, i.e., provided the function LSRequest() returned LS_SUCCESS. For details of the challenge-response mechanism and how to use it effectively, see page 45.
<i>lshandle</i>	The handle for this request is returned in <i>lshandle</i> . This handle must be used to later update and release this license. A client can have more than one handle active at a time. Space for <i>lshandle</i> must be allocated by the caller.
<i>serverInfo</i>	This information is passed to the license server for use in server hook functions. See "VLSeventAddHook()" on page 263.

Description Use **VLSrequestExt()** when using license server hooks. Before calling **VLSrequestExt()**, you must call **VLSinitServerInfo()**. (See "VLSinitServerInfo()" on page 65.)

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLRequestExt()

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL
VLS_CALLING_ERROR	<i>lshandle</i> is NULL <i>challenge argument</i> is non-NULL Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_NO_LICENSE GIVEN	<i>unitsReqd</i> is zero <i>lshandle</i> is not a valid handle.
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for requested feature to grant license.
LS_LICENSE_EXPIRED	License has expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license expired or trial license usage exhausted.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_VENDORIDMISMATCH	The vendor identification of requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_MAJORITY_RULE_FAILURE	Majority rule failure prevents token from being issued.

VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER-FILE	No license server has been set and unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_INTERNAL_ERROR	Failure occurred in setting timer. (Timer is only attempted to be set if timer is available for platform and if license requires timer for updates.)
VLS_ELM_LIC_NOT_ENABLE	The license was converted using the license conversion utility (from a 5.x license), but the DLT process is not running.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also “Challenge-response” below, **VLSeventAddHook()**

Challenge-response

The challenge-response mechanism can be used by a licensed application to authenticate the license server.

VLsrequestExt()

Syntax

```
typedef struct {
    unsigned long    ulReserved;
    unsigned long    ulChallengedSecret;
    unsigned long    ulChallengeSize;
    unsigned char    ChallengeData[30];
} CHALLENGE;

typedef CHALLENGE LS_CHALLENGE;

typedef struct {
    unsigned long    ulResponseSize;
    unsigned char    ResponseData[16];
} CHALLENGERESPONSE;
```

Member	Description
<i>ulReserved</i>	LSAPI requires this to be set to 0.
<i>ulChallengedSecret</i>	The index of the secret which the client application wishes the license server to use in computing its response to this challenge. This value may range from 1 to the number of secrets provided. The actual secrets are provided to the license server through the license code produced using the code generator and can include characters in the range A - Z, and 1 - 9.
<i>ulChallengeSize</i>	Number of characters in <i>ChallengeData</i> . This value cannot be 0.
<i>ChallengeData</i>	The actual string that is used in challenging the license server. (Mentioned as data in the explanation above.) This is a string of at most 30 characters, each of which can have any values, including 0.
<i>ulResponseSize</i>	Number of characters in the response to the challenge.
<i>ResponseData</i>	The string of characters representing the actual response.

Description In challenge-response, the license server associates a secret with a feature, provided by the license code. The application also contains this secret. In the license server validation process, an application will “challenge” the license

server with a data string. The license server computes a response according to some previously arranged algorithm using the values, *data* and *secret*, which it returns. The client application locally computes the expected response using *data* and *secret*, and verifies that the expected response matches the response returned by the license server.

In order for the authentication mechanism to work correctly and securely, both the license server and the client application must use the same algorithm to compute the response given the values of *data* and *secret*. LSAPI requires the use of the software, “RSA Data Security, Inc. MD4 Message Digest Algorithm” provided by RSA Data Security, Inc. to compute the response.

In practice, to save execution time and space, the client application need not invoke the MD4 Message Digest Algorithm at run time to calculate the response. Challenge-response pairs can instead be maintained in a pre-computed table.

SentinelLM allows for the usage of multiple secrets, with secrets indexed starting at 1. Client applications can challenge the license server to produce a response for a string date using the *secret[i]*, where *i* is the index of the secret (maximum is 7).

The following structures are used by the challenge parameter in challenge-response. *challenge* is an in/out parameter for the **LSRequest()** and **VLSrequestExt()** function calls and must be properly allocated and initialized by the calling process. Refer to the sample files, *crexamp.c*, *chalresp.c*, and *md4.c* for additional details on using this mechanism.

The parameter used to pass the challenge structure is also used by the library to return the response structure. The CHALLENGE pointer must therefore be typecast to CHALLENGERESPONSE * to obtain the correct response after the function call.

The response to a challenge made with *any* function call, for example, **LSRequest()** is valid only if that function call returns LS_SUCCESS. If LS_SUCCESS is not returned, the response to the challenge is undefined. For more information on how to associate secrets with a features, see “VLScgAllowSecrets()” on page 162 in Chapter 4 - License Code Generation API, “VLScgSetNumSecrets()” on page 164 in Chapter 4 - License Code

VLReleaseExt()

Generation API, and “VLScgSetSecrets()” on page 163 in Chapter 4 - License Code Generation API.

VLReleaseExt()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLReleaseExt (  
    LS_HANDLE      lshandle,  
    unsigned long   *unused1,  
    unsigned char   *logComment,  
    VLServerInfo    *serverInfo;
```

Argument	Description
<i>lshandle</i>	The handle returned by the corresponding LSRequest() .
<i>unused1</i>	Unused. Use the value, LS_DEFAULT_UNITS.
<i>logComment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.
<i>serverInfo</i>	This information is passed to the license server for use in server hook functions. See “VLSeventAddHook()” on page 263.

Returns

The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>lshandle</i> is ambiguous handle; it is not exclusively active or exclusively queued.
VLS_RETURN_FAILED	Generic message indicating that the license could not be returned.
VLS_ALL_UNITS_RELEASED	All units released.

VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also VLSeventAddHook()

VLScleanup()

See “VLScleanup()” on page 37.

VLSbatchUpdate()

Client	Server	Static Library	DLL
✓		✓	✓

Updates several licenses at once. Currently the client library defaults to automatic license renewal. You do not need to call this API unless you disable the automatic license renewal.

Syntax LS_STATUS_CODE VLSbatchUpdate (

int **numHandles*,

LS_HANDLE **lshandle*,

VLSbatchUpdate()

```

unsigned long      *unused1,
long              *unused2,
unsigned char      *unused3,
LS_CHALLENGE      *unused4,
LS_STATUS_CODE    *status;

```

Argument	Description
<i>numHandles</i>	Specifies the number of handles.
<i>lshandle (in)</i>	Array of <i>numHandles</i> handles, allocated by caller.
<i>unused1</i>	Currently ignored—pass in a NULL.
<i>unused2</i>	Currently ignored—pass in a NULL.
<i>unused3</i>	Use NULL as the value.
<i>unused4</i>	Use NULL as the value.
<i>status (out)</i>	Array of <i>numHandles</i> status codes, allocated by caller.

Description API function interface for updating several licenses. It handles properly the fact that some of the licenses may need to be updated locally, and some remotely. In case the handles need to be updated on different license servers, use the **VLSbatchUpdate()** calls interspersed with **VLSsetContactServer()** calls. This function contacts only one license server for the updates. This function does not call built-in error handlers at all. There is no limit on the number of handles passed.

Returns If everything fails, this function will return a non-LS_SUCCESS code. For failures in individual updates of license codes, this function will return LS_SUCCESS, but the value of the corresponding status element will be set to the error code. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle
VLS_CALLING_ERROR	<i>challenge</i> argument is non-NULL, but cannot be understood.
VLS_CALLING_ERROR	License server used for update is not the same one that was used for acquiring the license.

VLS_NO_LICENSE_GIVEN	Generic error indicating that the license was not updated.
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature.
LS_LICENSETERMINATED	Cannot update license because license already expired.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_LICENSE_EXPIRED	License has expired.
VLS_USER_EXCLUDED	User or machine are excluded from accessing requested feature.
VLS_APP_NODE_LOCKED	Requested feature is node locked but update request was issued from unauthorized machine.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_VENDORMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has a license.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_BUFFER_TOO_SMALL	An error occurred in the use of an internal buffer.

See Also “Challenge-response” on page 45, **LSUpdate()**,

VLsbatchUpdate()

VLSsetRemoteRenewalTime(), **VLSdisableLocalRenewal()**,
VLSenableLocalRenewal(), **VLSisLocalRenewalDisabled()**,
VLSsetTimeoutInterval()

Client Configuration Functions

The Client Configuration Functions allow an application to retrieve or overwrite the default setting. The following table summarizes the functions that enable certain properties of the client library to be configured.

Table 3-8: Client Configuration Functions

Function	Description
VLSsetContactServer()	Defines the license server's host name.
VLSgetContactServer()	Retrieves the license server's host name.
VLSsetServerPort()	Defines the license server's communication port.
VLSgetServerPort()	Obtains the license server's communication port.
VLSinitMachineID()	Sets the fields in <i>machineID</i> to default values.
VLSgetMachineID()	Sets <i>machineID</i> values for the current host.
VLSmachineIDtoLockCode()	Computes the <i>machineID</i> locking code.
VLSgetServerNameFromHandle()	Retrieves the license server's name based on <i>handle_id</i> .
VLSinitServerList()	Initializes a list of default license servers to search for a license.
VLSgetServerList()	Retrieves the default license server list.
VLSinitServerInfo()	Initializes the license serverInfo data structure to default values.
VLSsetHostIdFunc()	Sets the host ID function.
VLSsetBroadcastInterval()	Configures broadcast behavior.
VLSgetBroadcastInterval()	Retrieves broadcast behavior parameters.
VLSsetTimeoutInterval()	Configures timeout behavior.

Table 3-8: Client Configuration Functions (Continued)

Function	Description
VLSgetTimeoutInterval()	Retrieves timeout behavior parameters.
VLSsetHoldTime()	Sets license hold time.
VLSsetSharedId()	Redefines shared ID functions.
VLSsetSharedIdValue()	Registers a customized shared ID value.

Note There are also function calls relating to local vs. remote license renewal. For a detailed description, see “Local vs. Remote Renewal of Keys” on page 73.

VLSsetContactServer()

Client	Server	Static Library	DLL
✓		✓	✓

Specifies the computer the licensed application will contact for the license server.

Syntax LS_STATUS_CODE VLSsetContactServer (char *serverName);

Argument	Description
<i>serverName</i>	The host name of the computer running the license server.

Description Each licensed application must be aware of the location of a SentinelLM license server on the network. By default, on the first communication transaction each application first checks the environment variable, LSFORCEHOST for the name of the license server computer. If that environment variable exists, but the license server computer it specifies is not found, SentinelLM returns an error. If the LSFORCEHOST environment variable does not exist, the application checks the environment variable, LSHOST, for the name of the license server computer. If the variable is not set, it looks for a text file named *LSHOST* or *lshost*, which should contain the name of the license server computer, usually in the current directory. If that is also not available, the client uses a broadcast mechanism on

VLSetContactServer()

the local subnet to determine the existence and location of a SentinelLM license server. If a client makes a SentinelLM function call that involves contacting the license server and the license server is not found, the function call returns the error code, `VLS_NO_SERVER_FOUND`. Once contact has been established, the name of the computer on which the license server is running is cached and all future transactions (with the exception of **VLSDiscover()**) are directed to that license server only. If contact with that license server is lost, the SentinelLM client library returns an error.

After a license is successfully requested (via **LSRequest()** or its variants) SentinelLM will remember the name of the license server host which was contacted to obtain the license. In any further client-server communication involving this handle obtained by the client, SentinelLM will always communicate with the license server from which it obtained the license, regardless of intervening **VLSetContactServer()** calls. The license server name set by **VLSetContactServer()** will be contacted only for operations that do not involve an already valid handle. Therefore, in case the original license server goes down, you must request a fresh license (hence a fresh handle) from the new license server you wish to use, instead of attempting to send license update messages to the new license server, unless redundant license servers are in use. When a redundant license server fails, all clients' are automatically reconnected to one of the other redundant license servers.

VLSetContactServer() resets the cached host name to the value of *serverName*. It overrides `LSFORCEHOST` and the `LSHOST` environment variables and the `LSHOST` file. All future transactions will be directed to that host regardless of the validity of the host name or whether a license server is running at that host.

VLSetContactServer() has an extra role to play in case the application is linked with the UNIX integrated library, *liblssrv.a*.

The roles are summarized in the table below:

Linked With	<i>serverName</i>	Meaning
<i>libls.a</i>	Valid host name	Client should communicate with the license server on <i>serverName</i> .
	NULL	Client should determine <i>serverName</i> using default mechanism.
	NO-NET	Calling error.
<i>libnonet.a</i>	Valid host name	Calling error.
	NULL	Communicate with integrated license server.
	NO-NET	Communicate with integrated license server.
<i>liblssrv.a</i>	Valid host name	Client should communicate with the license server on <i>serverName</i> .
	NULL	Client should determine <i>serverName</i> using default mechanism.
	NO-NET	Communicate with integrated license server.

Note In the above discussion, NO-NET, NO_NET, no_net, and no-net are synonymous.

In general, *serverName* is obtained in the following order:

1. Any name supplied with **VLSsetContactServer()** call.
2. The LSFORCEHOST environment variable.
3. The LSHOST environment variable—Checked only at application start-up.
4. The *lshost* file—Checked only at application startup.

In case of *libls.a* and *liblssrv.a*, if no *serverName* is specified using **VLSsetContactServer()**, the LSHOST environment variable, or the *LSHOST* file, a subnet broadcast is used to find a license server.

The environment variable LSFORCEHOST overrides LSHOST and the broadcast mechanism.

VLSgetContactServer()

In case of *libnonet.a*, SentinelLM communicates with the stand-alone license server with no network communication.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_NO_RESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also *VLSgetContactServer()*

VLSgetContactServer()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the license server name.

Syntax LS_STATUS_CODE VLSgetContactServer (
 char **outBuf*,
 int *outBufSz*);

Argument	Description
<i>outBuf</i> (<i>out</i>)	Contains a single license server name, space allocated by caller.
<i>outBufSz</i>	Size of <i>outBuf</i> .

Description Returns the name of the license server host that will be contacted, in case the client has already set the license server name. Otherwise this function will fail. If

VLSsetServerPort()

the SentinelLM library is running in stand-alone mode, it returns the string, VLS_STANDALONE.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>outBuf</i> is NULL.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
LS_BUFFER_TOO_SMALL	<i>outBuf</i> is not large enough to store license server's name.

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also *VLSsetContactServer()*

VLSsetServerPort()

Client	Server	Static Library	DLL
✓		✓	✓

Sets the port number.

Syntax *int port_number (void);*

Description Defines the license server's communication port.

Returns Does not return anything.

VLSgetServerPort()

VLSgetServerPort()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the port number.

Syntax `int VLSgetServerPort (void);`

Description Obtains the number of the port to which all network messages intended for the license server will be sent. The default configured port number is 5093.

Returns The currently set license server port number is returned.

VLSinitMachineID()

Client	Server	Static Library	DLL
✓		✓	✓

Initializes the fields of the *machineID* data structure to the default values for the current host.

Syntax `LS_STATUS_CODE VLSinitMachineID (
 VLSmachineID LSFAR*machineID;)`

Argument	Description
<i>machineID</i>	User allocated structure where the machine ID will be maintained.

Description Sets the fields in *machineID* to their default values.

The license manager uses the following data structure to define the characteristics of a machine.


```
typedef struct {
    unsigned long  id_prom;
    char          ip_addr[VLS_MAXLEN];
    unsigned long  disk_id;
    char          host_name[VLS_MAXLEN];
    char          ethernet[VLS_MAXLEN];
    unsigned long  nw_ipx;
    unsigned long  nw_serial;
    char          portserv_addr[VLS_MAXLEN];
    unsigned long  custom;
    unsigned long  reserved;
    char          cpu_id;
    unsigned long  unused2;
} VLSmachineID;
```

The structure is called the *machineID*, and the contents of the first nine fields are called the fingerprint for the machine to which the contents apply. In practice, a developer may choose to use some subset of these fields for a given machine. To specify which fields are to be used, a flag word called a *lock_selector* is defined. A lock selector is a number which sets aside one bit for each fingerprinting element type. Each bit designates a locking criterion, and the lock selector represents the fingerprint elements for a given machine. Note that a lock selector does not describe the fingerprint, it only designates which fields in the machine ID are to be used to specify the fingerprint. The masks which define each locking criterion are given below.

```
#define VLS_LOCK_ID_PROM      0x1
#define VLS_LOCK_IP_ADDR     0x2
#define VLS_LOCK_DISK_ID     0x4
#define VLS_LOCK_HOSTNAME    0x8
#define VLS_LOCK_ETHERNET    0x10
#define VLS_LOCK_NW_IPX      0x20
#define VLS_LOCK_NW_SERIAL   0x40
#define VLS_LOCK_PORTABLE_SERV 0x80
#define VLS_LOCK_CUSTOM      0x100
#define VLS_LOCK_PROCESSOR_ID 0x200
```

The mask that defines all locking criteria is:

```
#define VLS_LOCK_ALL      0x1FF
```

VLSgetMachineID()

The machine ID and lock selector are input to the license generator and encrypted to create a locking code which then becomes part of the license that authorizes use of an application. When a license is requested by the application, a fingerprint for the machine is calculated and used to create a locking code. This must compare favorably with its counterpart in the license before execution of the application can be authorized.

Returns The status code, VLSgcg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_MACHINE_FAILURE_CODE	<i>machineID</i> is NULL.
--------------------------	---------------------------

For a complete list of the error codes, “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetMachineID()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSgetMachineID (
 unsigned long *lock_selector_in*,
 VLSmachineID LSFAR **machineID*,
 unsigned long LSFAR **lock_selector_out*;)

Argument	Description
<i>lock_selector_in</i>	User provided mask specifying locking criteria to be read.
<i>machineID</i>	User provided machine ID from which locking criteria will be read.
<i>lock_selector_out</i>	Mask returned specifying which locking criteria were read.

Description Sets the values of the machineID struct for the current host. The input machineID struct should first be initialized by calling **VLSinitMachineID()**. Then, calling this function will attempt to read only those items indicated by the *lock_selector_in*. If *lock_selector_out* is not NULL, **lock_selector_out* will be

set to a bit mask specifying which items were actually read. To try and obtain all possible machineID struct items, set *lock_selector_in* to VLS_LOCK_ALL.

Returns The status code, VLScg_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSmachineIDtoLockCode()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSmachineIDtoLockCode(
 VLSmachineID *LSFAR* **machineID*,
 unsigned long *lock_selector*,
 unsigned long *LSFAR* **lockCode*;

Argument	Description
<i>machineID</i>	Machine ID used to generate lock code.
<i>lock_selector</i>	Bit mask defining the different lock criteria to retrieve
<i>lockCode</i>	Lock code string generated from lock selector

Description This function computes the locking code from the machineID based on the lock selector. Note that every bit in *lock_selector* is significant. For instance, if you have a machineID that has valid information only for the IP address (lock selector is 0x2), then you should pass 0x2 into the *lock_selector* parameter. If you pass in any other *lock_selector* value, a different *lockCode* will result.

Returns The status code, VLScg_SUCCESS, is returned if successful and if *lock_selector* is zero. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetServerNameFromHandle()

VLSgetServerNameFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSgetServerNameFromHandle(
 LS_HANDLE *handle_id*,
 char *LSFAR* **outBuf*,
 int *outBufSz*;

Argument	Description
<i>handle_id</i>	The handle returned by LSRequest() or VLSrequestExt()
<i>outBuf</i>	User allocated buffer to receive license server name
<i>outBufSz</i>	Size of buffer in bytes

Description This function retrieves the name of license server based on *handle_id*. A valid *handle_id* is always obtained as a product of a successful license request. This handle is associated with the license server that was contacted for the license request. **VLSgetServerNameFromHandle()** can be used to retrieve the name of the license server which granted the license.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>outBuf</i> is NULL.
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<i>outBuf</i> is smaller than license server's name that will be returned.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSinitServerList()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSinitServerList (
 char *LSFAR* **serverList*,
 int *optionFlag*;

Argument	Description
<i>serverList</i>	Caller allocated array of license server names, or IP or IPX addresses.
<i>optionFlag</i>	A three-bit flag used to determine how license servers are found

Description This function initializes a list of default license servers to contact whenever a call is made to get a license. *serverList* should be in the same format as the last parameter of the **VLSdiscover()** call, and have the same syntax. See “VLSdiscover()” on page 101 for description of *optionFlag*. This API should be called prior to calling **LSRequest()** or **VLSqueuedRequest()**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
----------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetServerList

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSgetServerList (
 char *LSFAR* **outBuf*,
 int *outBufSz*;

Argument	Description
<i>outBuf</i>	User allocated buffer to receive the license server list
<i>outBufSz</i>	Length of buffer in bytes

Description This function returns the default license server list that was set previously through a call to **VLSinitServerList()**. If the default license server list has not been set, an empty string is returned in *outBuf*.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>outBuf</i> is NULL.
LS_BUFFER_TOO_SMALL	<i>outBuf</i> is smaller than license server's name that will be returned.
VLS_NO_SERVER_FILE	License server does not have a list file. License server has not been set and is unable to determine which license server to use.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSinitServerInfo()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSinitServerInfo (
VLSserverInfo *LSFAR*serverInfo*;

Argument	Description
<i>serverInfo</i>	User allocated buffer that will contain initialized VLSserverInfo() .

Description Initializes the *serverInfo* data structure to its default values.

Note This function must be called before calling **VLSrequestExt()** or **VLSreleaseExt()**.

Returns The status code, LS_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetHostIdFunc()

Client	Server	Static Library	DLL
✓	✓	✓	✓

Sets the host ID function.

Syntax LS_STATUS_CODE VLSsetHostIdFunc (long (**myGetHostIdFunc*) ());

Argument	Description
<i>myGetHostIdFunc</i>	The address of the custom host ID function. In Windows this must be the address returned by <i>MakeProcInst</i> .

VLSsetBroadcastInterval()

Description This function sets the host ID function for the client library to be the function pointed to by *myGetHostIdFunc*. This enables the customization of host ID locking.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
----------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetBroadcastInterval()

Client	Server	Static Library	DLL
✓		✓	✓

Sets the broadcast interval.

Syntax LS_STATUS_CODE VLSsetBroadcastInterval (long *interval*);

Argument	Description
<i>interval</i>	The <i>interval</i> between broadcasts in seconds.

Description If a licensed application performs a broadcast to establish the presence of a license server on the subnet, it makes two broadcast attempts, where the length of the second broadcast attempt is slightly longer than the first.

VLSsetBroadcastInterval() sets the *total* length of both attempts to be *interval* seconds. The default value of *interval* is 9 seconds.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
----------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLGetBroadcastInterval()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the broadcast interval.

Syntax long VLGetBroadcastInterval (void);

Description If a licensed application performs a broadcast to establish the presence of a license server on the subnet, it makes two broadcast attempts, where the length of the second broadcast attempt is slightly longer than the first.

Returns **VLGetBroadcastInterval()** returns the *total* length of broadcast attempts.

VLSetTimeoutInterval()

Client	Server	Static Library	DLL
✓		✓	✓

Sets the timeout interval.

VLSetTimeoutInterval()

Syntax LS_STATUS_CODE VLSetTimeoutInterval (long *interval*);

Argument	Description
<i>interval</i>	The timeout period in seconds.

Description This call sets the time-out *interval* for all direct application/license server communication to *interval* seconds. When a licensed application sends a request to a license server and the license server does not respond, it resends the message a few times. Each time, the length of the timeout *interval* is slightly longer than the previous. **VLSetTimeoutInterval()** sets the *total* length of a set of attempts to be *interval* seconds. The default value of *interval* is 30 seconds. Note that these timeouts are different from the broadcast timeouts.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
----------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLGetTimeoutInterval()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the timeout interval.

Syntax long VLGetTimeoutInterval ();

Description When a licensed application sends a request to a license server and the license server does not respond, it resends the message a few times. Each time, the length of the timeout interval is slightly longer than the previous one.

Returns This call retrieves the time-out interval for all direct application/license server communication.

VLSsetHoldTime()

Client	Server	Static Library	DLL
✓		✓	✓

Sets the hold time for licenses.

Syntax LS_STATUS_CODE VLSsetHoldTime (

char **featureName*,

char **version*,

int *timeInSecs*;

Argument	Description
<i>featureName</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>timeInSecs</i>	Time in seconds. Default: 15 seconds.

Description This function sets the hold time of licenses issued to the feature to *timeInSecs* seconds. This function call will only be effective if the license for the feature specifies that the hold time should be set by the application. This function call must be made before the first VLS_REQUEST, **LSRequest()**, or **VLSqueuedRequest()** call for it to be applicable. Once a license is requested using VLS_REQUEST or **LSRequest()**, the hold time is set for that application, and **VLSsetHoldTime()** will not change it.

VLSsetSharedId()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_CALLING_ERROR	An error occurred in the use of an internal buffer.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetSharedId()

Client	Server	Static Library	DLL
✓		✓	✓

Redefines the functions called to get the relevant sharing parameter of the client. For network use only.

Syntax LS_STATUS_CODE VLSsetSharedId (
 int *sharedId*,
 int (**mySharedIdFunc*) (char *);

Argument	Description
<i>sharedId</i>	Must be one of the following values: <ul style="list-style-type: none">• VLS_CLIENT_HOST_NAME_ID• VLS_USER_NAME_ID• VLS_X_DISPLAY_NAME_ID• VLS_VENDOR_SHARED_ID
<i>mySharedIdFunc</i>	Pointer to a function that will return the <i>sharedID</i> value.

Description This function must be used to register a customized *sharedID* function with the SentinelLM client library. The value of the *sharedID* must be passed back by

mySharedIdFunc through the character buffer. All *sharedID* character buffers will be truncated to 32 bytes. For instance, a customized function which returns the host name can be used by the client library instead of the built-in function to determine eligibility for sharing a license.

Note If the host name or user name are changed using this function, the change will also be reflected in the usage file written by the license server.

One of the many flexibilities provided by LM licensing is the sharing of same license keys, based on the following criteria:

1. User-name based sharing
2. Hostname based sharing
3. X-display based sharing
4. Application-defined sharing

This model is often used by software publishers who do not want to count every instance of a running application. They may allow multiple instances of a running application to share a single license token/key based on a common user name, host name or custom sharing criteria.

When any of the sharing-options are enabled in a license, the license server checks if the new request made by a client is coming from the same User/Host/X-display or not. If it is so, then it checks with the sharing-limit per license-key and then issues the same key to the new user.

Internally, **VLSrequestExt()** function, while sending a License Issue Request Message to the license server, passes on the information regarding its user-name, client-hostname, x-displayname to the license server. This information is kept by the license server in its internal tables for future use. The next time a license is requested for the same Feature, the saved information is used to determine whether this new request is originating from the same user/host/x-display.

By default, SentinelLM has default functions to get these values (i.e. user name, x-display, etc.). To use your own functions to retrieve these values, use the **VLSsetSharedId()** function to override the default functions.

VLSsetSharedIdValue()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>mySharedIdFunc</i> is NULL.
VLS_UNKNOWN_SHARED_ID	Invalid <i>sharedId</i> ; is either negative or exceeds maximum value.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetSharedIdValue()

Client	Server	Static Library	DLL
✓		✓	✓

Uses the value passed in by the caller as the shared ID for licensing purposes. For network use only.

Syntax LS_STATUS_CODE VLSsetSharedIdValue (
 int *sharedId*,
 char **sharedIdValue*;

Argument	Description
<i>sharedId</i>	Must be one of the following values: <ul style="list-style-type: none">• VLS_CLIENT_HOST_NAME_ID• VLS_USER_NAME_ID• VLS_X_DISPLAY_NAME_ID• VLS_VENDOR_SHARED_ID
<i>sharedIdValue</i>	A character buffer which can contain up to 32 characters.

Description This function goes along with **VLSsetSharedId()** and can be used to register a customized *sharedId* value with the SentinelLM client library. You can explicitly provide the *sharedId* itself using this function. The value of the *sharedId* must be passed through the character buffer. All *sharedId* character buffers will be truncated to 32 bytes. If you call both **VLSsetSharedId()** and

VLSsetSharedIdValue(), **VLSsetSharedId()** has priority and the value set by **VLSsetSharedIdValue()** will be ignored.

The same concept applies to **VLSsetSharedIdValue()** function as **VLSsetSharedId()** function. The difference between **VLSsetSharedId()** and **VLSsetSharedIdValue()** lies in the fact that **VLSsetSharedId()** function will make the **VLSrequestExt()** internally send different IDs as returned by the Developer-Defined functions, whereas **VLSsetSharedIdValue()** will make the **VLSrequestExt()** send the same ID irrespective of the fact “who is running the client,” “from where the client is being run,” and so on.

The first priority is given to the developer defined functions as set by **VLSsetSharedId()**. If no developer defined function is found then the priority is passed to the **SharedIdValue()** as set by **VLSsetSharedIdValue()** function. If neither the developer defined function nor the developer defined **SharedIdValue()** is found, the default functions are used.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	An error occurred in the use of an internal buffer.
-------------------	---

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Local vs. Remote Renewal of Keys

The key (license token) issued by the license server to a client upon request has to be renewed by calling **LSUpdate()** within the period of the license lifetime. The APIs related to enabling/disabling of a local renewal basically changes the time during the lifetime of the license, at which an update is sent to the license server. Unless updates are carried out by setting auto-timers, updating the license on the license server has to be carried out manually by the client before the expiration of the license lifetime. For more information on this, see “LSUpdate()” on page 38.

VLSdisableLocalRenewal()

The following function calls relate to license renewal:

Table 3-9: License Renewal Functions

Function	Description
VLSdisableLocalRenewal()	Disables local license renewal.
VLSenableLocalRenewal()	Resets local license renewal.
VLSisLocalRenewalDisabled()	Informs you whether or not local updates are enabled.
VLSgetRenewalStatus()	Returns renewal status.
VLSsetRemoteRenewalTime()	Sets the remote renewal period.
VLSdisableAutoTimer()	Disables automatic renewal of one feature.

VLSdisableLocalRenewal()

Client	Server	Static Library	DLL
✓		✓	✓

Forces all future license renewals to go to the license server.

Syntax LS_STATUS_CODE VLSdisableLocalRenewal (void);

This function has no arguments.

Description This disables the local license renewal mechanism. Under local renewal, calls to **LSUpdate()** do not result in a message being sent to the license server until the remote renewal time is reached. On executing this function call, all future license renewals made using **LSUpdate()** or VLS_UPDATE for all handles in this process, will go to the license server for renewal.

Returns The status code, LS_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **LSUpdate()**, **VLSenableLocalRenewal()**

VLSenableLocalRenewal()

Client	Server	Static Library	DLL
✓		✓	✓

Resets the license renewal mechanism to the default.

Syntax LS_STATUS_CODE VLSenableLocalRenewal (void);

This function has no arguments.

Description License server will only be contacted when a license is close to its expiration date. Resets the license renewal for all future license renewals made using **LSUpdate()** or VLS_UPDATE for all handles in this process.

Returns The status code, LS_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.
Updates until remote renewal time will not go to the license server. Updates will be returned locally. Only updates sent after the remote renewal time will be sent to the license server.

See Also **LSUpdate()**, **VLSdisableLocalRenewal()**

VLSisLocalRenewalDisabled()

Client	Server	Static Library	DLL
✓		✓	✓

Informs you whether or not local updates are enabled.

Syntax VLS_LOC_UPD_STAT VLSisLocalRenewalDisabled (void);

This function has no arguments.

VLSgetRenewalStatus()

Returns Returns the following error codes:

VLS_LOCAL_UPD_ENABLE	Local renewal is enabled. This is the initial status and the status after VLSenableLocalRenewal() is called.
VLS_LOCAL_UPD_DISABLE	Local renewal is disabled. This is the status after VLSdisableLocalRenewal() is called.

VLSgetRenewalStatus()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves license renewal status.

Syntax LS_STATUS_CODE VLSgetRenewalStatus (*void*);

This function has no arguments.

Description Returns the renewal status of the last successful license renewal made using **LSUpdate()** or VLS_UPDATE. If the last operation that successfully renewed a license involved contacting the license server, this function returns VLS_REMOTE_UPDATE. If the last operation that successfully renewed a license did not involve contacting the license server (was done locally), this function returns the value VLS_LOCAL_UPDATE. If an update has not occurred, it returns VLS_NO_UPDATES_SO_FAR.

Returns Returns the following error codes:

VLS_NO_UPDATES_SO_FAR	No updates have been made. Specifies the initial value.
VLS_LOCAL_UPDATE	During the most recent update, the license was valid and did not need to be renewed.
VLS_REMOTE_UPDATE	During the most recent update, the license was invalid and required update from the license server.

See Also **LSUpdate()**

VLSsetRemoteRenewalTime()

Client	Server	Static Library	DLL
✓		✓	✓

Sets the remote renewal time period.

Syntax LS_STATUS_CODE VLSsetRemoteRenewalTime (

char **featureName*,

char **version*,

int *timeInSecs*;

Argument	Description
<i>featureName</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>timeInSecs</i>	Time in seconds. Default: 15 seconds.

Description Sets the remote renewal period of licenses issued to the feature to *timeInSecs* seconds. This function call must be made before the first VLS_REQUEST or **LSRequest()** call for it to be applicable. Once a license is requested using VLS_REQUEST or **LSRequest()**, the remote renewal time is set for that application, and **VLSsetRemoteRenewalTime()** will not change it.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_CALLING ERROR	An error occurred in the use of an internal buffer.

VLSdisableAutoTimer()

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **LSRequest(), LSUpdate()**

VLSdisableAutoTimer()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSdisableAutoTimer (
 LS_HANDLE *lshandle*,
 int *state*;

Argument	Description
<i>lshandle</i>	The handle returned by LSRequest() or VLSrequestExt()
<i>state</i>	VLS_ON or VLS_OFF

Description Using the handle returned from requesting a license, a call to this function can be used to disable automatic renewal of one feature. Calling with an argument of zero handle disables auto renewal of *all* features.

Note On UNIX, call **VLSdisableAutoTimer()** before using **sleep()** or **SIGALRM**, or there could be a potential conflict with the timer signal.

On Win32, call **VLSdisableAutoTimer()** if thread has no message loop since the message loop is used to process the timer. If you disable the automatic timer, you must ensure that the license key is renewed periodically (before it expires) by calling **LSUpdate()**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Invalid <i>state</i> . Needs to be either VLS_ON or VLS_OFF
LS_BADHANDLE	Invalid handle.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Client Query Functions

There are three functions that return information about a client feature:

Table 3-10: Client Query Functions

Function	Description
VLSgetClientInfo()	Returns information about a client currently licensed by the license server.
VLSgetHandleInfo()	Returns information about a client given a handle.
VLSgetLicInUseFromHandle()	Returns the number of licenses used for the feature name used to obtain a given handle.

Query functions provide a snapshot of the current status of the license server and the features it licenses. Typically, users at a site are interested in information about how many concurrent copies (or licenses) a feature is licensed for, which users are currently using a particular feature, how soon a licensing agreement will expire, and so on. These functions can be used within application software, or to build stand-alone query utilities. All functions return the status code LS_SUCCESS on success or an appropriate error code. For a listing of error values, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

If a license server host name is not established, the client query function will attempt to locate a license server. Information about any instance of application

VLSdisableAutoTimer()

authorized by the SentinelLM license server is returned in the following structure:

Syntax

```
typedef struct client_info_struct {
    char                user_name[VLS_MAXLEN];
    unsigned long host_id;
    char                group[VLS_MAXLEN];
    long                start_time;
    long                hold_time;
    long                end_time;
    long                ey_id;
    char                host_name[VLS_MAXLEN];
    char                x_display_name[VLS_MAXLEN];
    char                shared_id_name[VLS_MAXLEN];
    int                 num_units;
    int                 q_wait_time;
    int                 is_holding;
    int                 is_sharing
                                intis_commuted;
} VLScientInfo;
```

Member	Description
<i>MAXLEN</i>	Set to 64 characters.
<i>user_name</i>	The login name of the user using the application. This information can be changed using the VLSsetSharedId() API call.
<i>host_id</i>	The host ID of the computer on which the user is working. This can be changed using the VLSsetHostIdFunc() call.
<i>group</i>	Name of the reserved group to which the user belongs. If the user does not belong to an explicitly named group, <i>DefaultGrp</i> is returned.
<i>start_time</i>	The time at which the current license code was issued by the license server.
<i>hold_time</i>	Specifies the hold time, in seconds, if any.
<i>end_time</i>	The time at which the user's current license will expire if not renewed.
<i>key_id</i>	The internal ID of the license currently issued to the user's application. After starting up, the license server issues licenses with unique IDs until it is restarted.

Member	Description
<i>host_name</i>	Name of the host/computer where the user is running the application. This information can be changed using the VLSsetSharedId() API call.
<i>x_display_name</i>	Name of the X display where the user is displaying the application. This information can be changed using the VLSsetSharedId() API call.
<i>shared_id_name</i>	A special vendor-defined ID that can be used for license sharing decisions. It always has the fixed value, default-sharing-ID, unless it is changed by registering a custom function using the VLSsetSharedId() API call. If you plan to use this ID, you should register your own function from your application, and choose Application-defined sharing while running the code generator.
<i>num_units</i>	Number of units consumed by the client so far.
<i>q_wait_time</i>	Unused.
<i>is_holding</i>	Has the value, VLS_TRUE, if the user's current license is being held after its expiration. Otherwise, the value is VLS_FALSE.
<i>is_sharing</i>	Total number of clients sharing this particular license, including the current client being queried. If sharing is disabled, <i>is_sharing</i> will be 0.
<i>is_commuted</i>	Total number of clients that have "checked out" a license from the network.

VLSgetClientInfo()

Client	Server	Static Library	DLL
✓		✓	✓

Returns information about a client feature.

Syntax LS_STATUS_CODE VLSgetClientInfo (

 char *featureName,

 char *version,

 int index,

*VL*SgetClientInfo()

char *unused1,
VLSclientInfo *clientInfo;

Argument	Description
<i>featureName</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>index</i>	Used to specify a particular client.
<i>unused1</i>	Use NULL as the value.
<i>clientInfo</i> (out)	The structure in which information will be returned. Space allocated by caller.

Description After this call, *clientInfo* contains information about all clients' features. Since it is possible for multiple clients of a particular feature to be active on the network, *index* is used to retrieve information about a specific client. The suggested use of this function is in a loop, where the first call is made with *index* 0 which retrieves information about the first client. Subsequent calls, when made with 1, 2, 3, and so on, will retrieve information about other clients of that feature type. So long as the index is valid, the function returns the success code, LS_SUCCESS. Otherwise, it returns the SentinelLM status code, VLS_NO_MORE_CLIENTS. Memory for *clientInfo* should be allocated before making the call.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL.
VLS_CALLING_ERROR	<i>clientInfo</i> parameter is NULL <i>index</i> is negative. Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_NO_MORE_CLIENTS	Finished retrieving client information for all clients.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.

VLS_MULTIPLE_VENDORID_FOUND	The license server has licenses for the same feature and version from multiple vendors. It is ambiguous which feature is requested.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER-FILE	No license server has been set and unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetHandleInfo()

Client	Server	Static Library	DLL
✓		✓	✓

Returns information about a client feature.

VLSgetLicInUseFromHandle()

Syntax LS_STATUS_CODE VLSgetHandleInfo (
 LS_HANDLE *lshandle*,
 VLSclientInfo **clientInfo*;

Argument	Description
<i>lshandle</i>	The handle returned by LSRequest() , VLSrequestExt() , or VLS_REQUEST.
<i>clientInfo (out)</i>	The structure in which information will be returned. Space allocated by caller.

Description This function also retrieves client information, except that *lshandle* replaces the arguments (*featureName*, *version*, and *index*) used in **VLSgetClientInfo()**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_BAD_HANDLE	Invalid handle. Handle may have already been released and destroyed from previous license operations or too many handles have already been allocated.
----------------	---

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetLicInUseFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Returns the total number of licenses issued by the license server for the feature name and version used to obtain this handle.

VLSgetLicInUseFromHandle()

Syntax LS_STATUS_CODE VLSgetLicInUseFromHandle (

LS_HANDLE	<i>lshandle</i> ,
int	<i>*totalKeysIssued</i> ;

Argument	Description
<i>lshandle</i>	The handle returned by any Request API call.
<i>totalKeysIssued (out)</i>	The number of licenses issued by the license server. Space should be allocated by the caller.

Description Given a valid handle returned by an **LSRequest()** call or its variants, it returns the total number of licenses issued by the license server for the feature name and version used to obtain this handle. Note that the information returned by this function will be correct only immediately after acquiring the handle. The information in the handle is *not* updated subsequently. For more current information, use **VLSgetFeatureInfo()**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle. Handle has already been released and destroyed from previous license versions or too many handles have been allocated.
LS_BUFFER_TOO_SMALL	<i>in_use_p</i> parameter is NULL.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **VLSgetFeatureInfo()**

VLSgetLicInUseFromHandle()

Feature Query Functions

The following table summarizes the feature query functions:

Table 3-11: Feature Query Functions

Function	Description
VLSgetFeatureInfo()	Retrieves feature licensing information from the license server.
VLSgetVersions()	Retrieves licensed version information for a feature.
VLSgetFeatureFromHandle()	Returns the feature name corresponding to the handle.
VLSgetVersionFromHandle()	Returns the version corresponding to the handle.
VLSgetTimeDriftFromHandle()	Returns the difference in seconds between the estimated current time on the license server and the estimated time on the client.
VLSgetFeatureTimeLeftFromHandle()	Returns the difference in seconds between the estimated current time on the license server and the estimated feature expiration time on the license server.
VLSgetKeyTimeLeftFromHandle()	Returns the difference in seconds between the estimated current time on the license server and the estimated license expiration time on the license server.

Information about specific features licensed by the SentinelLM license server is returned in the following structure.

Syntax

```
/* Feature Information structure */

typedef struct feature_info_struct {
    long          structSz;
    char          feature_name[VLS_MAXFEALEN];
    char          version[VLS_MAXFEALEN];
    int           lic_type;
    int           num_licenses;
    int           total_resv;
    int           lic_from_resv;
    int           qlic_from_resv;
    int           intrial_days_count;
    long          birth_day;
    long          death_day;
    int           int           int
}
```

VLSgetLicInUseFromHandle()

```

    lic_from_free_pool;
    qlic_from_free_pool
    int      is_node_locked;
    int      concurrency;
    int      sharing_crit;
    int      locking_crit;
    int      holding_crit;
    int      num_subnets;
    char      site_license_info [VLS_SITEINFOLEN];
    long      hold_time;
    int      meter_value;
    char      vendor_info [VLS_VENINFOLEN + 1];
    char      cl_lock_info[VLS_MAXCLOCKLEN];
    long      key_life_time;
    int      sharing_limit;
    int      soft_num_licenses;
    int      is_standalone;
    int      check_time_tamper;
    int      is_additive;
                                intisRedundant;
                                intmajority_rule;
                                int
    isCommuter;                num_servers;
                                intlog_encrypt_level;
                                intelan_key_flag;
                                longconversion_time;
                                longavg_queue_time;
                                ;                inttot_lic_reqd
                                intisELMEnabled
                                int
    commuter_keys_left;        commuted_keys
} VLSfeatureInfo;
```

Member	Description
structSz	Calling of the structure.
feature_name	Name of the feature whose information is retrieved. Maximum 64 characters.
version	Feature version.
lic_type	Type of license either trial or normal.
trial_days_count	Number of trial days.

VLSgetLicInUseFromHandle()

Member	Description
<i>birth_day</i>	Day of the license start date.
<i>death_day</i>	The time when the feature expires. The constant, VLS_NO_EXPIRATION, is returned if the license does not have any expiration date.
<i>num_licenses</i>	The total number of licenses the license server is authorized to issue.
<i>total_resv</i>	Number of licenses reserved using group reservations.
<i>lic_from_resv</i>	Number of reserved licenses issued to clients.
<i>lic_from_free_pool</i>	Number of unreserved licenses issued to clients.
<i>qlic_from_free_pool</i>	Number of reserved licenses issued to clients.
<i>is_node_locked</i>	Depending on the locking scheme of the feature, this returns one of the following constants: <ul style="list-style-type: none">• VLS_NODE_LOCKED (client locked to license server)• VLS_CLIENT_NODE_LOCKED (client locked)• VLS_FLOATING (license server locked)• VLS_DEMO_MODE (unlocked)
<i>concurrency</i>	Unused.
<i>sharing_crit</i>	Returns the license sharing criteria, which can be one of the following constants: <ul style="list-style-type: none">• VLS_NO_SHARING• VLS_USER_NAME_ID• VLS_CLIENT_HOST_NAME_ID• VLS_X_DISPLAY_NAME_ID• VLS_VENDOR_SHARED_ID

Member	Description
<i>locking_crit</i>	The license server locking criteria, which can be one of the following constants: <ul style="list-style-type: none"> • VLS_LOCK_ID_PROM • VLS_LOCK_IP_ADDR • VLS_LOCK_DISK_ID • VLS_LOCK_HOSTNAME • VLS_LOCK_ETHERNET • VLS_LOCK_NW_IPX • VLS_LOCK_NW_SERIAL • VLS_LOCK_PORTABLE_SERV • VLS_LOCK_CUSTOM • VLS_LOCK_CPU
<i>holding_crit</i>	The license holding criteria, which can be one of the following constants: <ul style="list-style-type: none"> • VLS_HOLD_NONE (no held licenses). • VLS_HOLD_VENDOR (the client specifies the hold time through the function, VLSsetHoldTime()). • VLS_HOLD_CODE (the license code specifies the hold time).
<i>hold_time</i>	The hold time specified for licenses issued for this feature.
<i>num_subnets</i>	The number of subnet specifications provided for the site.
<i>site_license_info</i>	A space-separated list of subnet wildcard specifications.
<i>meter_value</i>	Unused.
<i>vendor_info</i>	The vendor-defined information string.
<i>cl_lock_info</i>	Locking information about clients in a space-separated string of host IDs and/or IP addresses. If licenses-per-node restrictions have been specified, they are also returned in parentheses with each host ID/IP address. For instance, <i>cl_lock_info</i> could be: <i>0x8ef38b91(20#) 0xa4c7188d 0x51f8c94a(10#).</i>
<i>key_life_time</i>	The license lifetime for this feature (in seconds).
<i>sharing_limit</i>	The limit on how many copies of the licensed application can share the same license.

VLSgetFeatureInfo()

Member	Description
<i>soft_num_licenses</i>	The soft limit (for alerts) on the number of concurrent users of this feature.
<i>is_standalone</i>	Returns VLS_TRUE if this is a stand-alone license or VLS_FALSE if this is a network license.
<i>check_time_tamper</i>	Returns VLS_TRUE if this feature is time tamper proof or VLS_FALSE if not time tamper proof.
<i>is_additive</i>	Returns VLS_TRUE if this is an additive license or VLS_FALSE if this is an exclusive license.
<i>isRedundant</i>	Validates if the license is actually redundant.
<i>majority_rule</i>	Checks whether majority rule is on or off.
<i>num_servers</i>	Number of redundant license servers.
<i>isCommuter</i>	Commuter licenses.
<i>log_encrypt_level</i>	Encryption level in the network license for the license server's usage log file.
<i>elan_key_flag</i>	Validates if the Élan license is converted.
<i>conversion_time</i>	Time when the Élan license code is converted into SentinelLM license code.
<i>avg_queue_time</i>	Average time the past or present clients are in the queue. (Not implemented.)
<i>queue_length</i>	Length of the queue.
<i>tot_lic_reqd</i>	Required number of licenses for all queued clients.
<i>isELMEnabled</i>	Query to Elan licenses.
<i>commuted_keys</i>	Number of commuter keys that have been checked out.
<i>commuter_keys_left</i>	Number of computer keys left.

VLSgetFeatureInfo()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves licensing information about a feature using the structure, *feature_info*.

Syntax LS_STATUS_CODE VLSgetFeatureInfo(
 char **name*,
 char **version*,
 int *index*,
 char **unused1*,
 VLSfeatureInfo **featureInfo*;

Argument	Description
<i>name</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>index</i>	Used to specify a particular client.
<i>unused1</i>	Use NULL as the value.
<i>featureInfo (out)</i>	The structure in which information will be returned. Space must be allocated by caller.

Description Returns information on all features. You will need to initialize the *structSz* field of **VLSfeatureInfo()** structure being passed to this API before actually calling this API.

If *name* is not NULL, information about the feature indicated by *name* and *version* is returned.

If information about all licensed features is desired, name should be NULL, and *index* should be used in a loop as described for the function call,

VLSgetClientInfo(). Refer to the source code of the *lsmon.c* utility for additional information.

VLSgetFeatureInfo() returns the status code, VLS_NO_MORE_FEATURES, when it runs out of features to describe. If an error occurs, for example, the feature is unknown to the SentinelLM license server, an appropriate error code is returned. For a complete list of error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

*VL*SgetFeatureInfo()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>featureInfo</i> is NULL <i>index</i> is negative Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	<i>version</i> is NULL when <i>name</i> is non_NULL
VLS_NO_MORE_FEATURES	Finished retrieving feature information for all features on license server.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER-FILE	No license server has been set and unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetVersions()

Client	Server	Static Library	DLL
✓		✓	✓

Returns the list of versions licensed by the license server for a given feature.

Syntax LS_STATUS_CODE VLSgetVersions (

 char **featureName*,

 int *bufferSize*,

 char **versionList*,

 char **unused1*;

Argument	Description
<i>featureName</i>	Name of the feature.
<i>bufferSize</i>	Specifies the size of <i>versionList</i> .
<i>versionList</i> (out)	An array containing the version list. Space should be allocated by the caller.
<i>unused1</i>	Use NULL as the value.

Description This function returns a list of versions separated by spaces in the array, *versionList*. Space for this array must be allocated prior to the call, and the size of the array must be provided using *bufferSize*. This function is useful in situations where you could have licenses for several versions of your software and you wish to implement version-based licensing schemes.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature.
VLS_APP_UNNAMED	<i>featureName</i> is NULL.
VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.

VLSgetFeatureFromHandle()

VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	No license server has been set and unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_BUFFER_TOO_SMALL	An error occurred in the use of an internal buffer.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetFeatureFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Returns the feature name corresponding to *handle*.

Syntax

```
LS_STATUS_CODE VLSgetFeatureFromHandle (
    LS_HANDLE    handle,
    char         *buffer,
    int          bufferSize;
```

Argument	Description
<i>handle</i>	<i>Handle</i> returned by license request API.

VLSgetVersionFromHandle()

Argument	Description
<i>buffer</i> (out)	Buffer to hold the feature name. Space allocated by caller.
<i>bufferSize</i>	Size of the <i>buffer</i> .

Description The feature name is returned in *buffer* which must be allocated by the calling program. The size of *buffer* is passed in the argument *bufferSize*.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<i>buffer</i> parameter is NULL. Size of feature information exceeds <i>bufferSize</i> parameter.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetVersionFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Returns the version corresponding to handle.

Syntax LS_STATUS_CODE VLSgetVersionFromHandle (
 LS_HANDLE *handle*,
 char **buffer*,
 int *bufferSize*;

Argument	Description
<i>handle</i>	Handle returned by LSRequest(), VLSrequestExt(), or VLS_REQUEST.
<i>buffer</i>	Buffer to hold the feature version. Space allocated by caller.

VLsgetTimeDriftFromHandle()

Argument	Description
<i>bufferSize</i>	Size of the <i>buffer</i> .

Description The feature version is returned in *buffer* which must be allocated by the calling program. The size of *buffer* is passed in the argument, *bufferSize*.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<i>buffer</i> parameter is NULL. Size of feature information exceeds <i>bufferSize</i> parameter.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLsgetTimeDriftFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLsgetTimeDriftFromHandle (
LS_HANDLE *lshandle*,
long **secondsServerAheadOfClient* (*out*);

Argument	Description
<i>lshandle</i>	Handle returned by LSRequest() , VLSrequestExt() , or VLS_REQUEST or VLSQueuedRequest() .
<i>secondsServerAheadOfClient</i>	Caller allocates memory for *out* data. Function returns the difference between system clocks.

Description The function is used when the time properties of the client and server may not be in sync. It returns the difference in seconds between the estimated current time

VLSgetFeatureTimeLeftFromHandle()

on the license server and the estimated time on the client. The estimation error is usually the network latency time.

Note The information returned by this function will be correct only immediately after acquiring the handle. The information in the handle is *not* updated subsequently.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<i>secondsServerAheadOfClient</i> parameter is NULL.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetFeatureTimeLeftFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSgetFeatureTimeLeftFromHandle (
LS_HANDLE *lshandle*,
unsigned long **secondsUntilTheFeatureExpires (*out*)*);

Argument	Description
<i>lshandle</i>	Handle returned by LSRequest() , VLSrequestExt() , or VLS_REQUEST.
<i>secondsUntilTheFeatureExpires</i>	Caller allocates memory for <i>*out*</i> data. Function returns the number of seconds until the expiration of the license for this feature.

Description The function is used when the time properties of the client and server may not be in sync. It returns the difference in seconds between the estimated current time

VLGetFeatureTimeLeftFromHandle()

on the license server and the estimated feature expiration time on the license server.

Note The information returned by this function will be correct only immediately after acquiring the handle. The information in the handle is *not* updated subsequently.

VLGetFeatureTimeLeftFromHandle() provides the difference between the License Expiration Time and the Current System Time at the license server end. For example, if the license expiration date is 20th Aug 1998 (12:00PM) and the current time is 16th June 1998 (12:00AM), then this call will return the difference between these two times, in seconds. This is common to all the clients and is based on the license code for the feature.

Note **VLGetFeatureTimeLeftFromHandle()** does not return the number of trial days left in a trial license.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature.
LS_BUFFER_TOO_SMALL	<i>secondsUntilTheFeatureExpires</i> is NULL.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with the license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	The license server has not been set and cannot determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.

LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
----------------	---

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSgetKeyTimeLeftFromHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Syntax LS_STATUS_CODE VLSgetKeyTimeLeftFromHandle (
 LS_HANDLE *lshandle*,
 unsigned long **secondsUntilTheKeyExpires*;

Argument	Description
<i>lshandle</i>	Handle returned by LSRequest() , VLSrequestExt() , or VLS_REQUEST() .
<i>secondsUntilTheKey Expires</i>	Caller allocates memory for *out* data. Function returns the number of seconds for the run-time license to expire.

Description The function is used when the time properties of the client and server may not be in sync. It returns the difference in seconds between the estimated current time on the license server and the estimated license expiration time on the license server.

Note The information returned by this function will be correct only immediately after acquiring the handle. The information in the handle is *not* updated subsequently.

VLSgetKeyTimeLeftFromHandle() returns the difference between the time when the License Key (as issued by the license server to the client) expires (i.e. before this client must do an **LSupdate()**) and the current time. Since the information in the handle is not updated at regular intervals, the value returned by this call is in very close proximity to the key lifetime mentioned in the license. For example, if the key lifetime mentioned in the license is 2 minutes,

*VL*SgetKeyTimeLeftFromHandle()

the value returned by this call will be approximately 120. Naturally, this value varies with each client.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<i>secondsUntilTheKeyExpires</i> parameter is NULL.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Client Utility Functions

The following table lists functions that provide client library capabilities useful to certain specialized applications:

Table 3-12: Client Utility Functions

Function	Description
VLSdiscover()	Retrieves the names of the computers on the local subnet (or beyond) running the SentinelLM license server which are authorized to service requests from an application.
VLSaddFeature()	Adds licensing information to the license server's internal tables.
VLSaddFeatureToFile()	Adds licensing information about a feature to the license server's internal tables.
VLSdeleteFeature()	Removes licensing information from the license server's internal tables.
VLSgetLibInfo()	Retrieves SentinelLM client library information.
VLSshutDown()	Shuts down the license server.
VLSwhere()	Locates and returns information about the server.

VLSdiscover()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the names of the computers on the local subnet (or beyond) running the SentinelLM license server which are authorized to service requests from an application.

Syntax

```
LS_STATUS_CODE VLSdiscover(
    unsigned char *feature_name,
    unsigned char *version,
    unsigned char *reserved1,
    int           server_list_len,
    char          *server_list,
    int           optionFlag,
    char          *query_list;
```

Argument	Description	IN/OUT
<i>feature_name</i>	Name of the feature.	IN
<i>version</i>	Version of the feature.	IN
<i>reserved1</i>	Use any value.	IN
<i>server_list_len</i>	Specifies the size of <i>server_list</i> .	IN
<i>server_list</i>	Space separated list of license server names.	OUT
<i>optionFlag</i>	A three bit flag which guides the behavior of VLSdiscover() in finding the license servers. Details are discussed later.	IN
<i>query_list</i>	A colon separated list of <i>hostNames</i> to be queried during the search for license servers.	IN

Description *feature_name*, must be licensed by the same vendor as the library issuing the **VLSdiscover()** call. If *version* is NULL, it is treated as a wildcard and all license servers that are authorized to service requests for *feature_name* will respond regardless of *version*. If *feature_name* is NULL, *version* will be ignored and all SentinelLM license servers on the local subnet will respond. The space-

VLSdiscover()

separated name list of the responding SentinelLM license servers are returned in *server_list*. The buffer must be allocated prior to the call and its size provided using *server_list_len*.

query_list is a colon-separated list of host names and/or IP-addresses which are queried during the search for license servers.

optionFlag is a three-bit flag which can have any of the following values or a combination of them:

- **VLS_DISC_NO_USERLIST**—Does not check the host list specified by the user. By default, it first checks the LSFORCEHOST environment variable. If LSFORCEHOST doesn't exist, it reads the list specified by the user in the environment variable, LSHOST, and the file, LSHOST/*lshost*. (The content of these lists are joined together and appended to the contents of *query_list*) append them together and then append to the *query_list*. Finally, all the hosts on this combined list are queried during search for license servers.
- **VLS_DISC_RET_ON_FIRST**—If the combined query list is NULL, this function returns as soon as it contacts a license server and returns the name of this license server in *server_list*. Otherwise, it returns when it hears from a license server whose name is listed in the combined query list. In this case, it returns, in *server_list*, that particular license server name along with all other license servers which are not on the list, but responded by that time. If this option is not specified, this function, **VLSdiscover()**, obtains all the names of all the license servers which responded.
- **VLS_DISC_PRIORITIZED_LIST**—Treats the combined query list as a prioritized one, the leftmost being the highest priority host. After execution, *server_list* contains license servers sorted by this priority. If this option is not specified, the combined query list is treated as a random one.
- **VLS_DISC_DEFAULT_OPTIONS**—This flag is a combination of the aforementioned flags. It should be used if you are undecided which options you need.

- If you want to specify no flags, use the value VLS_DISC_NO_OPTIONS.

Returns The status code, LS_SUCCESS, is returned if stand-alone library is used. Otherwise, it will return the following error codes:

VLS_NO_RESPONSE_TO_BROADCAST	No license servers have responded.
LS_NO_SUCCESS	Failed to retrieve computer names on local subnet.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Examples To get a list of all the SentinelLM license servers running on the subnet, the call can be made as:

```
char server_list[MAX_BUF];
VLSdiscover(NULL, NULL, NULL, MAX_BUF, server_list, VLS_DISC_NO_OPTIONS,
NULL);
```

To get one license server having *feature* for all versions of application, **dots**:

```
char server_list[MAX_BUF];
VLSdiscover("DOTS", NULL, NULL, MAX_BUF, server_list,
VLS_DISC_RET_ON_FIRST, NULL);
```

where “DOTS” is the feature name for the application, **dots**.

To find out license servers for **dots** version 1.0 running on the local subnet as well as on computers 'troilus.soft.net' and '123.23.234.1', and get the results in prioritized order:

```
char query_list[100];
char server_list[MAX_BUF];
strcpy(query_list, "troilus.soft.net:123.23.234.1");
VLSdiscover("DOTS", "1.0", NULL, MAX_BUF, server_list,
VLS_DISC_PRIORITIZED_LIST, query_list);
```

VLAddFeature()

See Also VLSsetBroadcastInterval()

VLAddFeature()

Client	Server	Static Library	DLL
✓		✓	✓

Adds licensing information about a feature.

Syntax LS_STATUS_CODE VLAddFeature (

 unsigned char **licenseString*,

 unsigned char **unused1*,

 LS_CHALLENGE **unused2*;

Argument	Description
<i>licenseString</i>	String containing licensing information.
<i>unused1</i>	Use NULL as the value.
<i>unused2</i>	Use NULL as the value.

Description Dynamically adds the license code, *licenseString*, to the license server’s internal tables. If licensing information for this feature and version already exists in the license server’s tables, it may be overwritten with the new information.

Notice, feature is not permanently added to the license server, therefore the feature will not be on the license server when the license server is shutdown and restarted.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
LS_NO_SUCCESS	<i>licenseString</i> is NULL

VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also **VLSdeleteFeature()**

VLSaddFeatureToFile()

Client	Server	Static Library	DLL
✓		✓	✓

Adds licensing information about a feature.

VLAddFeatureToFile()

Syntax LS_STATUS_CODE VLAddFeatureToFile (
 unsigned char **licenseString*,
 unsigned char **unused1*,
 unsigned char **unused2*,
 LS_CHALLENGE **unused3*;

Argument	Description
<i>licenseString</i>	String containing licensing information.
<i>unused1</i>	Use NULL as the value.
<i>unused2</i>	Use NULL as the value.
<i>unused3</i>	Use NULL as the value.

Description Dynamically adds licensing information about a feature to the license server's internal tables. If licensing information for this feature already exists in the license server's tables, it may be overwritten with the new information. Notice, feature is permanently added to the license server, therefore the feature will be on the license server when the license server is shutdown and restarted.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
LS_NO_SUCCESS	<i>licenseString</i> is NULL.
VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.

VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also VLSdeleteFeature()

VLSdeleteFeature()

Client	Server	Static Library	DLL
✓		✓	✓

Deletes licensing information about a feature.

Syntax LS_STATUS_CODE VLSdeleteFeature (

unsigned char **featureName*,

unsigned char **version*,

VLDeleteFeature()

```
unsigned char    *unused1,
LS_CHALLENGE    *unused2;
```

Argument	Description
<i>featureName</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>unused1</i>	Unused.
<i>unused2</i>	Unused.

Description Deletes licensing information from the license server's internal tables, for the given *featureName* and *version*. This API does not delete licenses from the license file.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_APP_UNNAMED	<i>featureName</i> is NULL <i>version</i> is NULL.
VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
VLS_DELETE_LIC_FAILED	Generic error indicating the feature has not been deleted.
VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has a license.
VLS_MULTIPLE_VENDORID_FOUND	The license server has licenses for the same feature and version from multiple vendors. It is ambiguous which feature is requested.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.

VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

See Also VLSaddFeature()

VLSgetLibInfo()

Client	Server	Static Library	DLL
✓		✓	✓

Returns information about the SentinelLM client library currently being used in the structure pointed to by *pInfo*.

Syntax LS_STATUS_CODE VLSgetLibInfo(LS_LIBVERSION **pInfo*)

```
typedef struct {
    unsigned long ulInfoCode;
    char szVersion [VERSTRLEN];
    char szProtocol [VERSTRLEN];
    char szPlatform [VERSTRLEN];
    char szUnused1 [VERSTRLEN];
    char szUnused2 [VERSTRLEN];
}
```

VLShutDown()

```
} LS_LIBVERSION;
```

Member	Description
<i>ullInfoCode</i>	Unused.
<i>szVersion</i>	The version of the SentinelLM client library.
<i>szProtocol</i>	The communication protocol being used for application/ license server communication.
<i>szPlatform</i>	Platform of the client application.
<i>szUnused1</i>	Unused.
<i>szUnused2</i>	Unused.

Description Space for *pInfo* must be allocated by the caller.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NORESOURCES	<i>pInfo</i> is NULL.
----------------	-----------------------

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLShutDown()

Client	Server	Static Library	DLL
✓		✓	✓

Shuts down license server at specified *hostname*.

Syntax LS_STATUS_CODE VLShutDown (
char **hostname*;

Argument	Description
<i>hostname</i>	The host name of the computer running the license server.

Description A client can send this message to the license server in order to shut the license server down. Once shut down, there is no automatic way of restarting the license server through any client message. Any applications that may be running at that time could stop running after a while, as the license renewal messages will fail once the license server goes down. The license server does not check for running applications prior to shutting down.

The following permissions tests must succeed in order for this call to be successful:

- The client and license server must be running on the same network domain name.
- User identification of the license server process should match the client, or client must be run by superuser (root) as shown in the following table:

Server ----- Client	Win 95/98	WinNT (Admin)	UNIX (non-root)	UNIX (root)
UNIX (non-root)	Same <i>UserName</i>	—	Same <i>UserName</i> or <i>UserId</i>	—
Win 95/98 (non-Admin)	Same <i>UserName</i> or <i>SameHost</i>	—	Same <i>UserName</i>	—
WinNT (non-Admin)	Same <i>UserName</i>	—	Same <i>UserName</i>	—
Win 95/98 (Admin)	yes	yes	yes	yes
WinNT (Admin)	yes	yes	yes	yes
UNIX (root)	yes	yes	yes	yes

VLSwhere()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>hostName</i> parameter is NULL.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSwhere()

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the names of the computers on the local subnet (beyond running) the SentinelLM license server which are authorized to service requests from an application.

Syntax LS_STATUS_CODE VLSwhere(
 unsigned char **feature_name*,
 unsigned char **version*,
 unsigned char **unused1*,
 int *bufferSize*,

VLSwhere()

char
int *server_names,
 broadcastFlag;

Argument	Description	IN/OUT
<i>feature_name</i>	Name of the feature.	IN
<i>version</i>	Version of the feature.	IN
<i>unused1</i>	Use any value.	IN
<i>bufferSize</i>	Specifies the size of the buffer.	IN
<i>server_names</i>	Space separated list of license server names.	OUT
<i>broadcastFlag</i>	A three bit flag which guides the behavior of VLSwhere() in finding the license servers.	IN

Description Locates and returns information about the license servers.

Returns The status code, LS_SUCCESS, is returned if stand-alone library is used. Otherwise, it will return the following error codes:

VLS_NO_RESPONSE_TO_BROADCAST	Failed to retrieve computers names on local subnet.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Error Handling

The following table summarizes the three error-handling functions:

Table 3-13: Error-handling Functions

Function	Description
VLErrorHandle()	Toggles default error handling on or off.
LSGetMessage()	Prints error messages corresponding to specified error code.

VLSErrorHandle()

Table 3-13: Error-handling Functions

Function	Description
VLSsetErrorHandler()	Registers custom error handlers.
VLSsetUserErrorFile()	Configures the display of error messages.

SentinelLM has built-in responses to most error conditions expected to be encountered in the field. For a list of types of error conditions detected by SentinelLM, their descriptions, and the default built-in actions, see “Appendix C - Error and Result Codes for Client Functions” on page 283. The SentinelLM client library has a built-in error handler for each type of error listed.

An error handler is a simple function that tries to correct whatever situation caused the error condition to occur. In most cases, the conditions are difficult to correct, and the handlers perform some clean-up tasks and display error messages.

If an error occurs while processing a function call and the default error handlers are unable to correct the situation, the API functions return an error code after displaying an appropriate error message. If the built-in error handlers are able to correct the error-causing condition, the function call returns the success code, LS_SUCCESS, as if the error never occurred.

VLSErrorHandle()

Client	Server	Static Library	DLL
✓		✓	✓

Turns default error handling on or off.

Syntax

LS_STATUS_CODE VLSErrorHandle (int *flag*);

Argument	Description
<i>flag</i>	To turn on error handling, use VLS_ON. To turn off error handling, use VLS_OFF. Default: VLS_ON.

Description If the value of *flag* is the constant, VLS_ON, error handling is enabled. If *flag* is VLS_OFF, error handling is disabled. When error handlers are not being used, the client function call returns the status code of the latest error condition. The caller of the function should therefore check the value returned by the function before proceeding.

Returns The status code, LS_SUCCESS, is always returned. For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

LSGetMessage()

Client	Server	Static Library	DLL
✓		✓	✓

Prints error messages corresponding to specified error code.

Syntax LS_STATUS_CODE LSGetMessage (

LS_HANDLE *lshandle*,

LS_STATUS_CODE *value*,

unsigned char **buffer*,

unsigned long *bufferSize*;

Argument	Description
<i>lshandle</i>	Handle returned by LSRequest() , VLSrequestExt() , or VLS_REQUEST.
<i>value</i>	Error code.
<i>buffer (out)</i>	Buffer to store message.
<i>bufferSize</i>	Size of the <i>buffer</i> .

Description Returns in the *buffer* a text description of the error condition indicated by error code value, for the feature associated with *lshandle*. The *buffer* must be allocated by the calling function with its size indicated by *bufferSize*.

VLSsetErrorHandler()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_NO_MSG_TEXT	<i>buffer</i> is NULL <i>bufferSize</i> is zero or negative.
----------------	---

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetErrorHandler()

Client	Server	Static Library	DLL
✓		✓	✓

Enables registration of custom error handlers.

Syntax LS_STATUS_CODE VLSsetErrorHandler (
LS_STATUS_CODE (*myErrorHandler)(LS_STATUS_CODE, char *),
LS_STATUS_CODE LSErrorType;

Argument	Description
<i>myErrorHandler</i>	Pointer to the error-handling function.
<i>LSErrorType</i>	Error code to be handled.

Description In some situations, the default responses may not be suitable. Therefore, SentinelLM allows custom error handling routines to replace the default routines. Customized routines should perform actions that are functionally similar to the defaults.

myErrorHandler must point to the error handling function and adhere to the prototype outlined below. *LSErrorType* must indicate the type of the error to be handled. The SentinelLM default routines continue to handle other errors. The customized function should accept as input the error code of the condition that caused it to be called and the name of the feature. The same error-handling function can be used to handle all error conditions for all features of an

application, using internal conditional statements. The special target error code, VLS_EH_SET_ALL, can be used to set up the provided error handler to handle all errors.

Customized error handlers must adhere to the following prototype:

```
LS_STATUS_CODE myErrorHandler (
    LS_STATUS_CODE    errorCode,
    char              *featureName;
```

Argument	Description
<i>errorCode</i>	The error code to be handled.
<i>featureName</i>	The name of the feature involved in the error.

If customized error handlers are used, a client function call will return the value returned by the error handler if it was the last error handler to be called.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>myErrorHandler</i> parameter is NULL LSErrorType() is an invalid error type.
-------------------	--

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

VLSsetUserErrorFile

Client	Server	Static Library	DLL
✓		✓	✓

Configures the manner in which error messages are displayed.

Syntax

```
typedef enum {
    VLS_STDOUT, VLS_STDERR
} VLS_ERR_FILE;
```

VLssetUserErrorFile

```
LS_STATUS_CODE VLssetUserErrorFile(  
    VLS_ERR_FILE    msgFile,  
    char LSFAR      *filePath;  
)
```

Argument	Description
<i>msgFile</i>	The file to which error messages will be directed.
<i>filePath</i>	Full path name of desired error file

Description This function configures the displaying of error messages to the user through the default error handlers. If you disable the default error handlers, you do not need to use this function.

Note The default handling of error messages is as follows:

Windows Pop up a Message Box.
UNIX Write to *stderr*.

You can alter this behavior by providing either a *FILE** or a file path, while keeping the other parameter NULL. If you provide both parameters, preference will be given to the *FILE**.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Could not open <i>msgFile</i> .
-------------------	---------------------------------

For a complete list of the error codes, see “Appendix C - Error and Result Codes for Client Functions” on page 283.

Tracing SentinelLM Operation

Client	Server	Static Library	DLL
✓		✓	✓

Enables tracing of the internal operation of the SentinelLM client library.

Syntax LS_STATUS_CODE VLSsetTraceLevel (int *traceLevel*);

Argument	Description
<i>traceLevel</i>	The default value of <i>traceLevel</i> is VLS_NO_TRACE. Other valid values are: <ul style="list-style-type: none">• VLS_TRACE_KEYS• VLS_TRACE_FUNCTIONS• VSL_TRACE_ERRORS• VLS_TRACE_ALL

VLSetUserErrorFile

Chapter 4 - License Code Generation API

The License Code Generation Application Programming Interface (API) makes it possible to generate license codes to authorize use of an application program. The functions are prototyped in *lscgen.h* and the implementation is contained in *lscgen32.lib*. Use of these files enables you to write your own utility program to generate license codes. Such programs must be written to run under Win95/98 or Windows NT.

Programs that do license generation must first allocate an integer handle and a data structure of type *codeT*. The handle is used with all other License Generation functions, and must be initialized before any of those functions can be called. The *codeT* data structure is used to pass arguments back and forth between the program and the different library functions.

A typical sequence of operations to generate a license would look like the following:

- Be sure that a handle and a *codeT* data structure have been allocated.
- Call **VLScgInitialize()** to initialize the handle. This will ensure that the number of handles has not exceeded the limit, allocate space for internal data structures, and initialize the error list and error count.
- Call **VLScgReset()** to install default values into the *codeT* data structure. This must be done before setting the values of any of the fields in the data structure.
- Obtain input from the user that is to be used to define the license code. The order of input is important since some values will depend on others. The order of input refers to the *Allow* and *Set* functions of code struct.

We suggest you use the *Allow* function first to check the differential integrity of the field value, before using the *Set* function. Please refer to Table 4-3, “Functions of the Code Struct,” on page 132.

- Call the appropriate **VLScgAllowXXX()** function for each input to ensure that its value can be properly included into the license code.
- If the input can be accepted, call the corresponding **VLScgSetXXX()** function. This will lock the *codeT* data structure, install the value in the designated field, and then unlock the structure.
- If the set function causes an error, call **VLScgPrintError()** function to copy the error structure to a specified file.
- After all inputs have been received, call **VLScgGenerateLicense()** to create the license string.
- Call **VLScgCleanup()** to release the handle.

License Code Generation Functions

Available function calls fall into these categories:

- Basic functions
- Functions which retrieve or print errors
- Functions which set flags and data fields of code struct
- License generation functions
- License meter related functions

Example:

```

/*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
/*      This Module contains Proprietary Information of      */
/* Rainbow Technologies, Inc and should be treated as      */ Confidential      *
/*****/

#include <stdio.h>      /* For scanf(), sprintf() etc.*/

#include "lscgen.h"      /* For the code generator API.*/

/* The fixed feature name of licenses generated by this example
 * program.
 */
#define VLS_CGENXMPL_FEATURE_NAME "CGENXMPL"

/* Mnemonic used for setting code structure for long codes. */
#define VLS_LONG_CODE_TYPE_STR  "1"

/*
 * Utility function to print code generator API errors to
 * stderr.
 * It also calls the code generator library cleanup function on
 * the handle if necessary.
 */
static int VLSPrintErrors (VLScg_HANDLE *iHandle, int retCode)
{
    if (*iHandle != VLScg_INVALID_HANDLE) {
        (void) VLScgPrintError(*iHandle, stderr);
        (void) VLScgCleanup(iHandle);
    }
    return retCode;
} /* VLSPrintErrors() */

/*
 * A simple example to illustrate the use of the code
 * generation API to generate license strings.
 * This is a command line utility that generates license codes
 * for a fixed feature name, "CGENXMPL".

```

```

/* It prompts the user for the expiration date and then calls
/* the code generator API functions to generate an appropriate
/* license for CGENXMPL.
/* To build this example, compile and then link with the
/* appropriate code generator API library.
*/
int main ()
{
    /* Code generator library handle. */
    VLScg_HANDLE iHandle;

    /* Code generator APIs license code structure. */
    codeT      licCode;

    /* Expiration date information: acquired from user. */
    int      expMonthInt, expDayInt, expYearInt;
    /* String versions of above for calling code generator API functions.*/
    char      expMonth[10], expDay[10], expYear[10];

    /* For license string to be returned by code generator API. */
    char      *licStr = (char *) NULL;

    /* For return codes from code generator API functions. */
    int      retCode;

    /* Initialize the code generator library. */
    if ((retCode = VLScgInitialize(&iHandle)) != VLScg_SUCCESS) {
        (void) VLSPrintErrors(&iHandle, retCode);
        fprintf(stderr, "\nERROR: Code generator library initialization failed.\n");
        return retCode;
    } /* if (!VLScgInitialize()) */

    /* Initialize the license code structure. */
    if ((retCode = VLScgReset(iHandle, &licCode)) !=
        VLScg_SUCCESS)
        return VLSPrintErrors(&iHandle, retCode);

    /* Specify that we want to generate a long code. */
    if ((retCode = VLScgSetCodeLength(iHandle, &licCode,
        VLS_LONG_CODE_TYPE_STR))
        != VLScg_SUCCESS)
        return VLSPrintErrors(&iHandle, retCode);

    /* Set the feature name. */

```

```

if (VLScgAllowFeatureName(iHandle, &licCode) == 0)
    return VLSPrintErrors(&iHandle, VLScg_FAIL);

if ((retCode = VLScgSetFeatureName(iHandle, &licCode,
    VLS_CGENXMPL_FEATURE_NAME))
    != VLScg_SUCCESS)
    return VLSPrintErrors(&iHandle, retCode);

/*
 * Prompt for and acquire the expiration date from the user.
 */
printf("License Expiration Month [1-12] : ");
scanf("%d", &expMonthInt);
printf("License Expiration Day [1-31] : ");
scanf("%d", &expDayInt);
printf("License Expiration Year      : ");
scanf("%d", &expYearInt);
/* Convert expiration date information to strings. */
sprintf(expMonth, "%d", expMonthInt);
sprintf(expDay,   "%d", expDayInt);
sprintf(expYear,  "%d", expYearInt);

/* Set the expiration date. */
if (VLScgAllowLicExpiration(iHandle, &licCode) == 0)
    return VLSPrintErrors(&iHandle, VLScg_FAIL);

if (((retCode = VLScgSetLicExpirationMonth(iHandle, &licCode,
    expMonth))
    != VLScg_SUCCESS) ||
    ((retCode = VLScgSetLicExpirationDay(iHandle, &licCode,
    expDay))
    != VLScg_SUCCESS) ||
    ((retCode = VLScgSetLicExpirationYear(iHandle, &licCode,
    expYear))
    != VLScg_SUCCESS))
    return VLSPrintErrors(&iHandle, retCode);

/* Generate the license: memory for license string is allocated
 * by library. */
if ((retCode = VLScgGenerateLicense(iHandle, &licCode,
    &licStr))
    != VLScg_SUCCESS)
    return VLSPrintErrors(&iHandle, retCode);

```

VLScgInitialize()

```
/* Print out the license string. */
(void) fprintf(stdout, "%s\n", licStr);

/* Free the license string, which was allocated by VLScgGenerateLicense() */
free(licStr);

/* Terminate use of code generation library cleanly. */
(void) VLScgCleanup(&iHandle);

return 0;

} /* main() */
```

Basic Functions

The following table summarizes the basic functions for this library:

Table 4-1: Basic Functions

Function	Description
VLScgInitialize()	Initializes the handle.
VLScgCleanup()	Destroys the created handle.
VLScgReset()	Resets the structure with default values.

VLScgInitialize()

Syntax `int VLScgInitialize(
 VLScg_HANDLE *iHandleP)`

Argument	Description
<i>iHandleP</i>	The pointer to the instance handle for this library. Provides access to the internal data structure.

Description Required library initialization call. Every API call requires a valid handle. This function allocates resources required for generating licenses. This function must be called before using any other **VLScgXXX()** function.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_MAX_LIMIT_CROSSED	No more handles left.
VLScg_INVALID_HANDLE	Call VLScgCleanup() to free the resources associated with invalid handle.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

See Also VLScgCleanup()

VLScgCleanup()

Syntax int VLScgCleanup(
VLScg_HANDLE *iHandleP)

Argument	Description
<i>iHandleP</i>	The pointer to the instance handle for this library.

Description This function destroys the handle and its associated resources created by **VLScgInitialize()**.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgReset()

VLScgReset()

Syntax int VLScgReset(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	Name of the structure.

Description This function resets the *codeP* structure by filling in default values. It must be called before calling **VLScgSetXXX()** functions.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

Functions Which Retrieve or Print Errors

When errors are encountered during execution of License Generation functions, they are queued to the handle that controls access to the library in use. These errors may be printed immediately, or allowed to accumulate and flushed at a later time. The following table summarizes the functions used to retrieve or print errors:

Table 4-2: Functions Which Retrieve and Print Errors

Function	Description
VLScgGetNumErrors()	Retrieves number of error messages recorded.
VLScgGetErrorLength()	Retrieves the length of a error message.
VLScgGetErrorMessage()	Retrieves the earliest error from the handle.
VLScgPrintError()	Spills the error struct to a file.

VLScgGetNumErrors()

Syntax int VLScgGetNumErrors(
 VLScg_HANDLE **iHandleP*,
 int *numMsgsP*)

Argument	Description
<i>iHandleP</i>	The pointer to the instance handle for this library.
<i>numMsgsP</i>	The number of messages queued to the handle.

Description This function retrieves the number of messages queued to the handle and returns it in *numMsgsP*.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_NO_RESOURCES	If no resources are available.
VLScg_FAIL	If operation failed.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgGetErrorLength()

Syntax int VLScgGetErrorLength(
 VLScg_HANDLE *iHandle*,
 int *msgNum*,
 int *errLenP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>msgNum</i>	The number of the message whose length is to be queried.
<i>errLenP</i>	The length of the message identified by <i>msgNum</i> .

VLScgGetErrorMessage()

Description This function retrieves the length of message # *msgNum* recorded in the handle. It includes the space required for NULL termination.

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_NO_RESOURCES</code>	If no resources are available.
<code>VLScg_FAIL</code>	If operation failed.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgGetErrorMessage()

Syntax

```
int VLScgGetErrorMessage(  
    VLScg_HANDLE  iHandle,  
    char          *msgBuf,  
    int           bufLen )
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>msgBuf</i>	A user allocated buffer into which the reference message will be copied.
<i>bufLen</i>	The byte length of the message copied into <i>msgBuf</i> .

Description This function retrieves the oldest error queued to the handle, and copies a maximum of *bufLen* bytes to *msgBuf* as a null-terminated string. *msgBuf* is a user allocated buffer and must be *bufLen* bytes in length. Upon successful completion of this function, the message retrieved will have been removed from the queue.

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_NO_RESOURCES</code>	If no resources are available.
---------------------------------	--------------------------------

VLScg_FAIL	If operation failed.
------------	----------------------

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgPrintError()

Syntax int VLScgPrintError(
 VLScg_HANDLE *iHandle*,
 FILE **file*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>file</i>	File pointer.

Description This function writes the accumulated errors to the specified file.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_NO_RESOURCES	If no resources are available.
VLScg_FAIL	If operation failed.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgPrintError()

Functions for Setting the Fields in Code Struct

The following table summarizes the functions used to set flags and data fields of the code struct.

Note The sequence of input is very important for the *VLScgAllow* functions and *VLScgSet* functions. You need to use the *Allow* function first to check the differential integrity and syntax of the field value, before using the *Set* function. The *Set* function will put it in the correct structure and format.

Table 4-3: Functions of the Code Struct

Function	Description
VLScgAllowAdditive() VLScgSetAdditive()	Sets the license to exclusion or additive.
VLScgSetCodeLength()	Sets the license code length.
VLScgSetLicType()	Sets the license type.
VLScgAllowHeldLic() VLScgSetHoldingCrit()	Enables/disables license hold time and determines where that hold time is specified.
VLScgAllowNetworkFlag() VLScgAllowStandAloneFlag() VLScgSetStandAloneFlag()	Sets whether license will be for stand-alone or network computer.
VLScgAllowSharedLic() VLScgSetSharedLicType()	Enables shared licenses and sets sharing criteria.
VLScgAllowTrialLicFeature() VLScgSetTrialDaysCount()	Sets the number of trial days.
VLScgAllowLockMechanism() VLScgSetClientLockMechanism()	Sets client's fingerprint criteria.
VLScgSetServerLock Mechanism1()	Sets license server primary fingerprint criteria. Installs license server's fingerprint criteria in primary lock.
VLScgSetServerLock Mechanism2()	Sets license server secondary fingerprint criteria. Installs license server's fingerprint criteria in secondary lock.

Table 4-3: Functions of the Code Struct (Continued)

Function	Description
VLScgAllowClockTamperFlag() VLScgSetClockTamperFlag()	Controls action on detection of clock being set back on the machine.
VLScgAllowOutLicType() VLScgSetOutLicType()	Sets the license output format.
VLScgAllowLicenseType() VLScgSetLicenseType()	Controls the license type.
VLScgAllowCodegenVersion() VLScgSetCodegenVersion()	Sets the version of license codes to generate. Checks if the current license code setting allows multiple features.
VLScgAllowRedundantFlag() VLScgSetRedundantFlag()	Controls whether the license will be used with redundant license servers.
VLScgAllowMajorityRuleFlag() VLScgSetMajorityRuleFlag()	Controls whether the majority of redundant license servers must be running.
VLScgAllowCommuterLicense() VLScgSetCommuterLicense()	Enables commuter licenses to be checked out.
VLScgAllowLogEncryptLevel() VLScgSetLogEncryptLevel()	Controls the network license encryption level for the license server's usage log file.
VLScgAllowMultiKey() VLScgSetKeyType()	Controls whether a license will be single or multi-feature.
VLScgAllowMultipleServerInfo() VLScgSetNumServers()	Fields for information on various license servers.
VLScgAllowSecrets() VLScgSetSecrets() VLScgSetNumSecrets()	Sets the value of the specified challenge-response secrets. Sets the total number of secrets for the challenge-response.
VLScgAllowVendorInfo() VLScgSetVendorInfo()	Sets vendor-defined information in the license.
VLScgAllowFeatureName() VLScgSetFeatureName()	Sets the name of the feature to be licensed.
VLScgAllowFeatureVersion() VLScgSetFeatureVersion()	Sets the version number to be licensed.

VLScgPrintError()

Table 4-3: Functions of the Code Struct (Continued)

Function	Description
VLScgAllowLockModeQuery() VLScgSetClientServerLockMode()	Sets locking mode for the license server computer. Installs client server lock mode in codeP.
VLScgAllowServerLockInfo() VLScgSetServerLockInfo1() VLScgSetServerLockInfo2()	Sets license server primary locking code. Installs license server lock code in primary lock. Sets license server secondary locking code. Installs server lock code in secondary lock.
VLScgAllowClientLockInfo() VLScgSetClientLockInfo()	Sets the client locking code.
VLScgAllowKeysPerNode() VLScgSetKeysPerNode()	Sets the number of license tokens per node for the specified number of clients.
VLScgAllowSiteLic() VLScgSetSiteLicInfo() VLScgSetNumSubnets()	Sets address of subnets licensed application will be restricted to. Sets the number of subnets the licensed application is restricted to.
VLScgAllowNumFeatures() VLScgSetNumFeatures()	Sets the number of features.
VLScgSetNumClients()	Sets the number of client locking codes to be specified.
VLScgAllowNumKeys() VLScgSetNumKeys()	Sets the number of concurrent licenses allowed.
VLScgAllowSoftLimit() VLScgSetSoftLimit()	Sets soft limit number.
VLScgAllowKeyLifeUnits() VLScgSetKeyLifetimeUnits()	Sets unit of time used to specify time between license renewals.
VLScgAllowKeyHoldUnits() VLScgSetKeyHoldtimeUnits()	Sets units of time to be used to specify license hold time.
VLScgAllowKeyLifetime() VLScgSetKeyLifetime()	Sets time between license renewals.
VLScgAllowKeyHoldtime() VLScgSetKeyHoldtime()	Sets the time a license will be held.

Table 4-3: Functions of the Code Struct (Continued)

Function	Description
VLScgAllowLicBirth() VLScgSetLicBirthMonth()	Sets the month of the license start date.
VLScgSetLicBirthDay() VLScgSetLicBirthYear()	Sets the day of the license start date.
VLScgAllowLicExpiration() VLScgSetLicExpirationMonth()	Sets month license expires.
VLScgSetLicExpirationDay() VLScgSetLicExpirationYear()	Sets day month the license expires.
VLScgAllowShareLimit() VLScgSetShareLimit()	Sets the number of users that can share a license.
VLScgSetNumericType()	Sets the value of numeric type.
VLScgSetLoadSWLicFile	Sets and loads the software license file (<i>lscgen.lic</i>).

VLScgAllowAdditive()

Syntax int VLScgAllowAdditive(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*),

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetAdditive()

VLScgSetAdditive()

Syntax int VLScgSetAdditive(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **flag*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	The value of <i>flag</i> indicates whether the license to be generated is additive/exclusive. The legal values are: <ul style="list-style-type: none">• VLScg_ADDITIVE = "0"• VLScg_EXCLUSIVE = "1"

Description This function determines how this license will interact with a license already installed for this feature and version. If a license is defined as exclusive, it will override an existing license for the same feature and version. If a license is additive, its number of users licensed for the feature and version is added to an existing installed license.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_EXCLUSIVE.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_ADDITIVE.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgSetCodeLength()

Syntax int VLScgSetCodeLength(
 VLScg_HANDLE *iHandle*,

codeT
char *codeP,
 *flag)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	<i>Flag</i> values are used to set the <i>code_type</i> member of <i>codeT</i> struct. Legal values are: <ul style="list-style-type: none"> VLScg_SHORT_CODE_STRING = "0" VLScg_LONG_CODE_STRING = "1" VLScg_SHORT_NUMERIC_CODE = "2"

Description Sets the license code length to short or long.
License codes are 10 characters or longer uppercase alphanumeric or all-numeric strings. The code generator will generate long, short or short, numeric license codes.

- Short codes contain less information than the long code and cannot support certain licensing option. However, they have the advantage of being easier to generate and easier to communicate to end users.
- Long codes contain as many characters as needed.
- Short, numeric codes generate numeric strings only and requires minimal information from the user. This code contains the least information.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INPUT	If either <i>codeP</i> or <i>flag</i> are NULL.
VLScg_INVALID_INT_TYPE	Value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_SHORT_CODE_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_LONG_CODE_STRING.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgSetLicType()

VLScgSetLicType()

Syntax

```
int VLScgSetLicType(  
    VLScg_HANDLE iHandle,  
    codeT *codeP,  
    char *lictype)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>lictype</i>	Set the type of license. <ul style="list-style-type: none">• VLScg_TRIAL_LIC_STRING = "1"• VLScg_NORMAL_LIC_STRING = "0"

Description Sets the type of license to either trial or normal. Trial licenses are relative time-limited licenses that use a trial period of 1 to 120 days. Notice, trial licenses do not start until the first time the application is executed (as opposed to the time that the application is installed).

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_LIC_TYPE	If license type is not valid.
------------------------	-------------------------------

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgAllowHeldLic()

Syntax int VLScgAllowHeldLic(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetHoldingCrit()

Syntax int VLScgSetHoldingCrit(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **flag*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	The <i>flag</i> is used to set the criteria for held licenses. Values are: <ul style="list-style-type: none"> VLScg_HOLD_NONE_STRING = "0" - Held licenses not allowed. VLScg_HOLD_VENDOR_STRING = "1" - Client API specifies hold time. VLScg_HOLD_CODE_STRING = "2" - License code specifies hold time.

VLScgAllowStandAloneFlag()

Description This defines the criteria for determining the hold time for a license, and controls whether or not held licenses are allowed for this feature. Hold time provides a grace period after the license is released during which only the original license requestor will be granted the license. Validates and installs the value of the *flag* in the license code structure.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_HOLD_CODE_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_HOLD_NONE_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowStandAloneFlag()

Syntax

```
int VLScgAllowStandAloneFlag(  
    VLScg_HANDLE    iHandle,  
    codeT           *codeP )
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The VLScgSetXXX() function tests whether the corresponding VLScgSetXXX() should be called. If VLScgAllowXXX() returns 1 then the corresponding VLScgSetXXX() function can be called. Otherwise, it will return 0 as false.

VLScgAllowNetworkFlag()

Syntax	int VLScgAllowNetworkFlag(<i>iHandle</i> , codeT * <i>codeP</i>)	VLScg_HANDLE
---------------	---	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetStandAloneFlag()

Syntax

```
int VLScgSetStandAloneFlag(
    VLScg_HANDLE    iHandle,
    codeT            *codeP,
    char             *flag)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	<p><i>Flag</i> values are used to set the <i>standalone_flag</i> of <i>codeT</i> struct. Legal values are:</p> <ul style="list-style-type: none"> VLScg_NETWORK_STRING = "0" VLScg_STANDALONE_STRING = "1"

Description Sets whether license will be for stand-alone or network computer. Stand-alone and network applications cannot be used interchangeably.

VLScgAllowSharedLic()

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_STANDALONE_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_NETWORK_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowSharedLic()

Syntax int VLScgAllowSharedLic(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetSharedLicType()

Syntax int VLScgSetSharedLicType(
 VLScg_HANDLE *iHandle*,

```
codeT
char
*codeP,
*flag )
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	This <i>flag</i> enables shared licenses and specifies the sharing criteria. Legal values are: <ul style="list-style-type: none"> VLScg_NO_SHARING_STRING = "0" VLScg_USER_SHARING_STRING = "1" VLScg_HOSTNAME_SHARING_STRING = "2" VLScg_XDISPLAY_SHARING_STRING = "3" VLScg_VENDOR_SHARING_STRING = "4" - Vendor defined / customized. Need to customize the client library for this.

Description The concept of shared license is only applicable to network licenses. If sharing is enabled a user can use multiple instances of a protected application without consuming more than one license. Call this function enables sharing and also sets which criteria to use to determine eligibility of the user to share a license already granted to an existing user: user name, x-display ID, host name, or vendor-defined.

Sharing allows multiple copies of your application to run at the same time without using more than one license.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_VENDOR_SHARING_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_NO_SHARING_STRING.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgAllowTrialLicFeature()

VLScgAllowTrialLicFeature()

Syntax int VLScgAllowTrialLicFeature(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetTrialDaysCount()

Syntax int VLScgSetTrialDaysCount(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char**daysStr*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>daysStr</i>	String representing the number of days to use in a trial period.

Description Sets the number of trial days to the count specified by the *daysStr* parameter. The count string defines a window of time during which the application can run after the first time the license is requested.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete

list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLockMechanism()

Syntax int VLScgAllowLockMechanism(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetClientLockMechanism()

Syntax int VLScgSetClientLockMechanism(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **criterion*,
 int *client_num*)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>criterion</i>	Mask defining which fields of <i>machineID</i> are to be used for locking. Value should be in hex format.
<i>client_num</i>	Number identifying the client for whom the lock is to be set.

VLScgSetServerLockMechanism1()

Description Installs a client's fingerprint criteria in the code structure. A fingerprint is computed by selecting operating characteristics of the host system and forming a mask with bits set corresponding to those characteristics. The different fingerprinting elements are defined in the `VLScg_LOCK_` section of *lscgen.h*, and includes criteria such as ID Prom, IP address, disk ID, etc.

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_INVALID_HEX_TYPE</code>	If value is not in hexadecimal format.
<code>VLScg_EXCEEDS_MAX_VALUE</code>	If value is too large.
<code>VLScg_LESS_THAN_MIN_VALUE</code>	If the value is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetServerLockMechanism1()

Syntax

```
int VLScgSetServerLockMechanism1(  
    VLScg_HANDLE  iHandle,  
    codeT          *codeP,  
    char           *criterion,  
    int            server)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>criterion</i>	The lock code to install. Value should be in hex format.
<i>server</i>	Number of license servers.

Description This function sets the criteria for the primary license server. Installs a license server's primary fingerprint criteria in the code structure. A fingerprint is computed by selecting operating characteristics of the host system and forming a mask with bits set corresponding to those characteristics. The different fingerprinting elements are defined in the `VLScg_LOCK_` section of *lscgen.h*,

and includes criteria such as ID Prom, IP address, disk ID, etc. A license server can be locked to either of two groups of fingerprints. The second group will be tried if the first licensed fingerprint group fails to match the license server's fingerprint at the end-user site.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_HEX_TYPE	if criterion is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	if <i>client_num</i> is too large.
VLScg_LESS_THAN_MIN_VALUE	if the <i>client_num</i> is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetServerLockMechanism2()

Syntax

```
int VLScgSetServerLockMechanism2(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,
    char          *criterion
    int           server)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>criterion</i>	The lock code to install (in hex).
<i>server</i>	Number of license servers.

Description This function sets the criteria for the secondary license server. Installs a license server's secondary fingerprint criteria in the code structure. A fingerprint is computed by selecting operating characteristics of the host system and forming a mask with bits set corresponding to those characteristics. The different fingerprinting elements are defined in the VLScg_LOCK_ section of *lscgen.h*, and includes criteria such as ID Prom, IP address, disk ID, etc. A license server

VLScgAllowClockTamperFlag()

can be locked to either of two groups of fingerprints. The second group will be tried if the first licensed fingerprint group fails to match the license server's fingerprint at the end-user site.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_HEX_TYPE	If criterion is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If <i>server</i> is too large.
VLScg_LESS_THAN_MIN_VALUE	If the <i>server</i> is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowClockTamperFlag()

Syntax int VLScgAllowClockTamperFlag(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The VLScgSetXXX() function tests whether the corresponding VLScgSetXXX() should be called. If VLScgAllowXXX() returns 1 then the corresponding VLScgSetXXX() function can be called. Otherwise, it will return 0 as false.

VLScgSetClockTamperFlag()

Syntax int VLScgSetClockTamperFlag(
 VLScg_HANDLE *iHandle*,

codeT *codeP,
char *flag.)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	Valid values are: <ul style="list-style-type: none"> VLScg_NO_CHECK_TAMPER_STRING - Do not check clock tamper = "0" VLScg_CHECK_TAMPER_STRING - Check clock tamper = "1"

Description Controls action on detection of clock being set back on the machine. Clock tamper check will only be done when the license server starts up, but the license server will not exit on detection of tampering. Only those license strings that specify they want the check will be denied if tampering is detected. Other features will continue to be served by the license server. Even if someone sets the clock back after starting the license server, and then dynamically adds a tamper-sensitive license string, the license server will detect it and throw the license string out. When the license server accepts a license string at start-up but detects later that the clock has been set back, it does not grant a license for the feature until the clock is reset to its correct value.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a decimal number.
VLScg_INVALID_RANGE	If value is not in the range allowed.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgAllowOutLicType()

VLScgAllowOutLicType()

Syntax int VLScgAllowOutLicType(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetOutLicType()

Syntax int VLScgSetOutLicType(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **flag*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	Valid values are: <ul style="list-style-type: none">• VLScg_ENCRYPTED_STRING = "0"• VLScg_EXPANDED_READABLE_STRING = "1"• VLScg_CONCISE_READABLE_STRING = "2"

Description Controls the type of license string generated. License output formats can be: encrypted, expanded readable, and concise readable.

The license code contains all of the information that defines the license agreement between you and your customer: how many users can run the application at a time, whether the license will expire after a specific number of days, whether the application can only run on a specific computer, and so on. Encrypted license strings contain this information about the license agreement, but cannot be read by your customers.

Concise readable license codes store information about the provisions of a licensing agreement in readable form, such as plain text with white spaces so that it is easily read (and understood) by the user.

The expanded readable license string, a string is appended to the numeric values to specify what that numeric value stands for, e.g., *60_MINS* implies that 60 specifies the time in minutes. These strings do not appear in the concise format, only a *60* appears in the concise readable license string, as opposed to *60_MINS* in the expandable readable format.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a decimal number.
VLScg_INVALID_RANGE	If value is not in the range allowed.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLicenseType()

Syntax int VLScgAllowLicenseType(
iHandle,
codeT **codeP*,) VLScg_HANDLE

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

VLScgSetLicenseType()

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetLicenseType()

Syntax	int VLScgSetLicenseType(<i>iHandle</i> , codeT * <i>codeP</i> , char * <i>flag</i> .)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	<p><i>Flag</i> is used to set the <i>code_type</i> member of <i>codeT</i> struct. The values are:</p> <ul style="list-style-type: none"> VLScg_NORMAL_LIC_STRING - Non-trial license = "0" VLScg_TRIAL_LIC_STRING - Trial license = "1"

Description Controls the license type for non-trial and trial licenses.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_TRIAL_LIC_STRING.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_NORMAL_LIC_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowCodegenVersion()

Syntax	int VLScgAllowCodegenVersion(<div><div>iHandle,</div><div>codeT</div><div>*codeP,)</div></div>		VLScg_HANDLE
	Argument	Description	
	iHandle	The instance handle for this library.	
	codeP	The pointer to the codeT struct.	
Returns	The VLScgSetXXX() function tests whether the corresponding VLScgSetXXX() should be called. If VLScgAllowXXX() returns 1 then the corresponding VLScgSetXXX() function can be called. Otherwise, it will return 0 as false.		

VLScgSetCodegenVersion()

Syntax	int VLScgSetCodegenVersion(<div><div>iHandle,</div><div>codeT</div><div>*codeP,</div></div>		VLScg_HANDLE	
	Argument	Description		char*
	iHandle	The instance handle for this library.		
	codeP	The pointer to the codeT struct.		
	flag	Sets the possible values for version_num flag.		
Description	Sets the version of license codes to generate. Checks if the current license code setting allow multiple features.			
Returns	The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:			
	VLScg_INVALID_INT_TYPE	If value is not numeric.		

VLScgAllowRedundantFlag()

VLScg_EXCEEDS_MAX_VALUE	If value exceeds MAX_CODEGEN_VERSION.
-------------------------	--

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowRedundantFlag()

Syntax	int VLScgAllowRedundantFlag(<i>iHandle</i> , codeT * <i>codeP</i> ,)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetRedundantFlag()

Syntax	int VLScgSetRedundantFlag(<i>iHandle</i> , codeT * <i>codeP</i> , char * <i>flag</i> ,)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Argument	Description
<i>flag</i>	Valid values are: <ul style="list-style-type: none"> VLScg_NON_REDUNDANT_CODE_STRING - Non-redundant license = "0" VLScg_REDUNDANT_CODE_STRING - Redundant license = "1"

Description Controls whether the license will be used with redundant license servers. Redundancy allows the total number of licenses to remain available to the enterprise even if one or more license servers fail. License balancing allows the developer's end user to set up an initial distribution of license tokens among different sites. The SentinelLM license servers will automatically adjust the distribution of the licenses to match the actual usage pattern of the license tokens across the enterprise.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_REDUNDANT_CODE_STRING.
VLScg_LESS_THAN_MIN_VALUE	If value is less than VLScg_NON_REDUNDANT_CODE_STRING.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgAllowMajorityRuleFlag()

Syntax int VLScgAllowMajorityRuleFlag(
iHandle,
codeT *codeP,) VLScg_HANDLE

Argument	Description
<i>iHandle</i>	The instance handle for this library.

VLScgSetMajorityRuleFlag()

Argument	Description
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetMajorityRuleFlag()

Syntax	int VLScgSetMajorityRuleFlag(<i>iHandle</i> , codeT * <i>codeP</i> , char * <i>flag</i> ,)	VLScg_HANDLE
---------------	---	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	Valid values are: <ul style="list-style-type: none"> VLScg_MAJORITY_RULE_FOLLOWS_STRING - Sets the <i>majority_rule_flag</i> = "1" VLScg_MAJORITY_RULE_NOT_FOLLOWS_STRING - Unset the <i>majority_rule_flag</i> = "0"

Description Controls whether the majority of redundant license servers must be running. If the number of redundant license servers running is less than half of the number of license servers specified in the license file, then all servers will stop servicing all old and new clients. For example, if 7 redundant license servers are specified, at least 4 of them must be running to satisfy the majority rule.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScq_INVALID_INT_TYPE	If value is not numeric.
------------------------	--------------------------

VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MAJORITY_RULE_FOLLOWS_ STRING.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_MAJORITY_RULE_NOT_FOLLOW S_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowCommuterLicense()

Syntax	int VLScgAllowCommuterLicense(<i>iHandle</i> , codeT * <i>codeP</i> ,)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetCommuterLicense()

Syntax	int VLScgSetCommuterLicense(<i>iHandle</i> , codeT * <i>codeP</i> , char * <i>flag</i> .)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

VLScgAllowLogEncryptLevel()

Argument	Description
<i>flag</i>	Valid values are: <ul style="list-style-type: none"> VLScg_NOT_ISSUE_COMMUTER_CODES_STRING = “0” VLScg_ISSUE_COMMUTER_LICENSE_CODE_STRING = “1”

Description Enables commuter licenses.

This function is used to generate keys for traveling clients. Commuter licensing allows end users to “check out” a license from a network served license group and “check it in” when they are done using the license.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_ISSUE_COMMUTER_CODES_ STRING.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_NOT_ISSUE_COMMUTER_ CODES_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLogEncryptLevel()

Syntax	int VLScgAllowLogEncryptLevel(<i>iHandle</i> , codeT * <i>codeP</i> ,)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.

Argument	Description
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetLogEncryptLevel()

Syntax	int VLScgSetLogEncryptLevel(<i>iHandle</i> , <i>codeT</i> * <i>codeP</i> , char * <i>flag</i> ,)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	Allowed value are: <ul style="list-style-type: none"> • "0" • "1" • "2" • "3" • "4"

Description Controls the encryption level to the network licenses for the license server's usage log file.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	if value is not a decimal number.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MAX_ENCRYPTION_LEVEL.

VLScgAllowMultiKey()

VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_NO_ENCRYPTION.
---------------------------	---

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowMultiKey()

Syntax

```
int VLSgAllowMultiKey(
    iHandle,T
    codeT          *codeP,)
VLSg_HANDLE
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeyType()

Syntax	int VLScgSetKeyType(<i>iHandle</i> , codeT * <i>codeP</i> , char * <i>flag</i> .)	VLScg_HANDLE
---------------	--	--------------

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	<i>Flag</i> used to set the <i>code_type</i> member of <i>codeT</i> struct. The values are: <ul style="list-style-type: none"> • VLScg_SINGLE_KEY_STRING = “0” • VLScg_MULTI_KEY_STRING = “1”

Description Controls whether a license will be single or multi-feature license code types.

Single Feature: Predefined short, numeric license codes where the license code is for a single feature. Notice, if you select Predefined-Single Feature, the Feature name must be no more than 2 numeric digits. Most of the attributes are already defined for you and cannot be modified.

Multi Feature: Predefined, short numeric license types where multiple features (value between 2 - 11) can be placed into a single license code. Notice, if you select Predefined-Multi Feature, the Feature name must be no more than 2 numeric digits. Most of the attributes are already defined for you and cannot be modified.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a decimal number.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MULTI_KEY_STRING.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_SINGLE_KEY_STRING.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowMultipleServerInfo()

Syntax int VLScgAllowMultipleServerInfo(
 iHandle,
 codeT *codeP,) VLScg_HANDLE

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The VLScgSetXXX() function tests whether the corresponding VLScgSetXXX() should be called. If VLScgAllowXXX() returns 1 then the

VLScgAllowSecrets()

corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgAllowSecrets()

Syntax int VLScgAllowSecrets(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetSecrets()

Syntax int VLScgSetSecrets(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **valu*,
 int *num*,

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>valu</i>	Any printable ASCII text.
<i>num</i>	Number of secrets: should be from 0 to <i>num_secrets</i> -1.

Description Sets the value of the specified challenge-response secrets.

Both the application and the license contain data known as secrets. When an application wishes to challenge, it generates a random text string, which is passed as the challenge value to the license server. In response to this challenge value, the license server examines the software license to determine the secret and computes the corresponding answer. The result is then passed back to the client application as the response to the challenge.

The purpose of the challenge is to verify that there is a valid license present. Even a tampered license server cannot respond correctly to the challenge.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_CHARACTERS	If string is not valid.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetNumSecrets()

Syntax

```
int VLScgSetNumSecrets(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,
    char          *valu,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>valu</i>	This value sets the number of secrets.

Description Sets the total number of secrets for the challenge-response mechanism. Up to seven secret text strings can be specified, each up to twelve characters long. The secrets are encrypted within the license itself, with only the license

VLScgAllowVendorInfo()

server knowing how to decrypt the secrets. The license server will then compute an authentication response when challenged by a client to confirm its identity.

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_INT_TYPE</code>	If value is not numeric.
<code>VLScg_EXCEEDS_MAX_VALUE</code>	If value exceeds <code>VLScg_MAX_NUM_SECRETS</code> .
<code>VLScg_LESS_THAN_MIN_VALUE</code>	If value is lower than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowVendorInfo()

Syntax `int VLScgAllowVendorInfo(
 VLScg_HANDLE iHandle,
 codeT *codeP,)`

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetVendorInfo()

Syntax `int VLScgSetVendorInfo(
 VLScg_HANDLE iHandle,`

VLScgAllowFeatureName()

codeT *codeP,
char *info,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Any printable ASCII text except #. Maximum of 98 characters.

Description Sets vendor-defined information in the license. Supported only for long license codes.

Any piece of information can be encoded into a license code. The information can be retrieved later through a client library function call. This capability is useful for keeping track of distributors or implementing a variety of licensing schemes.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_CHARS	If string is not valid.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowFeatureName()

Syntax int VLScgAllowFeatureName(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.

VLScgSetFeatureName()

Argument	Description
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetFeatureName()

Syntax int VLScgSetFeatureName(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Any printable ASCII text except #.

Description A feature name can represent a single executable file, multiple executable files, or a portion (a function) of an executable file. A feature name may be a maximum of 11 ASCII characters for short license codes and a maximum of 24 for long license codes and two for short, numeric license codes and multi-feature license codes.

Notice, all applications must have a name by which they will be identified.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_NO_FEATURE_NAME	If the name is NULL.
VLScg_RESERV_STR_ERROR	If the string is a reserved string.
VLScg_INVALID_CHARS	If the string characters are not printable.

VLScg_EXCEEDS_MAX_VALUE	If value exceeds 21.
-------------------------	----------------------

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowFeatureVersion()

Syntax int VLScgAllowFeatureVersion(
VLScg_HANDLE *iHandle*,
codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetFeatureVersion()

Syntax int VLScgSetFeatureVersion(
VLScg_HANDLE *iHandle*,
codeT **codeP*,
char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Any printable ASCII text except #. Maximum of 11 characters.

Description Version number is optional. Not supported for short license codes.

VLScgAllowLockModeQuery()

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_RESERV_STR_ERROR</code>	If the string is a reserved string.
<code>VLScg_INVALID_CHARS</code>	If the string characters are not printable.
<code>VLScg_EXCEEDS_MAX_VALUE</code>	If string exceeds maximum number of characters.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLockModeQuery()

Syntax `int VLScgAllowLockModeQuery(
 VLScg_HANDLE iHandle,
 codeT *codeP,)`

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetClientServerLockMode()

Syntax `int VLScgSetClientServerLockMode(
 VLScg_HANDLE iHandle,`

VLScgSetClientServerLockMode()

codeT
char *codeP,
 *flag.)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	The <i>flag</i> values are: <ul style="list-style-type: none"> VLScg_FLOATING_STRING - License server is locked = "0" VLScg_BOTH_NODE_LOCKED_STRING - Clients and license server are locked = "1" VLScg_DEMO_MODE_STRING - Trial license (no locking) = "2" VLScg_CLIENT_NODE_LOCKED_STRING - Only clients are locked = "3"

Description Sets whether license server is locked, clients and license server are both locked, only clients are locked, or neither license server nor clients are locked. Validates the value of *flag* and installs it in the license code structure.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see "Appendix D - Error and Result Codes for License Generation Functions" on page 291.

VLScgAllowServerLockInfo()

VLScgAllowServerLockInfo()

Syntax int VLScgAllowServerLockInfo(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetServerLockInfo1()

Syntax int VLScgSetServerLockInfo1(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **lockCode*,
 int *num*,),

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>lockCode</i>	The lock code to be checked and set. Lock code should be an 8-character hex string (32-bit numeric locking code), optionally preceded by "0x."
<i>num</i>	Position in <i>server_lock_info1</i> where <i>lockCode</i> is stored starting from <i>codeP</i> to <i>num_server_1</i> .

Description Installs the value of *lockCode* in the code structure field *server_lock_info1[num]* to set the primary locking code.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds the maximum number of license servers.
VLScg_LESS_THAN_MIN_VALUE	If the value is less than minimum number of license servers.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetServerLockInfo2()

Syntax

```
int VLScgSetServerLockInfo2(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,
    char          *lockCode,
    int           num,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>lockCode</i>	The lock code to be checked and set. Lock code should be an 8-character hex string (32-bit numeric locking code), optionally preceded by “0x.”
<i>num</i>	Position in <i>server_lock_info2</i> where <i>lockCode</i> is stored starting from <i>codeP</i> to <i>num_server_1</i> .

Description Installs the value of *lockCode* in the code structure field *server_lock_info2[num]* to set the secondary locking code.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
------------------------	--

VLScgAllowClientLockInfo()

VLScg_EXCEEDS_MAX_VALUE	If value is too large.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowClientLockInfo()

Syntax int VLScgAllowClientLockInfo(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetClientLockInfo()

Syntax int VLScgSetClientLockInfo(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **lockCode*,
 int *num*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>lockCode</i>	This buffer is used to set the lock code information for clients.

Argument	Description
<i>num</i>	Number of clients: should be from 0 to maximum number of clients specified -1.

Description Sets the client locking code.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If number is greater than <i>num_nl_clients</i> -1. Number of node locked clients.
VLScg_LESS_THAN_MIN_VALUE	If number is less than 0.
VLScg_INVALID_IP_TYPE	If value is not in dot format.
VLScg_UNKNOWN_LOCK	If the locking criteria is unknown.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowKeysPerNode()

Syntax

```
int VLScgAllowKeysPerNode(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeysPerNode()

VLScgSetKeysPerNode()

Syntax int VLScgSetKeysPerNode(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **keys*,
 int *num*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>keys</i>	Used to set the number of <i>keys</i> per node. Give any decimal value. Should be from 0. Give <i>NOLIMITSTR</i> for no limit.
<i>num</i>	Number of clients: should be from 0 to the maximum number of clients -1.

Description This function sets the number of *keys* per node for the specified number of clients.

For each client locked and client&server locked node, the number of copies running on each computer is controlled. This is an extra per-host restriction in addition to the overall number of licenses.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If number is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If number exceeds <i>num_nl_clients</i> -1.
VLScg_LESS_THAN_MIN_VALUE	If number is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowSiteLic()

Syntax int VLScgAllowSiteLic(
 VLSdg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetSiteLicInfo()

Syntax int VLScgSetSiteLicInfo(
 VLSdg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,
 int *num*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Set the subnet address. You can use wildcards (e.g., *.123.*.28) to specify a range.
<i>num</i>	Subnet number, from 0 to <i>codeP</i> to <i>num_subnet_1</i> .

Description Sets subnet address. See **VLScgSetNumSubnets()**. Specifies the number of subnets used for site licensing.

VLScgSetNumSubnets()

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_RANGE	If value is not in the range allowed and if value is not a valid character.
---------------------	---

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetNumSubnets()

Syntax int VLScgSetNumSubnets(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the number of subnets: should be from 1 to VLScg_MAX_NUM_SUBNETS 0 is a special value which means no site licensing.

Description Sets the number of subnets the licensed application can run on. To set actual site addresses, use **VLScgSetSiteLicInfo*()**.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If input is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If <i>num</i> is greater than <i>codeP</i> to <i>num_subnets</i> .
VLScg_LESS_THAN_MIN_VALUE	If <i>num</i> is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowNumFeatures()

Syntax int VLScgAllowNumFeatures(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*),

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetNumFeatures()

Syntax int VLScgSetNumFeatures(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **flag*),

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>flag</i>	Sets the <i>flag</i> for number of features in case of multi-feature.

Description Sets the number of features.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If input is not a decimal number.
------------------------	-----------------------------------

VLScgSetNumClients()

VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MAX_NUM_FEATURES.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_MIN_NUM_FEATURES.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetNumClients()

Syntax int VLScgSetNumClients(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*.)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Number of client locking codes to be specified.

Description Applications can be locked to specific client computers using locking codes that uniquely identify those computers.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If input is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum number of clients.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowNumKeys()

Syntax

```
int VLScgAllowNumKeys(  
    VLScg_HANDLE    iHandle,  
    codeT           *codeP,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetNumKeys()

Syntax

```
int VLScgSetNumKeys(
    VLScg_HANDLE    iHandle,
    codeT           *codeP,
    char            *info,
    intnum.)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the number of concurrent licenses: should be from 0 to <i>NOLIMITSTR</i> for no limit.
<i>num</i>	Should be 0 in case of single feature and from 0 to " <i>no_of_features</i> -1" in case of multi-feature.

Description Sets the number of concurrent licenses allowed. (Network license only.)

VLScgAllowSoftLimit()

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, it will return the following error codes:

<code>VLScg_INVALID_INT_TYPE</code>	If value is not a non-negative integer.
<code>VLScg_EXCEEDS_MAX_VALUE</code>	If value exceeds maximum number of keys allowed. Maximum value for long codes is 32767 and maximum value for short codes is 255.
<code>VLScg_LESS_THAN_MIN_VALUE</code>	If value is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowSoftLimit()

Syntax `int VLScgAllowSoftLimit(
 VLScg_HANDLE iHandle,
 codeT *codeP,)`

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetSoftLimit()

Syntax `int VLScgSetSoftLimit(
 VLScg_HANDLE iHandle,`

VLScgAllowKeyLifeUnits()

codeT
char *codeP,
 *info,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets soft limit: should be from 0 to <i>NOLIMITSTR</i> for no limit. <i>NOLIMITSTR</i> is not allowed if the license is a commuter license.

Description The soft limit number defines a threshold at which a warning can be issued that the maximum number of licenses is being approached. Must be less than the maximum number of users (the hard limit).

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If information is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If information exceeds maximum number of keys allowed. Maximum value for long codes is 32767 and maximum value for short codes is 255.
VLScg_LESS_THAN_MIN_VALUE	If information is less than 0 nor <i>num</i> is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowKeyLifeUnits()

Syntax int VLScgAllowKeyLifeUnits(
 VLScg_HANDLE *iHandle*,
 codeT *codeP,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.

VLScgSetKeyLifetimeUnits()

Argument	Description
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeyLifetimeUnits()

Syntax int VLScgSetKeyLifetimeUnits(
 VLSeg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Lifetime specification units of keys: from 0 to 3. The values are: <ul style="list-style-type: none">• "0" - Multiple of 1 minute(s), maximum 15 minutes.• "1" - Multiple of 10 minute(s), maximum 150 minutes.• "2" - Multiple of 30 minute(s), maximum 450 minutes.• "3" - Multiple of 60 minute(s), maximum 900 minutes.

Description This function specifies the units of time used to specify the time between renewals. A license must be renewed by the application on a regular schedule or the license will be reclaimed. See **VLScgSetKeyLifetime()**.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If information is a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds 3.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowKeyHoldUnits()

Syntax int VLScgAllowKeyHoldUnits(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeyHoldtimeUnits()

Syntax int VLScgSetKeyHoldtimeUnits(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.

VLScgAllowKeyLifetime()

Argument	Description
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Hold time specification units of keys: from 0 to 3. The values are: <ul style="list-style-type: none"> “0” - Multiple of 1 minute(s), maximum 15 minutes “1” - Multiple of 10 minute(s), maximum 150 minutes. “2” - Multiple of 30 minute(s), maximum 450 minutes. “3” - Multiple of 60 minute(s), maximum 900 minutes.

Description Network licenses may be held for a time when released by a specific user. During that time only the original requestor of the license can be granted the license again. This function sets the units of time used to specify the hold time.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds 3.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowKeyLifetime()

Syntax

```
int VLScgAllowKeyLifetime(
    VLScg_HANDLE    iHandle,
    codeT            *codeP,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeyLifetime()

Syntax int VLScgSetKeyLifetime(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Absolute value in minutes of license lifetime. Maximum depends on lifetime units set by VLScgSetKeyLifetimeUnits() .

Description A license must be renewed by the application on a regular schedule or the license will be reclaimed. This function specifies the number of minutes between renewals. Maximum and granularity depends on **VLScgSetKeyLifetimeUnits()**.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_NOT_MULTIPLE	If value is not a correct multiple.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum key lifetime.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowKeyHoldtime()

VLScgAllowKeyHoldtime()

Syntax int VLScgAllowKeyHoldtime(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetKeyHoldtime()

Syntax int VLScgSetKeyHoldtime(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Absolute values in minutes. Maximum depends on units set by VLScgSetKeyHoldtimeUnits() . <i>NOLIMITSTR</i> for infinite hold time.

Description Network licenses may be held for a time when released by a specific user. During that time only that user can reclaim the license. This function specifies the hold time. This function sets the value *codeP->key_holdtime* to the value of *info* and performs small checks to validate user input.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_NOT_MULTIPLE	If value is not a correct multiple.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed hold time.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 0.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLicBirth()

Syntax int VLScgAllowLicBirth(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The VLScgSetXXX() function tests whether the corresponding VLScgSetXXX() should be called. If VLScgAllowXXX() returns 1 then the corresponding VLScgSetXXX() function can be called. Otherwise, it will return 0 as false.

VLScgSetLicBirthMonth()

Syntax int VLScgSetLicBirthMonth(
 VLScg_HANDLE *iHandle*,

VLScgSetLicBirthDay()

codeT *codeP,
char *info,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the month of year to 1-12 or Jan-Dec.

Description Sets the month of the license start date. Not applicable if year is infinite.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_CHARACTERS	If not a valid string.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed month (exceeds 12).
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetLicBirthDay()

Sets the day of the license start date.

Syntax int VLScgSetLicBirthDay(
 VLScg_HANDLE *iHandle*,
 codeT *codeP,
 char *info,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the day of the month (1-31).

Description Sets the day of the license start date. Not applicable if year has been set to infinite.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_DATE	If value is not valid for the month.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed day.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetLicBirthYear()

Syntax

```
int VLScgSetLicBirthYear(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,
    char          *info,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Enter year in 4 digits (e.g., 1999) to avoid year 2000 problem.

Description Sets the year of the license start date. Not applicable if year is infinite.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_YEAR	If year is invalid.
VLScg_INVALID_BIRTH_YEAR	If year is too early.

VLScgAllowLicExpiration()

VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed year.
-------------------------	--

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowLicExpiration()

Syntax int VLScgAllowLicExpiration(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetLicExpirationMonth()

Syntax int VLScgSetLicExpirationMonth(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the month of year: 1-12 or Jan-Dec.

Description Sets month of date license expires. Not applicable if year is infinite.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_CHARACTERS	If not a valid string.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed month (exceeds 12).
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetLicExpirationDay()

Syntax

```
int VLScgSetLicExpirationDay(
    VLScg_HANDLE  iHandle,
    codeT         *codeP,
    char          *info,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Sets the day of the month: 1-31.

Description Sets the day of the month of the date on which the license expires. No need to set if the year is infinite.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_DATE	If value is not valid for the month.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed day.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

VLScgSetLicExpirationYear()

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetLicExpirationYear()

Syntax int VLScgSetLicExpirationYear(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **info*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>info</i>	Enter year in 4 digits (e.g., 1999) to avoid year 2000 problem. <i>NEVERSTRING</i> for infinite.

Description Sets the year of the date that the license expires.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_YEAR	If year is invalid.
VLScg_INVALID_DEATH_YEAR	If year is too early.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed year.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgAllowShareLimit()

Syntax int VLScgAllowShareLimit(
 VLSeg_HANDLE *iHandle*,
 codeT **codeP*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.

Returns The **VLScgSetXXX()** function tests whether the corresponding **VLScgSetXXX()** should be called. If **VLScgAllowXXX()** returns 1 then the corresponding **VLScgSetXXX()** function can be called. Otherwise, it will return 0 as false.

VLScgSetShareLimit()

Syntax int VLScgSetShareLimit(
 VLSeg_HANDLE *iHandle*,
 codeT **codeP*,
 char **decimalNUM*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>decimalNUM</i>	Controls the number of users/clients who can share a single license. Use a decimal numeric value setting to control the number of users that can share a license. <i>NOLIMITSTR</i> for unlimited.

Description If sharing is set, multiple users or a single user using multiple instances of your application, can share a license.

VLScgSetNumericType()

This function restricts the number of clients who can share a license. The *decimalNUM* limit forces the issue of a new license, when the sharing limit has been reached for a particular license.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetNumericType()

```
Syntax      int VLScgSetNumericType(
                VLScg_HANDLE    iHandle,
                codeT            *codeP,                               intrnum,)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>num</i>	Numeric type values are: <ul style="list-style-type: none"> • VLScg_NUMERIC_UNKNOWN = “0” • VLScg_NOT_NUMERIC = “1” • VLScg_MISC_SHORT_NUMERIC = “2” • VLScg_MISC_NUMERIC = “3”

Description Sets the value of numeric type.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_EXCEEDS_MAX_VALUE	Value exceeds the maximum value of 3.
-------------------------	---------------------------------------

VLScg_LESS_THAN_MIN_VALUE	If value is less than 0.
VLScg_INVALID_INT_TYPE	If the value is not a non-negative integer.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgSetLoadSWLicFile()

Syntax int VLScgSetLoadSWLicFile(
 VLScg_HANDLE *iHandle*,
 char **filename*,

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>filename</i>	Complete name and path of sw license file.

Description Sets and loads the software license file (*lscgen.lic*).

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, a specific error codes is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

License Generation Functions

The following table summarizes the license generation functions:

Table 4-4: License Generation Functions

VLScgGenerateLicense()	Generates the license string.
VLScgDecodeLicense()	Decodes the license string.

VLScgGenerateLicense()

VLScgGenerateLicense()

Syntax int VLScgGenerateLicense(
 VLScg_HANDLE *iHandle*,
 codeT **codeP*,
 char **result*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>codeP</i>	The pointer to the <i>codeT</i> struct.
<i>result</i>	Address of pointer pointing to generated license string.

Description This function generates the license string for the given *codeT* struct. It should be called after all the **VLScgSet()** functions are called. Memory allocation and free for *codeT* are the responsibilities of the caller of function.

Memory allocation for the license string is handled by this function. Its address is to be passed by the caller of this function in the second argument.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLScg_INVALID_VENDOR_CODE	If vendor identification is illegal.
VLScg_VENDOR_ENCRYPTION_FAIL	If vendor-customized encryption fails.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLScgDecodeLicense()

Syntax int VLScgDecodeLicense(
 VLScg_HANDLE *iHandle*,
 char **AnyLicenseString*,
 char **lic_string*,

VLScgDecodeLicense()

int
codeT *lic_string_length*,
 **codeP*.)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>AnyLicenseString</i>	User provided license string to be decoded.
<i>lic_string</i>	User allocated buffer to receive decoded license string.
<i>lic_string_length</i>	Length of decoded license string returned.
<i>codeP</i>	Pointing to <i>codeT</i> containing input license string.

Description This function decodes the license string *AnyLicenseString* and puts the corresponding *codeT* struct in the last argument. Pointer to *codeT* struct is to be passed as the last argument. This pointer will contain the *codeT* corresponding to *AnyString*. This function takes care of all memory allocations it uses.

Returns The status code, *VLScg_SUCCESS*, is returned if successful. Otherwise, it will return the following error codes:

<i>VLScg_INVALID_VENDOR_CODE</i>	If vendor identification is illegal.
<i>VLScg_VENDOR_ENCRYPTION_FAIL</i>	If vendor-customized encryption fails.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

License Meter Related Functions

The following table summarizes the license meter related functions:

Table 4-5: License Meter Related Functions

<i>VLScgGetLicenseMeterUnits()</i>	Returns the number of license generation units.
<i>VLScgGetTrialLicenseMeterUnits()</i>	Returns the number of trial license generation units.

VLScgGetLicenseMeterUnits()

VLScgGetLicenseMeterUnits()

Syntax int VLScgGetLicenseMeterUnits(
 VLSdg_HANDLE *iHandle*,
 long **initialUnitsP*,
 long **unitsLeftP*) int *codegen_version*,)

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>initialUnitsP</i>	The number of units that were initially available.
<i>unitsLeftP</i>	The number of units remaining.
<i>codegen_version</i>	Version of the code generator (7 for SentinelLM 7.x).

Description Returns the number of license generation units available in the attached license meter key.

Returns The status code, VLSdg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLSdg_LICMETER_EXCEPTION	Unknown value in accessing the license meter.
VLSdg_LICMETER_ACCESS_ERROR	Error accessing the license meter.
VLSdg_LICMETER_CORRUPT	License meter is corrupted.
VLSdg_LICMETER_VERSION_MISMATCH	License meter has an invalid version.

For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

On platforms that do not support hardware keys, the function returns *V_FAIL*.

VLScgGetTrialLicenseMeterUnits()

Syntax

```
int VLScgGetTrialLicenseMeterUnits(
    VLScg_HANDLE    iHandle,
    int              units,
    intcodegen_version)
```

Argument	Description
<i>iHandle</i>	The instance handle for this library.
<i>units</i>	The number of licenses available.
<i>codegen_version</i>	Version of the code generator (7 for SentinelLM 7.x).

Description Returns the number of trial license generation units available in the attached license meter.

Returns The status code, `VLScg_SUCCESS`, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

Trial License Related Functions

The following table summarizes the trial license related functions:

Table 4-6: Trial License Related Functions

VLSgetTrialPeriodLeft()	Returns the remaining time left in a trial license.
---------------------------------	---

VLGetTrialPeriodLeft()

```
Syntax      int VLSgetTrialPeriodLeft(
                unsigned char    *feature_name,
                unsigned char    *version
                unsigned long
```

VLScgGetTrialPeriodLeft()

**trialperiod* unsigned char
*LSFAR *unused1,)*

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	Version of the feature. Must be unique.
<i>trialperiod</i>	Number of seconds left in the trial license. Points to integer in the <i>trialperiod</i> parameter.
<i>unused1</i>	Uses NULL as the value.

Description Returns the remaining time left in a trial license. The usage period for trial licenses does not begin until the application is first executed, i.e., not when the application is installed.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

Chapter 5 - Redundancy API

Redundancy allows the total number of licenses to remain available to the enterprise even if one or more license servers fail. For example, if an end user has a 100-user license (100 tokens), the administrator can disperse the license load to three license servers in different segments (these could be across the world). License Server One will have 30, License Server Two will have 30, and License Server Three will have 40. If any license server fails, the license tokens it is serving will be taken over by the remaining license servers. With this type of architecture, a single network segment will not have to handle the load of the entire network traffic.

For information on setting up and using redundant license servers, please see the *SentinelLM Developer's Guide*.

The following table summarizes the redundancy functions:

Table 5-1: Redundancy Functions

Function	Description
VLSaddFeature()	Dynamically adds licensing information about a feature into the license server's internal tables. If licensing information for this feature and version already exists in the license server's tables, it may be overwritten with the new information. Feature is not permanently added to the license server when the license server is shutdown and restarted.
VLSaddFeatureExt()	Adds a license dynamically.
VLSaddFeatureToFile()	Dynamically adds licensing information to the license server's internal tables and normal or redundant license file.
VLSaddFeatureToFileExt()	Writes a license dynamically.

Table 5-1: Redundancy Functions

Function	Description
VLSaddServerToPool()	Sends a request to add a new license server into the pool. This API will actually modify the license structure in order to add the given license server to the pool.
VLSchangeDistbCrit()	Changes license token distribution criteria on license servers in the redundant license server pool.
VLSdelServerFromPool()	Requests to remove a license server's name from the pool. This API will actually modify the license redundant file in order to delete the given license server from the pool.
VLSdiscoverExt()	Returns the license server characteristic information, which has the keys for a particular specified feature and version. The client can decide a license server preference on some criteria
VLSgetDistbCrit()	Returns the current token distribution status for the given license feature and version.
VLSgetDistbCritToFile()	Requests the license server to provide current token distribution status for the given license feature and version or for all features or version (wild card characters are acceptable). Writes the distribution to a file.
VLSgetHostName()	Takes the IP address as input and tries to resolve it into the <i>hostName</i> , if possible.
VLSgetHostAddress()	Accepts <i>hostName</i> as input and tries to resolve it into IP or IPX address, if possible.
VLSgetFeatureInfoToFile()	Requests the license server to provide information for the given license feature and version.
VLSgetLeaderServerName()	Returns the current leader license server's name by contacting any license server. The license server to be contacted is selected by VLSgetServerName() call. So a license server's name must be set before a call is made to this function.

Table 5-1: Redundancy Functions

Function	Description
VLSgetLicSharingServerList()	Returns the license server's names, which are sharing tokens for a given feature name and version. The <i>server_name_list</i> will contain license server names (<i>hostNames</i> or IPX addresses).

VLSaddFeature()

Syntax

```
int VLSaddFeature (
    unsigned char    LSFAR *licenseStr,
    unsigned char    LSFAR *unused1,
    LS_CHALLENGE     LSFAR *unused2
```

Argument	Description
<i>licenseStr</i>	The <i>license string</i> that will be added.
<i>unused1</i>	Should be NULL.
<i>unused2</i>	Should be NULL.

Description Dynamically adds licensing information about a feature into the license server and adds the license code to the license server's internal tables. If licensing information for this feature and version already exists in the license server's tables, it may be overwritten with the new information contained in *licenseStr*. Notice, feature is not permanently added to the license server when the license server is shutdown and restarted.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
LS_NO_SUCCESS	<i>licenseString</i> is NULL

VLAddFeatureExt()

VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	License server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLAddFeatureExt()

Syntax

```
int VLAddFeatureExt (
    unsigned char    LSFAR *licenseString,
    unsigned char    LSFAR *DistCritString
                                unsigned char
```

LSFAR *unused1
 LS_CHALLENGE LSFAR *unused2

Argument	Description
<i>licenseString</i>	The license string that will be added.
<i>DistCritString</i>	Distribution criteria string. The string will allocate the license to another license server, if the main license server is locked.
<i>unused1</i>	Should be NULL.
<i>unused2</i>	Should be NULL>

Description Adds a license dynamically to the license server.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
LS_NO_SUCCESS	<i>licenseString</i> is NULL
VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.

VLAddFeatureToFile()

VLS_NO_SERVER_FILE	License server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLAddFeatureToFile()

Syntax

```
int VLAddFeatureToFile (
    unsigned char    LSFAR *licenseString,
    LSFAR *unused1,
    unsigned char    LSFAR *unused3,
    unsigned char    LSFAR *unused3)
```

Argument	Description
<i>licenseString</i>	The <i>license_string</i> character.
<i>unused1</i>	Should be NULL.
<i>unused2</i>	Should be NULL.
<i>unused3</i>	Should be NULL.

Description Writes a license dynamically to either the redundant license file or normal license file. Notice, feature is permanently added to the license server when the license server is shutdown and restarted.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
LS_NO_SUCCESS	<i>licenseString</i> is NULL.
VLS_ADD_LIC_FAILED	Generic error indicating that the feature has not been added.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_NO_SERVER_FILE	License server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, "Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions" on page 295.

VLAddServerToPool()

VLAddServerToPool()

Syntax int VLAddServerToPool (
 char *LSFAR* *server_name,
 *server_addr, char *LSFAR*

Argument	Description
<i>server_name</i>	Name of the license server to add to the pool.
<i>server_addr</i>	IP or IPX address of the license server.

Description Will send a request to add a new license server into the pool. This API will actually modify the license server redundant license file in order to add the given license server to the pool.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>server_name</i> is NULL <i>server_address</i> is NULL <i>challenge</i> argument is non-NULL, but cannot be understood. Using stand-alone library. This function cannot be used with stand-alone library.
LS_NO_SUCCESS	Generic error indicating that the license server could not be added to the pool.
VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_SERVER_ALREADY_PRESENT	Attempted to add a license server that is already in the pool.
VLS_POOL_FULL	Pool already has maximum number of license servers. No more license servers can be added.

VLS_BAD_HOSTNAME	<i>hostName</i> is not valid.
VLS_NOT_AUTHORIZED	Invalid user.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_CONF_FILE_ERROR	Error in configuration file.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSchangeDistbCrit()

Syntax

```
int VLSchangeDistbCrit (
    char          LSFAR *feature_name,          charLSFAR
    *version                      charLSFAR
    *dist_crit
```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	Version of the feature. Must be unique.
<i>dist_crit</i>	<i>Dist_crit</i> consists of the names of license server, which will have licenses for the given <i>feature_name</i> and <i>version</i> . The <i>dist_crit</i> string must be null-terminated.

Description Requests to change the distribution criteria for the given license feature and version.

VLSDelServerFromPool()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

LS_BAD_DIST_CRIT	Change <i>dist_crit</i> and allocate some keys to the deleted license server.
LS_NON_REDUNDANT_SERVER_CONTACTED	LSHOST is set to a non-redundant license server.
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_NO_AUTHORIZATION	License server does not recognize this feature name.
LS_NO_SUCH_FEATURE	<i>Feature_version</i> is non-existent.
LS_UNRESOLVED_SERVER_NAME	License server's name cannot be resolved.
LS_MSG_TO_LEADER	The request has been sent to the leader license server.

For a complete list of the error codes, "Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions" on page 295.

VLSDelServerFromPool()

Syntax int VLSDelServerFromPool(
char *LSFAR *server_name*, char *LSFAR*
**server_name*,

Argument	Description
<i>server_name</i>	Name of the license server to delete from the pool.
<i>server_addr</i>	IP or IPX address of license server.

Description Will request to remove a license server's name from the pool of redundant license servers. This API will actually modify the redundant license file in order to delete the given license server from the pool.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>server_name</i> is NULL <i>server_address</i> is NULL <i>challenge</i> argument in non-NULL, but cannot be understood. Using stand-alone library. This function cannot be used with stand-alone library.
LS_NO_SUCCESS	Generic error indicating that the license server could not be deleted from the pool
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_SERVER_NOT_PRESENT	Attempted to delete a license server that is not in the pool.
VLS_ONLY_SERVER	Cannot remove the last license server from the pool.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_BAD_HOSTNAME	<i>hostName</i> is not valid.
VLS_NOT_AUTHORIZED	Invalid user,
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_CONF_FILE_ERROR	Error in configuration file.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

VLSDiscoverExt()

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSDiscoverExt()

Syntax	int VLSdiscoverExt(unsigned char <i>LSFAR *feature name,</i> unsigned char <i>LSFAR *version</i> unsigned char <i>LSFAR *unused1</i> * <i>num_servers</i> int * <i>option_Flag</i> * <i>sharing_crit</i> char <i>LSFAR *vendor_list</i>	int int
---------------	--	----------------------------

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	<i>Version</i> of the feature. Must be unique.
<i>unused1</i>	Should be NULL.
<i>num_servers</i>	Number of license servers for which <i>discoverInfo</i> array is allocated.
<i>discoverInfo</i>	The core function that receives the broadcast message, splits and puts the license server's name in array format. VLSdiscoverInfo () struct that will contain requested information.

Argument	Description
<i>option_Flag</i>	<p>The option flag is allowed to be logically ORed with other flags. However, this flag will have first priority.</p> <p>Valid flags are:</p> <ul style="list-style-type: none"> LS_BAD_PARAMETER - License server's name is NULL or an empty string. LS_SERVER_DOES_NOT_EXIST - Named license server does not exist. LS_LEADER_NOT_KNOWN - Leader name is not known. LS_NON_REDUNDANT_SERVER_CONTACTED - Sets LSHOST to non-redundant license server. LS_UNRESOLVED_SERVER_NAME - License server's name is not resolvable. VLS_CALLING_ERROR - License server's name is NULL or an empty string. VLS_SERVER_ALREADY_PRESENT - License server's name already exists in the redundant license server pool. VLS_LEADER_NOT_PRESENT - Leader name is not known. VLS_NON_REDUNDANT_SERVER - Sets LSHOST to non-redundant license server. VLS_ONLY_SERVER - Only one server remained in the pool.
<i>sharing_crit</i>	<p>The license server will match client's internal information with the keys it is already granted. Values are:</p> <ul style="list-style-type: none"> VLScg_NO_SHARING VLScg_USER_SHARING VLScg_HOSTNAME_SHARING VLScg_XDISPLAY_SHARING VLScg_VENDOR_SHARING
<i>vendor_list</i>	<p>Consists of server names. These license servers will be contacted. The names of all the license servers that have licenses for specified <i>feature_name</i> and <i>version</i> will be returned in <i>vendor_list</i> in the same order as in the original (before the call) <i>vendor_list</i>.</p>

VLSDdiscoverExt()

Description Returns the license server characteristic information of the license server which has the license tokens for a specified feature and version. The client can specify a license server preference based on some criteria.

Each license server that is contacted will determine if it has a license that matches the requested feature name and version. If found, the license server will then notify the client with the following information:

- Protocol supported
- Total number of clients connected to the license server
- Server IP address
- Number of units/tokens available
- Whether this client has already been granted a license for the feature and version (based on *sharing_crit*)

Returns The status code, LS_SUCCESS, is returned if stand-alone library is used. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>num_servers</i> is less than or equal to zero.
VLS_NO_RESPONSE_TO_BROADCAST	License servers have not responded.
LS_NO_SUCCESS	Generic error indicating the license server's characteristic information could not be retrieved.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
VLS_DISC_NO_USERLIST	Do not check the host list specified by the user. By default, it first records LSFORCEHOST environment variable. If LSFORCEHOST does not exist, it reads the file <i>LSHOST/lshost</i> .

VLS_DISC_RET_ON_FIRST	If the combined query list is NULL, it returns as soon as it is contacted by the license server and returns the license servers' name in <i>server_list</i> . Otherwise, it returns when it is contacted by the license server the names listed in the combined query list. In this case, it returns, in <i>server_list</i> , that particular default, if this option is not specified. VLSdiscover() returns all the license servers which responded.
VLS_DSC_PRIORITIZER_LIST	Treat the combined query list as a prioritized one, left most being the highest priority host. It returns, in <i>server_list</i> , license servers sorted in the order of priority host. It returns, in <i>server_list</i> , license servers sorted in the order of priority. If this option is not specified, the combined query list is treated as random.
VLS_DISC_REDUNDANT_ONLY	Expecting reply only from redundant license servers. All non-redundant license servers will ignore the message.
VLS_DISC_DEFAULT_OPTIONS	This flag is a combination of the aforementioned flag. Use it if you are not sure which flag you want to specify.

For a complete list of the error codes, "Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions" on page 295.

VLSgetDistbCrit()

Syntax

```
int VLSgetDistbCrit (
    char          *feature_name,
    *feature_version, char          *dist_crit,
    int           distcrit_buflen
```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>feature_version</i>	Version of the feature. Must be unique.

Argument	Description
<i>dist_crit</i>	<i>Dist_crit</i> consists of the names of license server, which have licenses for the given <i>feature_name</i> and <i>version</i> . The <i>dist_crit</i> string must be null-terminated.
<i>distcrit_bufLen</i>	Size of memory allocated for <i>dist_crit</i> .

Description Returns the current token distribution status for the given license feature and version.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>feature_name</i> is NULL <i>version</i> is NULL <i>dist_crit</i> is NULL <i>dist_crit_len</i> is zero or negative <i>challenge</i> argument is non-NULL, but cannot be understood. Using stand-alone library. This function cannot be used with stand-alone library.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
LS_BUFFER_TOO_SMALL	<i>dist_crit</i> buffer not large enough to store information.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_NON_REDUNDANT_FEATURE	Feature is non-redundant and thus cannot be used in this redundancy-related operation.
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.

VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has a license.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetDistbCritToFile()

Syntax

```
int VLSgetDistbCritToFile (
    char          LSFAR *feature_name,
    char          LSFAR *feature_version,
    LSFAR *file_name,
    char
```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>feature_version</i>	Version of the feature.
<i>file_name</i>	License server will write distribution criteria for the specified feature or version to the file.

Description Requests the license server to provide current token distribution status for the given license feature and version, or for all features, or for all versions, or for all features and all versions (wild card characters are acceptable).

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>feature_name</i> is NULL <i>file_name</i> is NULL. Using stand-alone library. This function cannot be used with stand-alone library.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
VLS_FILE_OPEN_ERROR	An error occurred opening the file.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_NON_REDUNDANT_FEATURE	Feature is non-redundant and thus cannot be used in this redundancy-related operation.
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization process.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_BUFFER_TOO_SMALL	Buffer provided is too small.
LS_NO_SUCH_FEATURE	<i>feature_version</i> is non-existent.
LS_NON_REDUNDANT_SERVER_CONTACTED	LSHOST is set to non-redundant license server.

VLS_CALLING_ERROR	License server's name is NULL or an empty string.
VLS_SERVER_ALREADY_PRESENT	License server's name already exists in the redundant license server pool.
VLS_LEADER_NOT_PRESENT	Leader name is not known.
VLS_NON_REDUNDANT_SRVR	Sets LSHOST to non-redundant license server.

For a complete list of the error codes, "Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions" on page 295.

VLSgetFeatureInfoToFile()

Syntax

```
int VLSgetFeatureInfoToFile (
    unsigned char    LSFAR *feature_name,
    unsigned char    LSFAR *version
    LSFAR *file_name                                     char
```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	Version of the feature. Must be unique.
<i>file_name</i>	License server will write distribution criteria for the specified feature or version to the file.

Description Requests the license server to provide all feature information for the given license to *file_name*. Wild cards are acceptable.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>file_name</i> is NULL <i>feature_name</i> is NULL.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.

*VL*SgetHostName()

VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetHostName()

Syntax

```
int VLSgetHostName (
    char          LSFAR *IP_address,
    char          LSFAR *hostname
    HostNameBufLen      int
```

Argument	Description
<i>IP_address</i>	IP addresses to be converted to <i>hostname</i>
<i>hostname</i>	IP address to be converted to <i>hostname</i>
<i>HostNameBufLen</i>	The length of the message copied into <i>hostname</i> .

Description Will take the IP address as input and try to resolve it into the *hostName*, if possible.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>IP_address</i> is NULL <i>hostName</i> is NULL <i>hostNameBufLen</i> is NULL Using stand-alone library. This function cannot be used with stand-alone library.
VLS_INVALID_IP_ADDRESS	<i>IP_address</i> is not valid.
VLS_UNRESOLVED_IP_ADDRESS	<i>IP_address</i> is valid, but could not be resolved.
LS_BUFFER_TOO_SMALL	Length of <i>hostName</i> returned exceeds <i>hostNameBufLen</i> .
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetLeaderServerName()

Syntax int VLSgetLeaderServerName (
 char LSFAR *leader_name,)

Argument	Description
<i>leader_name</i>	Current lead license server's name. Return types: <ul style="list-style-type: none"> • 0 = Success. Found leader license server name. • 1 = Contact license server is not a redundant license server. • 2 = Other error.

Description Returns the current lead license server's name by contacting any license server. The license server to be contacted is selected by **VLSgetServerName()** call. So a license server's name must be set before a call is made to this function.

*VL*SgetLeaderServerName()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>leader_name</i> is NULL <i>leadername_len</i> is NULL.
LS_BUFFER_TOO_SMALL	<i>leadername_len</i> is smaller than the license server name that will be returned.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_LEADER_NOT_PRESENT	Unknown leader.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_NON_REDUNDANT_FEATURE	Feature is non-redundant and thus cannot be used in this redundancy-related operation.
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
LS_UNRESOLVED_IP_ADDRESS	IP address given is not correct.
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_BUFFER_TOO_SMALL	Buffer provided is too small.
VLS_CALLING_ERROR	License server's name is NULL or an empty string.
VLS_SERVER_ALREADY_PRESENT	License server's name already exists in the redundant license server pool.

VLS_LEADER_NOT_PRESENT	Leader name is not known.
VLS_NON_REDUNDANT_SRVR	Sets LSHOST to non-redundant license server.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLGetHostAddress()

Syntax

```
int VLSgetHostAddress (
    char *LSFAR *hostname,
    char *LSFAR *IP_AddressBuf,
    int *IPAddrBufLen
```

Argument	Description
<i>hostname</i>	The host name of the computer containing the license server that is using the log file.
<i>IP_AddressBuf</i>	Pointer to the IP address buffer.
<i>IPAddrBufLen</i>	The length of the message copied into <i>IP_AddressBuff</i> .

Description Will take *hostName* as input and tries to resolve it into IP address, if possible.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>IPaddressBuf</i> is NULL <i>IPAddrBufLen</i> is NULL. Using stand-alone library. This function cannot be used with stand-alone library.
VLS_UNRESOLVED_HOSTNAME	<i>IP_address</i> is valid, but could not be resolved. IPX protocol is current.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetLicSharingServerList()

VLSgetLicSharingServerList()

Syntax int VLSgetLicSharingServerList (

char	LSFAR *feature_name,	
char	LSFAR *feature_version	
SHR_SRVR_TYPE	*server_list	
int	LSFAR *server_list_len	int*num_servers

Argument	Description
feature_name	Name of the feature.
feature_version	Version of the feature.
server_list	A list that contains the license server's names (<i>hostNames</i> or IPX addresses).
server_list_len	License server will retrieve all the license servers names. If the list is larger than the specified limit, it will be truncated.
num_servers	Identifies the number of license servers.

Description Returns the license server names which are sharing tokens for a given feature name and version. The *server_name_list* will contain license server names (*hostNames* or IPX addresses).

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	feature_name is NULL feature_version is NULL server_list is NULL server_list_len is zero.
LS_BUFFER_TOO_SMALL	server_list_len is smaller than license server name that will be returned.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.

VLS_NON_REDUNDANT_FEATURE	Feature is non-redundant and thus cannot be used in this redundancy-related operation.
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable in servicing license operation.
LS_UNRESOLVED_HOSTNAME	Host name given is not correct.
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_BUFFER_TOO_SMALL	Buffer provided is too small.
VLS_CALLING_ERROR	License server's name is NULL or an empty string.
VLS_SERVER_ALREADY_PRESENT	License server's name already exists in the redundant license server pool.
VLS_LEADER_NOT_PRESENT	Leader name is not known.
VLS_NON_REDUNDANT_SRVR	Sets LSHOST to non-redundant license server.

For a complete list of the error codes, "Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions" on page 295.

VLGetLicSharingServerList()

Chapter 6 - License Queuing API

License queuing is the ability of our license servers to take a license request for a feature and place it in reserve until a license is available. Once the license is available, the license server will then notify the requesting application that the license is now ready for use.

License Queuing Example Code

The following sample is for illustration purposes only. For a working sample application, please refer to `qbounce.c` in the samples directory.

```

/*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
/*      This Module contains Proprietary Information of      */
/* Rainbow Technologies, Inc and should be treated as      *//* Confidential
*/
*****/

#include "lserv.h"

Static LS_Handle ls_handle;

/* Prototype of timer handler function      */
void TimerHandler ();

int main(argc, argv)
```

```

int argc;
char **argv;

{
    char feature_name [] = "My Application";
    char version_name [] = "1.0";
    LS_STATUS_CODE returnCode = 0;
    int number_of_units_requested = 1;
    VLSqueuePreference queue_preference;
    int request_flag

if (VLS_INITIALIZE()) { /* Initialize the LS API */
    return (1);
}

request_flag = VLS_REQ_GET | VLS_REQ_QUEUE;

/* Stay in queue at most 30 minutes */
queue_preference.wait_time = 1800;

/* Once license available for this client, reserve it
   for 5 minutes */
queue_preference.hold_time = 600;

queue_preference.priority_num = 1; /* Not used */

/* Don't queue me if there are 5 or more entries
   on the queue */
queue_preference.absPosition = 5;

/* Don't queue me if there are 2 or more entries from my
   reservation group on the queue */
queue_preference.grpPosition = 2;

/* Request key from SentinelLM license manager */
returnCode =
    VLSqueuedRequest
    ( LS_ANY,
      (unsigned char LSFAR *) "SentinelLM User",
      (unsigned char LSFAR *) feature_name,
      (unsigned char LSFAR *) version_name,
      &number_of_units_requested,

```

```

        (unsigned char LSFAR *) NULL,
        (LS_CHALLENGE LSFAR *) NULL,
        &ls_handle,
        &queue_preference,
        &request_flag);

if (returnCode == LS_SUCCESS)
{
    if (request_flag & VLS_REQ_GET)
    {
        /* License was available, run the application! */

    }
    else if (request_flag & VLS_REQ_QUEUE)
    {
        /* Was placed on the queue */

        /* TODO: Start timer for sending periodic queue updates
           (every 50 secs is recommended). Assume function
           TimerHandler () will be called when the timer expires
           (see below). */

    }
}
else
{
    /* Queued request was not successful, clean up and exit. */
    VLScleanup ();
    return (1);

    /* End if success */

} /* end main ()

void TimerHandler ()
{
    /* Called periodically in order to check the queue status.*/

    long expiration_time
    LS_STATUS_CODE returnCode

    returnCode = VLSupdateQueuedClient (

```

```

ls_handle,
&expiration_time
(unsigned char LSFAR *) NULL
(LS_CHALLENGE LSFAR *) NULL;

/* Is the queued license available

if (returnCode == LS_SUCCESS &&
    expiration_time > 0 )
{
    if ((returnCode =
        VLSgetQueuedLicense
        (ls_handle,
         (unsigned char LSFAR *) NULL
         (LS_CHALLENGE LSFAR *) NULL)) == LS_SUCCESS)
    {
        /* Disable the application's timer and run the
           application! */

        /* Enable automatic heartbeats to the server */
        VLSdisableAutoTimer (ls_handle, VLS_ON);
    }
    else
    {
        /* Error getting the license, clean up and quit. */
        VLScleanup ();

        /* Terminate the process */
    }
}
}

```

License Queuing Functions

The following table summarizes the license queuing functions:

Table 6-1: License Queuing

Function	Description
VLSqueuedRequest() VLSqueuedRequestExt()	An integrated request for an authorized license code from the license server. Use this API to: <ul style="list-style-type: none">• Request a license, with option to queue (<i>requestFlag</i> = VLS_REQ_GET VLS_REQ_QUEUE).• Request a license without queuing (<i>requestFlag</i> = VLS_REQ_GET). This option has the same effect as calling an non-queuing API request (LSRequest(), VLSrequestExt(), etc.).• Request to be placed on the queue, even if the license server has available licenses (<i>requestFlag</i> = VLS_REQ_QUEUE).
VLSgetQueuedClientInfo()	Retrieves the current information of a queued client, such as the number of requested licenses, <i>feature_name</i> , <i>version</i> , and <i>index</i> .
VLSremoveQueuedClient()	Removes a queued client from the queue.
VLSremoveQueue()	Deletes the entire queue.
VLSgetHandleStatus()	Reports the current status of the handle.
VLSupdateQueuedClient()	Once the client has been put in the queue, it must call this API periodically to inquire its current status with the license server. Moreover, calling this function has the effect of informing the license server that the client is alive and is still seeking the license.
VLSgetQueuedLicense()	Obtains license, once it has been granted. This function is called only after a call to VLSupdateQueuedClient() reveals that a license has been granted to a queued client.
VLSinitQueuePreference()	Initializes provided queue preference structure to default values.

VLQueuedRequest() and VLQueuedRequestExt()

VLQueuedRequest() and VLQueuedRequestExt()

Syntax

```

int VLQueuedRequest(
    unsigned char    LSFAR,*license_system
    unsigned char    LSFAR,*publisher_name,
    unsigned char    LSFAR,*product_name          unsigned char) LSFAR *version
    unsigned long    LSFAR *units_reqd            unsigned char LSFAR
    *log_comment     LS_CHALLENGE LSFAR *challenge
    LS_HANDLE        LSFAR *lshandle              VLQueuePreference
    LSFAR *qPreference int LSFAR requestFlag;

int VLQueuedRequestExt(
    unsigned char    LSFAR,*license_system
    unsigned char    LSFAR,*publisher_name,
    unsigned char    LSFAR,*product_name          unsigned char) LSFAR *version
    unsigned long    LSFAR *units_reqd            unsigned char LSFAR
    *log_comment     LS_CHALLENGE LSFAR *challenge
    LS_HANDLE        LSFAR *lshandle              VLQueuePreference
    LSFAR *qPreference int LSFAR requestFlag
    VLSserverInfo LSFAR server_info;

```

Argument	Description
<i>license_system</i>	A license requested in the system. Pointer to the string which uniquely identifies a particular license system.
<i>publisher_name</i>	Refers to the name of the publisher (manufacturer) of the product. Cannot be NULL and must be unique. It is recommended that a company name and trademark be used.
<i>product_name</i>	Feature name. The name of the product requesting licensing resources. Cannot be NULL and must be unique.
<i>version</i>	<i>Version</i> for which licenses are requested. Must be unique for the associated feature.
<i>units_reqd</i>	Number of units requested to run the license. The license system verifies that the requested number of units exist and is possible to reserve those units, but no units are actually consumed at that time. The default is 1, and this value is used if NULL value is passed.

VLSQueuedRequest() and VLSQueuedRequestExt()

Argument	Description
<i>log_comment</i>	A string that is written by the license manager to the comment field of the usage log file.
<i>challenge</i>	Pointer to a <i>challenge</i> structure. The challenge-response will also be returned.
<i>lshandle</i>	Handle to the license for which the user has requested. If the user has successfully received the license, the status of the handle is VLS_ACTIVE_HANDLE. Otherwise, the client is put in the queue and the status of the handle is VLS_QUEUED_HANDLE.
<i>qPreference</i>	Pointer to the VLSQueuePreference() structure, which is used to specify the client's preference for how it wishes to be placed in the queue. After the call is made, the structure contains the values assigned by the license server when it has placed the client in the queue.
<i>requestFlag</i>	<p>Valid values are:</p> <ul style="list-style-type: none">• VLS_REQ_GET - specifies a non-queuing request (without queuing the client). If license is not available, client will not be queued.• VLS_REQ_QUEUE - specifies to queue the client (without returning with the license). Even if license is available, client will be queued. <p>If both are specified the client informs the license server to give the license, if available, otherwise queue the client. Upon return from this API, this parameter will be set to either VLS_REQ_GET (specifying the license has been granted) or, VLS_REQ_QUEUE (specifying that the client has been queued).</p>
<i>server_info</i>	Information about the server.

Description The API provides the mechanism to the calling application to ask the license server to grant a license, if available. If no license is available, the client will be queued. The client can call **VLSupdateQueuedClient()** to inquire if a license is available. Once a license is available, the client can call **VLSgetQueuedLicense()** to obtain the license.

In response, the license server will either issue the key when (and if) the license is available, put the client in the queue when the license is not available, or issue

VLQueuedRequest() and *VLQueuedRequestExt()*

an appropriate error message, which describes the cause for not being able to service the request.

The client will pass the following information to the license server:

- Time in seconds for the client to wait in the queue for the license.
- Time in seconds for the server to hold the license once it becomes available.
- Priority relative to other clients.
- The maximum position within the queue before which the client can be queued.
- The maximum position within the group queue, before which the client can be queued.

Notice that the `LS_MAX_QLEN` environment variable can override the *qPreference* structure. The end-user can put a limit on the maximum size of the queue by defining the `LS_MAX_QLEN` environment variable. This variable depends upon the availability of memory resources. The different values of `LS_MAX_QLEN` are:

- `LS_MAX_QLEN` not set. Client preference is applied.
- `LS_MAX_QLEN` = -1. Client preference is ignored and the client is always queued.
- `LS_MAX_QLEN` = 0. Queue is disabled and no clients will be put in the queue.
- `LS_MAX_QLEN` > 0. Overrides the client's preference.

Similarly variable `LS_MAX_GRP_QLEN` will override the setting of the max group wait time in the *qPreference* structure.

Variables `LS_MAX_WAIT_SEC` and `LS_MAX_HOLD_SEC` override the max wait time and max hold time elements of the *qPreference* structure.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>request_flag</i> specifies queuing but <i>qPreference</i> is NULL. <i>lshandle</i> is NULL. <i>challenge</i> argument is non-NULL, but cannot be understood.
VLS_APP_UNNAMED	<i>product_name</i> is NULL version is NULL
VLS_NO_LICENSE_GIVEN	<i>units_reqd</i> is zero. Invalid handle specified. Generic error indicating that the license is not granted.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for the requested feature to grant a license.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
LS_LICENSE_EXPIRED	License has expired.
VLS_NOMORE_QUEUE_RESOURCES	Queue is full.
VLS_APP_NODE_LOCKED	Requested feature is node locked, but request was issued from an unauthorized machine.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.

VLQueuedRequest() and VLQueuedRequestExt()

VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> is specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_MAJORITY_RULE_FAILURE	Majority rule failure prevents token from being issued.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
VLS_ELM_LIC_NOT_ENABLE	The license was converted using the license conversion utility. (From a 5.x license), but the DLT process is not running.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetQueuedClientInfo()

Syntax

```
int VLSgetQueuedClientInfo
    unsigned char    LSFAR, *feature_name
    unsigned char    LSFAR *version
    int              index
    VLSQueuedClientInfo LSFAR *client_info ;
```

Argument	Description
<i>feature_name</i>	Feature name of the client for which we are requesting information.
<i>version</i>	Version for which licenses are requested. Must be unique for the associated feature.
<i>index</i>	Index of the client with the license server, for a particular feature.
<i>client_info</i>	The structure in which information will be returned. Pointer to the VLSQueuedClientInfo() structure, which specifies the client information.

Description Fills the structure pointed by *client_info* to a structure containing the current information of a queued client identified by specified *feature_name*, *version*, and *index*.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>client_info</i> parameter is NULL. <i>index</i> is negative. Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	<i>feature_name</i> is NULL <i>version</i> is NULL
VLS_NO_LICENSE_GIVEN	Finished retrieving client information for all the clients.

VLRemoveQueuedClient()

VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
VLS_MULTIPLE_VENDORID_FOUND	The license server has licenses for the same feature and version from multiple vendors. It is ambiguous which feature is requested.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLRemoveQueuedClient()

Syntax

```
int VLRemoveQueuedClient
    unsigned char    LSFAR, *feature name
    unsigned char    LSFAR *version
    int              qkey_id      ;
```

Argument	Description
<i>feature_name</i>	Feature name of the client for which we are requesting information.

Argument	Description
<i>version</i>	<i>Version</i> for which licenses are requested. Must be unique.
<i>qkey_id</i>	Identifier of the client queue, which needs to be removed.

Description This API provides an administrative mechanism to remove a queued client. **VLSremoveQueuedClient()** will be available to:

- The user who started the license server, which actually signifies when the client was put in the queue.
- The root/administrator account.
- The user-account that originally goes to the queue placement.

Internally, this API will send a message to signal the license server that a specified client in the queue for a specified feature should be removed.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>q_key_id</i> parameter cannot be negative. Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	<i>feature_name</i> is NULL <i>version</i> is NULL
VLS_NO_SUCH_CLIENT	License server does not have the specified client
VLS_CLIENT_NOT_AUTHORIZED	Client is not authorized to make the specified request.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.

VLSremoveQueue()

VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSremoveQueue()

Syntax `int VLSremoveQueue` `unsigned char`
 `LSFAR, *feature name`
 `unsigned char LSFAR *version ;`

Argument	Description
<i>feature_name</i>	Identifies the license whose queue needs to be removed.
<i>version</i>	Version for which licenses are requested. Must be unique.

Description This API will provide a mechanism to delete the complete queue for a specified license.

VLSremoveQueue() will be available to:

- The user-account who started the license server, which actually signifies when the client was put in the queue.
- the root/administrator account.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	<i>feature_name</i> is NULL <i>version</i> is NULL
VLS_CLIENT_NOT_AUTHORIZED	Client not authorized to remove queue.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_NO_SERVER_FILE	The license server has not been set and is unable to determine which license server to use.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetHandleStatus()

Syntax int VLSgetHandleStatus
LS_Handle *lshandle* ;

Argument	Description
<i>lshandle</i>	Identifies the handle previously returned by VLSQueuedRequest() .

VLSupdateQueuedClient()

Description Reports the current status of the handle.

Returns Returns the following error codes:

LS_BADHANDLE	Invalid handle. Handle is already released and destroyed from previous license operations.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
VLS_AMBIGUOUS_HANDLE	<i>lshandle</i> is an ambiguous handle; it is not exclusively active or exclusively queued.
VLS_ACTIVE_HANDLE	<i>lshandle</i> is an active handle.
VLS_QUEUED_HANDLE	<i>lshandle</i> is a queued handle.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSupdateQueuedClient()

Syntax

```
int VLSupdateQueuedClient(  
    LS_HANDLE      lshandle, longLSFAR  
    *absExpiryTime, unsigned charLSFAR *unused1  
    LS_CHALLENGE   LSFAR *unused2    ;
```

Argument	Description
<i>lshandle</i>	The handle previously returned by VLSqueuedRequest() . The status of the handle must be VLS_QUEUED_HANDLE or an error will occur.

Argument	Description
<i>absExpiryTime</i>	Once the license is available with the license server, the next call to this API returns in this parameter, the absolute expiry time before which the client should get the license using VLGetQueuedLicense() . If any call to VLSupdateQueuedClient() returns a non-negative value in this parameter, then the license has been granted and set aside for the client. There is no need to continue its periodic call to this function. The next step is to obtain the license by calling VLGetQueuedLicense() . Possible values for <i>absExpiryTime</i> are: <ul style="list-style-type: none"> • Zero = license is not available. • Non-zero = license is available and will stop calling the API.
<i>unused1</i>	Uses NULL as the value.
<i>unused2</i>	

Description The client calls this API, requesting the license server to put him in the queue. Once the client has been put in the queue, it must call this API periodically to inquire its current status with the license server. Moreover, it also informs the license server that, he is alive and is seeking the license.

Notice, the client will be needs to make at least one queue update, within 5 minutes of the previous queue-update or the request to queue itself. This is imperative so as to make the license server aware of the active clients. If the license server does not receive an update request from a client within 5 minutes of the last queue-update, it will then assume the client to be inactive and remove the client from the queue.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>absExpiryTime</i> is NULL. Handle cannot be active. <i>challenge</i> argument is non-NULL, but cannot be understood.
LS_BADHANDLE	Invalid handle.

VLUpdateQueuedClient()

LS_LICENSETERMINATED	Cannot update license because license has already expired.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.
LS_NOLICENSESAVAILABLE	All licenses are in use.
LS_LICENSE_EXPIRED	License has expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_FINGERPRINT_MISMATCH	User or machine excluded from accessing the requested feature.
VLS_APP_NODE_LOCKED	Feature is node locked, but update request was issued from an unauthorized machine.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_INVALID_DOMAIN	The domain of the license server is different from that of the client.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetQueuedLicense()

Syntax

```
int VLSgetQueuedLicense(
    LS_HANDLE    lshandle,
    *log_comment,
    LSFAR *challenge    );
```

unsigned charLSFAR
LS_CHALLENGE

Argument	Description
<i>lshandle</i>	The handle previously returned by VLSQueueRequest() . The status of the handle must be VLS_QUEUED_HANDLE and the last call to VLSupdateQueuedClient() must have reported that the licenses have been made available with the license server.
<i>log_comment</i>	A string that is written by the license manager to the comment field of the usage log file. Pointer to a <i>challenge</i> structure. The challenge-response will also be returned.
<i>challenge</i>	The challenge-response for this operation.

Description Once the queued client identifies that the required licenses are made available with the license server, it calls this API to fetch the license. This API will be passed from the client library handle only and, internally, it will send all the memorized information to the license server. On return it will provide a valid client-handle value that will be used in later API calls.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>challenge</i> argument is non-NULL, but cannot be understood.
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	An error occurred in the use of an internal buffer.
VLS_NO_LICENSE_GIVEN	Generic error indicating that the license is not granted.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature.

*VL*SgetQueuedLicense()

LS_LICENSE_EXPIRED	License has expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
LS_NOLICENSESAVAILABLE	All licenses are in use.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_FINGERPRINT_MISMATCH	Client-locked. Locking criteria does not match.
VLS_APP_NODE_LOCKED	Requested feature is node locked, but request was issued from unauthorized machine.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has the license.
VLS_INVALID_DOMAIN	The domain of the license server is different from that of client.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_BAD_SERVER_MESSAGE	Message returned by the license server could not be understood.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the license operation.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
VLS_ELM_LIC_NOT_ENABLE	The license was converted using the license conversion utility. (From a 5.x license), but the DLT process is not running.

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VL_SinitQueuePreference()

Syntax `int VLSinitQueuePreference`
 `VLSqueuePreference *qPreference` ;

Argument	Description
<i>qPreference</i>	Pointer to the VLQueuePreference() structure, which specifies the client preference for getting into the queue. After the call is made, the structure signifies the actual values, which the license server allocates to the client while putting him in the queue.

Description Initializes the `VLSQueuePreference()` structure to default values.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>qPreference</i> is NULL.
-------------------	-----------------------------

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLInitQueuePreference()

Chapter 7 - Commuter License API

Commuter licensing is the capability to temporarily check out an authorization to use a protected application from a SentinelLM license server to a portable computer. The most common use of this feature is to allow use of a protected application on a laptop computer that will be disconnected from the network.

Commuter License Related Functions

The following table summarizes the commuter license related functions:

Table 7-1: Commuter License Related Functions

VLSgetCommuterInfo()	Returns the commuter license information.
VLSgetAndInstallCommuter-Code()	Obtains the commuter code from the license server and issues the commuter authorization to the client side persistence database
VLSuninstallAndReturnCommuter-Code()	Removes the commuter authorization from the client side persistence database and returns the token to the license server.

VLSgetCommuterInfo()

Syntax int VLSgetCommuterInfo
 unsigned char **feature_name*,

*VL*SgetAndInstallCommuterCode()

unsigned char	<i>*version</i>	<i>intindex</i>
VLSCommuterInfo	<i>*commuter_info,</i>	

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>index</i>	Used to specify a particular client.
<i>commuter_info</i>	Displays the number of clients for commuter licenses.

Description Returns the commuter license information.

VLsgetCommuterInfo() can be used two ways:

1. Specify *feature_name* and *version* as non-NULL and API will return information about this feature. API will ignore the *index* argument.
2. If *feature_name* is NULL, then API will return information about the *index* feature in the persistence database. API will ignore the *version* argument. API will be called until it return VLS_NO_MORE_FEATURES by incrementing the *index* every time.

Returns The status code, VLScg_SUCCESS, is returned if successful. For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSgetAndInstallCommuterCode()

```

Syntax      int VLSgetAndInstallCommuterCode
                unsigned char      *feature_name,
                unsigned char      *feature_version
                                long*units_reqd
                                int*duration      int
                *lock_mask
                *log_comment
                *challenge,

```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>feature_version</i>	Version of the feature.

Argument	Description
<i>units_reqd</i>	Number of units required to run the license. The license system verifies that the requested number of units exist and may reserve those units, but no units are actually consumed at that time.
<i>duration</i>	Displays the number of clients for commuter licenses.
<i>lock_mask</i>	Mask defining which fields are to be used for locking. On entry, <i>lock_mask</i> specifies the locking-criteria that should be used for looking the commuter-code. If a zero is given, the API will lock the code to Disk ID (windows), otherwise it will lock to host name. Notice, the API will replace the zero with <i>lock_mask</i> for Disk ID or host name before sending this value to the license server.
<i>log_comment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.
<i>challenge</i>	The challenge-response for this operation. Pointer to a <i>challenge</i> structure. The challenge-response will also be returned in this structure.

Description Obtains the commuter code from the license server and installs the stand-alone commuter authorization at the client.

Returns The status code, VLScg_SUCCESS, is returned if successful. Otherwise, it will return the following error codes:

VLS_CALLING_ERROR	<i>duration</i> is NULL <i>lock_mask</i> is NULL.
-------------------	--

For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

VLSuninstallAndReturnCommuterCode()

Syntax int VLSuninstallAndReturnCommuterCode
 unsigned char **feature_name*,

VLUninstallAndReturnCommuterCode()

unsigned char **feature_version* unsigned char
**log_comment*

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>feature_version</i>	Version of the feature.
<i>log_comment</i>	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.

Description Uninstalls the commuter authorization from the client and returns the commuter authorization to the license server.

Returns The status code, *VLScg_SUCCESS*, is returned if successful. For a complete list of the error codes, “Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions” on page 295.

Chapter 8 - Usage Log Functions

The Usage log functions provide capability of controlling and manipulating the usage log file.

The following table summarizes the usage log functions:

Table 8-1: Usage log functions

Function	Description
VLSchangeUsageLogFileName()	This API changes the name of the existing usage log file. This change can be done while the file is being used.
VLSgetUsageLogFileName()	API determines the name of the existing usage log file.

VLSchangeUsageLogFileName()

Syntax

```
int VLSchangeUsageLogFileName  
    char *hostName, char  
    *newFileName
```

Argument	Description
<i>hostName</i>	The host name of the computer containing the license server that is using the log file.
<i>newFileName</i>	The new name you want to use for the log file.

Description Changes the name of the existing usage log file. This change can be done while the file is being used.

*VL*SgetUsageLogFileName()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VL

SgetUsageLogFileName()

Syntax int VL

SgetUsageLogFileName (
char *hostName, char*fileName

Argument	Description
hostName	The host name of the computer containing the license server that is using the log file.
fileName	The name of the existing usage log file is returned in this argument.

Description Determines the name of the existing usage log file.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

Chapter 9 - Utility Functions

The utility functions are only available on UNIX platforms.:

Table 9-1: Utility Functions

Function	Description
VLSscheduleEvent()	Schedules eventhandler to be awakened after so many seconds. It handles only SIGALRM signal.
VLSdisableEvents()	Disables the events scheduled. To disable a particular event pass the event handler function name as the argument. To disable all the events pass NULL as argument.
VLSeventSleep()	Disables the feature for an allotted time.

VLSscheduleEvent()

Syntax

```
int VLSscheduleEvent (
    unsigned long          *seconds,
                                long          *repeat_event
                                void*eventHandler
```

Argument	Description
<i>seconds</i>	Time interval in <i>seconds</i> .
<i>eventHandler</i>	Signal handler.
<i>repeat_event</i>	Number of event repetitions.

Description This function is called for scheduling *eventHandler* to be awakened after so many *seconds*. Handles only SIGALRM signal.

VLDisableEvents()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLDisableEvents()

Syntax int VLDisableEvents (
void **eventHandler* ,

Argument	Description
<i>eventHandler</i>	Signal handler.

Description This function is called for disabling the events scheduled. To disable a particular event pass the event handler function name as the argument. To disable all the events pass NULL as argument.

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLSeventSleep()

Syntax int VLSeventSleep (
void VLSeventSleep (unsigned int seconds)

Argument	Description
<i>seconds</i>	Time in <i>seconds</i> to sleep.

Description This function is called for disabling the license operations for an allotted time and interferes with the system alarms.
VLSeventSleep() must be used in conjunction with **VLDisableAutoTimer()**.

VLSeventSleep()

Returns The status code, LS_SUCCESS, is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see “Appendix D - Error and Result Codes for License Generation Functions” on page 291.

VLSeventSleep()

Appendix A - Sample Applications

Each platform has an examples directory. For UNIX platforms this includes a file called *Makefile*. *Makefile* can be used to build the sample programs, utilities, and to customize parts of SentinelLM. For Windows platforms, the file is called *samples32.mak*.

When you run *sample32.mak*, use the following commands:

```
nmake /f sample32.mak sample-program
```

This appendix lists the available sample programs, utilities, and SentinelLM components.

Sample Program Summary

The following table lists the sample programs, the features illustrated in each, and on which platforms the programs are available:

Table A-1: Sample Programs, Features, and Platforms

Program	Features	Platforms
bounce	Simple function macros	Windows NT, Windows 95/98
dots1	Simple function macros	UNIX
qbounce	Queueing API	Windows NT, Windows 95/98
timer	Simple function macros and using timer signals	UNIX
tutor1	Simple function macros	UNIX

Table A-1: Sample Programs, Features, and Platforms

Program	Features	Platforms
single	Single-call licensing	UNIX
stars1	LSAPI function calls and error handlers	UNIX

Note Programs ending in “1” also have “0” versions without licensing.

Customization Samples

On the UNIX platforms the following components/files are available:

Table A-2: Customization Sample Files

Component	File(s)
linking	<i>Makefile</i>
converting license codes	<i>rdctoenc.o, enctordc.o</i>
the license manager	<i>server.o</i>
lsdecode	<i>lsde.o</i>
lslic	<i>lslic.c</i>
lsmon	<i>lsmon.c</i>
lswhere	<i>lswhere.c</i>
distcgen	<i>distcgen.o</i>
the code generator	<i>lscgen.o</i>
enctordc	<i>enctordc.o</i>
rdctoenc	<i>rdctoenc.0</i>
Challenge-response	<i>crexamp.c, chalresp.[c h], md4.[c h]</i>

Appendix B - Customization Features

The SentinelLM package is optionally shipped with a number of precompiled object modules to enable you to re-link the license manager and the code generator executables, and override certain predefined SentinelLM characteristics.

There are compatibility issues for object files generated by different versions of compilers on Microsoft Windows platforms. Therefore, *server.o* and *lscgen.o* files are not included in the Windows distribution. Please contact Technical Support (see page xix) for information about customization tools availability for your version of Windows developer platforms.

The following table summarizes the customizing functions:

Table B-1: Customizing Functions

VLServerVendorInitialize()
VLSeventAddHook()
VLSconfigureTimeTamper()
VLSisClockSetBack()
VLSencryptLicense()
VLSdecryptLicense()
VLSencryptMsg()
VLSdecryptMsg()
VLSchangePortNumber()
getCustomHostId()

Note On the UNIX platform, creating customized executables requires the use of the *Makefile* in the *examples* directory and various object files provided in the *lib* directory of the shipped software. If you customize your license server, ship it under a different name from the original and change the port number on which it receives network messages so that your customized server does not interfere with other vendors' license servers that may be running at a customer's site.

All customized encryption and decryption functions for the network licenses must adhere to the following rules:

1. No **malloc()** or **free()** calls are allowed in the functions.
2. No signal-unsafe calls are allowed.
3. All strings must be NULL-terminated.
4. All functions must return 0 on success.
5. Buffers are guaranteed to be at least 500 characters long. Lengths of output strings need not be the same as the input strings.

To build your customized functions, copy your source files to *c:\Program Files\Rainbow Technologies\sentLM\MsvcDev\custom*. In this directory you will find the Makefile *custom32.mak*. Make a copy of this file and name it **MAKEFILE**. Edit this file. Add your customized object files in the following section:

```
# For now, use the default functions from the SentinelLM library:
ENCRYPT_LIC_OBJS =
DECRYPT_LIC_OBJS =
ENCRYPT_MSG_OBJS =
DECRYPT_MSG_OBJS =
CHANGE_PORT_OBJS =
CHANGE_HOSTID_OBJS =
TIME_TAMPER_OBJS =
SERVER_HOOK_OBJS =
```

Go to the DOS prompt and run *make*.

Initializing the Server

These functions are called by the server during server initialization. This is where calls to **VLSeventAddHook()** should be placed in order to configure the server to consult vendor event handler functions.

VLServerVendorInitialize()

Client	Server	Static Library	DLL
	✓	✓	

Initializes the server.

Syntax `LSERV_STATUS VLServerVendorInitialize (void);`

This function has no arguments.

VLSeventAddHook()

Client	Server	Static Library	DLL
	✓	✓	

Registers an event handler with the server.

Syntax `LSERV_STATUS VLSeventAddHook (
 int eventName,
 int (*handlerFuncPtr)(VLShandlerStruct *, char *, char *, int),
 char *identifier,);`

Argument	Description
<i>eventName</i>	Specifies the type of event. Handler function will be called LS_REQ_PRE right before the license request is processed by the server. Handler function will be passed LS_REQ_POST right after the license request is processed by the server. Handler function will be called LS_REL_PRE right before the license release is processed by the server. Handler function will be passed LS_REL_POST right after the license release is processed by the server.
<i>(*handlerFuncPtr)</i> <i>(VLShandlerStruct *, char*, char *, int)</i>	The function pointer.
<i>identifier</i>	The client <i>identifier</i> to match.

Description Hooks are based on events. For each event, there is a pre-event hook and a post-event hook.

Currently the only events with hooks are license request and license release. So you can have a hook function BEFORE a license request is processed by the server or AFTER a request is processed. In the “pre” hook, you can decide on the licensing action such as looking up external information before granting a request. In the post hook, you cannot change the license decision but can provide custom information to be passed to the client.

Note You can use only one hook and do not have to use all the hook functions.

The file below for this example can be found in *srhkdemo.c*. The entire sample hook project can be found in the following files: *reqprhk1.c*, *reqpshk1.c*, *relprhk1.c*, *relpshk1.c*, *reqprhk2.c*, *reqpshk2.c*, and *relprhk2.c*. The client portion of the project can be found in *hookdemo.c*.

```

/*****
/*
/* Copyright (C) 1999 Rainbow Technologies, Inc.
/* All Rights Reserved
/*
*****/

#include "lservcst.h"
extern int LSReqPreHook1(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSReqPostHook1(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSRelPreHook1(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSRelPostHook1(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSReqPreHook2(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSReqPostHook2(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSRelPreHook2(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
extern int LSRelPostHook2(VLHandlerStruct *handleStruct, char *inBuf, char
*outBuf, int outBufSz);
LSERV_STATUS VLServerVendorInitialize(void) {
#ifdef _VWIN31X_
    VLSeventAddHook(LS_REQ_PRE,
        LSReqPreHook1,
        "Hook1");
    VLSeventAddHook(LS_REQ_POST,
        LSReqPostHook1,
        "Hook1");
    VLSeventAddHook(LS_REL_PRE,
        LSRelPreHook1,
        "Hook1");
    VLSeventAddHook(LS_REL_POST,
        LSRelPostHook1,
        "Hook1");
    VLSeventAddHook(LS_REQ_PRE,
        LSReqPreHook2,
        "Hook2");
    VLSeventAddHook(LS_REQ_POST,
        LSReqPostHook2,
        "Hook2");
    VLSeventAddHook(LS_REL_PRE,
        LSRelPreHook2,
        "Hook2");
    VLSeventAddHook(LS_REL_POST,

```

```
        LSRelPostHook2,  
        "Hook2");  
#endif  
return(LSERV_STATUS_SUCCESS);  
}
```

Protecting Against Time Clock Changes

Software-based license protection schemes may break down if the end user changes the system time. The SentinelLM license server can be configured to detect tampering of the system clock.

SentinelLM checks about 500 system files (in strictly read-only mode) to determine if the system clock of the machine it is running on has been set back in order to use an expired license. It does this on startup, and periodically thereafter. Checking takes about 10 to 20 seconds.

SentinelLM calls the function **VLsconfigureTimeTammer()** before performing any time tamper checks. This configuration function can be used to modify the default behavior of SentinelLM regarding time tamper checking. You need to perform the following steps:

1. Write your own **VLsconfigureTimeTammer()** function which takes the following arguments, and writes valid values into *all* of the arguments.
2. If you plan to use your own clock tamper checking function, you should write another function **VLsClockSetBack()** which returns 0 if the system clock has not been set back, and 1 otherwise.
3. In the *Makefile* in the examples directory [UNIX], modify the **TIME_TAMPER_OBJ** macro so that its value is the name of the object file containing your new function(s).
4. Relink the license server (or your application if in stand-alone mode).

VLSconfigureTimeTamper()

Client	Server	Static Library	DLL
	✓	✓	

Syntax

```
void VLSconfigureTimeTamper (
    VLSactionOnTmTamper *actionOnTmTamper,
    VLStmTamperMethod *tmTamperMethod,
    int *gracePeriod,
    int *percentViolations,
    int *numViolationsForError
);

int VLSisClockSetBack( );
```

Types **VLSactionOnTmTamper()** and **VLStmTamperMethod()** are defined in *lserv.h*:

```
typedef enum {VLS_CONT_AFTER_TM_TAMPER, VLS_EXIT_AFTER_TM_TAMPER}
    VLSactionOnTmTamper;

typedef enum {VLS_ENABLE_DEFAULT_TM_TAMPER,
    VLS_DISABLE_DEFAULT_TM_TAMPER}
    VLStmTamperMethod;
```

In the table below, default values are indicated in brackets ([]).

Argument	Description
<i>actionOnTmTamper</i>	Whether to exit from the license manager (or your application if in stand-alone mode) once time clock tampering is detected. [VLS_CONT_AFTER_TM_TAMPER]
<i>tmTamperMethod</i>	Whether to use the SentinelLM built-in system clock tamper checking function, or use one provided by you. [VLS_ENABLE_DEFAULT_TM_TAMPER]

Argument	Description
<i>gracePeriod</i>	Useful only in case <i>tmTamperMethod</i> is VLS_ENABLE_DEFAULT_TM_TAMPER. If SentinelLM finds the system clock has been set back by less than <i>gracePeriod</i> seconds, it will not count the offending system file as a violation.
<i>percentViolations</i>	Percentage of system files that must be found in violation of the grace period before concluding that the system clock has been set back. Pass the value of 0 for this argument or ignore the functionality..
<i>numViolationsForError</i>	Number of system files that must be found in violation of the grace period before concluding that the system clock has been set back. [5] 0 to ignore this.

The default algorithm uses a grace period of 86,400 seconds (1 day) and allows 1% of the files to violate the grace period.

Note Out of *percentViolations* and *numViolationsForError*, the lower evaluated value will be used.

VL SisClockSetBack()

Client	Server	Static Library	DLL
	✓	✓	

Notifies the license server to check whether the clock has been set back.

Syntax int VLSisClockSetBack();

This function has no arguments.

Description This function is called only in case the **VL SconfigureTimeTamper()** function returns *tmTamperMethod* to be VLS_DISABLE_DEFAULT_TM_TAMPER.

Returns Returns 0 if the clock has not been set back.

Encrypting License Codes

License code encryption can be modified to add an additional layer of encryption/decryption security. License encryption and decryption is used by the license server, the code generator, and the SentinelLM utility, **lsdecode**. All three programs must be re-linked. Licensed applications do not encrypt or decrypt license codes. Client applications need not be re-linked.

Note Encryption is not available for stand-alone licenses.

VLSEncryptLicense()

Client	Server	Static Library	DLL
	✓	✓	

Encrypts license codes.

Syntax

```
int VLSEncryptLicense ( )
char    *origText;
char    *encryptedTextBuffer;
int     buffSize;
```

Argument	Description
<i>origText</i>	The original license code.
<i>encryptedTextBuffer</i>	The encrypted license code to be returned.
<i>buffSize</i>	Size of the encrypted text buffer.

Description **VLSEncryptLicense()** will always receive any of the ASCII character set in its input text string. Since the output of this function will be written directly to the code generator's output file as an encrypted license code, this function must not generate any unprintable or special characters.

The function may generate any printable ASCII characters other than:

Character	Hex Value	Description
#	0x23	Pound sign or number sign or hash mark.
\n	0x0A	Backslash-n.
\t	0x09	Backslash-t.
(0x28	Opening parenthesis.
)	0x29	Closing parenthesis.
-	0x2D	Hyphen or dash or minus sign.
,	0x2C	Comma.

In fact, by generating a larger character set than the input, the encryption algorithm can generate shorter license codes. To add another layer of encryption and decryption follow these steps:

1. Write custom **VLSEncryptLicense()** and **VLSDecryptLicense()** functions in separate source files.
2. In the *examples* directory of the distribution tree, the example *Makefile* can be used to re-link the license server, the code generator, and **lsdecode** directly. In the example *Makefile*, set the variable, **ENCRYPT_LIC_OBJ**, to the object file containing **VLSEncryptLicense()**, and **DECRYPT_LIC_OBJ** to the object file containing **VLSDecryptLicense()**.
3. Issue the **make** commands for the license server, the code generator, **lsdecode**, and the distributor's code generator (optional).

Returns 0 if successful; other value on failure.

Example file:

```

/*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
/*****
/* Usage of VLSEncryptLicense() */
#include <stdio.h>

```

```

#include <string.h>
#include "ltest.h"
int VLSencryptLicense(outputString,inputString,size)
char outputString[MAX_LIC_SIZE];
char inputString[MAX_LIC_SIZE];
int size;
{
    int j=0;
    fprintf(stdout,"ENCRYPTING LICENSE\n");
    while ((outputString[j]!='\0') &&(outputString[j+1]!='\0') && (outputString[j]!='\n')
&&(outputString[j+1]!='\n') && (j<size)) {
        inputString[j]=outputString[j+1];
        inputString[j+1]=outputString[j];
        j=j+2;
    }
    inputString[j]=outputString[j];
    j++;
    if (outputString[j]=='\0') {inputString[j]=outputString[j]; j++;}
    if (outputString[j]=='\n') {inputString[j]=outputString[j]; j++;}
    return(0);
}

```

VLSdecryptLicense()

Client	Server	Static Library	DLL
	✓	✓	

Decrypts license codes.

Syntax

```

int VLSdecryptLicense (
    char      *origText;
    char      *decryptedTextBuffer;
    int       buffSize;

```

Argument	Description
<i>origText</i>	The original license code.
<i>decryptedTextBuffer</i>	The decrypted license code to be returned.
<i>buffSize</i>	Size of the decrypted text buffer.

Encrypting Messages

Description See **VLSEncryptLicense()** above.

Example file:

```
/*
*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
*****
/* Usage of VLSDecryptLicense() */
#include <stdio.h>
#include <string.h>
#include "ltest.h"
int VLSDecryptLicense(outputString,inputString,size)
char outputString[MAX_LIC_SIZE];
char inputString[MAX_LIC_SIZE];
int size;
{
    int j=0;
    fprintf(stdout,"DECRYPTING LICENSE\n");
    while ((outputString[j]!='\0') &&(outputString[j+1]!='\0') && (outputString[j]!='\n')
&&(outputString[j+1]!='\n') && (j<size)) {
        inputString[j]=outputString[j+1];
        inputString[j+1]=outputString[j];
        j=j+2;
    }
    inputString[j]=outputString[j];
    j++;
    if (outputString[j]=='\0') {inputString[j]=outputString[j]; j++;}
    if (outputString[j]=='\n') {inputString[j]=outputString[j]; j++;}
    return(0);
}
```

Encrypting Messages

All SentinelLM network communication is encrypted. However, for added security an additional layer of encryption and decryption can be added. Customizing involves changes to both the license server and the client application.

VLSEncryptMsg()

Client	Server	Static Library	DLL
✓	✓	✓	

Encrypts messages.

Syntax

```
int VLSEncryptMsg (
    char    *origText;
    char    *encryptedTextBuffer;
    int     buffSize;
```

Argument	Description
<i>origText</i>	The original message text.
<i>encryptedTextBuffer</i>	The encrypted message text.
<i>buffSize</i>	Size of the encrypted text buffer.

Description **VLSEncryptMsg()** can receive any ASCII characters as its input text string. The function can produce any ASCII characters other than \0 (0x0). To add another layer of encryption and decryption follow these steps:

1. Write custom **VLSEncryptMsg()** and **VLDecryptMsg()** functions in separate source files.
2. In the *examples* directory of the distribution tree, the example *Makefile* can be used to re-link the license server directly and edited to link with the application to be licensed using the new message encryption. In the example *Makefile*, set the variable, `ENCRYPT_MSG_OBJ`, to the object file containing **VLSEncryptMsg()**, and `DECRYPT_MSG_OBJ` to the object file containing **VLDecryptMsg()**.
3. Issue the **make** commands for the license server and the application. The client application must be incrementally linked with the new object files before linking with the SentinelLM client library.

Returns 0 if successful; other value on failure.

Example file:

```

/*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
/*
*****/
/* Usage of VLSencryptMsg() */
#include <stdio.h>
#include <string.h>
#include "ltest.h"
int VLSencryptMsg(outputString,inputString,size)
char outputString[MAX_MSG_SIZE];
char inputString[MAX_MSG_SIZE];
int size;
{
    int j=0;
    fprintf(stdout,"encrypting MESSAGE\n");
    while ((outputString[j]!='\0') &&(outputString[j+1]!='\0') && (outputString[j]!='\n')
&&(outputString[j+1]!='\n') && (j<size)) {
        inputString[j]=outputString[j+1];
        inputString[j+1]=outputString[j];
        j=j+2;
    }
    inputString[j]=outputString[j];
    j++;
    if (outputString[j]=='\0') {inputString[j]=outputString[j]; j++;}
    if (outputString[j]=='\n') {inputString[j]=outputString[j]; j++;}
    return(0);
}

```

VLSdecryptMsg()

Client	Server	Static Library	DLL
✓	✓	✓	

Decrypts messages.

Syntax

```

int VLSdecryptMsg (
char      *origText;

```



```
char      *decryptedTextBuffer;
int       buffSize;
```

Argument	Description
<i>origText</i>	The original message text.
<i>decryptedTextBuffer</i>	The decrypted message text.
<i>buffSize</i>	Size of the decrypted text buffer.

Description See **VLScryptMsg()** on the previous page.

Example file:

```

/*****
/*
/*      Copyright (C) 1999 Rainbow Technologies, Inc.      */
/*      All Rights Reserved      */
/*
*****/
/* Usage of VLScryptMsg() */
#include <stdio.h>
#include <string.h>
#include "ltest.h"
int VLScryptMsg(outputString,inputString,size)
char outputString[MAX_MSG_SIZE];
char inputString[MAX_MSG_SIZE];
int size;
{
    int j=0;
    fprintf(stdout,"decrypting MESSAGE \n");
    while ((outputString[j]!='\0') &&(outputString[j+1]!='\0') && (j<size)) {
        inputString[j]=outputString[j+1];
        inputString[j+1]=outputString[j];
        j=j+2;
    }
    inputString[j]=outputString[j];
    j++;
    if (outputString[j]=='\0') {inputString[j]=outputString[j];}
    return(0);
}

```

Changing the Default Port Number

This requires separate changes to the license server and the licensed application.

VLChangePortNumber()

Client	Server	Static Library	DLL
	✓	✓	

Changes the port number.

Syntax

```
int VLChangePortNumber ( )  
    int currentPort;
```

Argument	Description
<i>currentPort</i>	Current port number.

Description Sets port number to *newPort*. This function is called only once, at license server start-up time.

To customize the license server:

1. Write a custom **VLChangePortNumber()** function in a separate source file.
2. In the *examples* directory of the distribution tree, the example *Makefile* can be used to re-link the license server directly. In the example *Makefile*, set the variable, `CHANGE_PORT_OBJ`, to the object file containing **VLChangePortNumber()**.
3. Issue the **make** commands for the license server.

The licensed application can obtain or reset its port number through the client library function calls, **VLSgetServerPort()** and **VLSsetServerPort()**. These set-up functions must be called before making a request.

Returns 0 if successful; other value on failure.

Note Optionally, you may change the port number by using the port switch when starting the license server.

Example file:

```

/*****
/*
/* Copyright (C) 1999 Rainbow Technologies, Inc.
/* All Rights Reserved
/*
/*****
#include "lservcst.h"
#include "lserv.h"
#include <stdio.h>
#ifdef __STDC__
int VLSchangePortNumber( int newPort)
#else
int VLSchangePortNumber(newPort)
int newPort;
#endif
{
    newPort=6000;
    return(newPort);
}

```

Customizing the Host ID

SentinelLM provides a developer with the capability to have a client send a customized fingerprint along with standard fingerprints as determined by the client library.

In making a request for a key for a particular feature/version, the client sends the information about the fingerprints (IP Address, host name, PROM ID etc.) of its host machine. This fingerprint information is then compared against the fingerprint information available with the server, through the license string for that feature/version.

Customizing a host ID consists of performing the following steps:

- Create the custom host ID function

- Register the custom host ID function on the server
- Register the custom host ID function on the client
- Build the server
- Create an updated client ID generator

Creating the Custom Host ID Function

The first step to implement the customized fingerprint is to write a custom host ID (basically a customized fingerprint) function. This function must return a “long” value, based on the customized logic that is unique for each host. The following is an example of generating a custom host ID. In this example, the custom host ID is being generated by converting each of the standard machine fingerprints to integer values, and then adding them all together.

```
long getCustomHostId()
{
    VLSmachineID LSFAR machineID;
    unsigned long LSFAR lock_selector_out,temp1, temp2;
    long temp;

    VLSinitMachineID(&machineID); /*Set default values.*/

    /*Get the locking information for all available locking mechanisms*/

    VLSgetMachineID(VLS_LOCK_ID_PROM|VLS_LOCK_IP_ADDR|VLS_LOCK_DISK_ID|
        VLS_LOCK_HOSTNAME|VLS_LOCK_ETHERNET|VLS_LOCK_NW_IPX|
        VLS_LOCK_NW_SERIAL|VLS_LOCK_PORTABLE_SERV,&machineID,&lock_selector_out);

    temp2 = machineID.id_prom;
    temp1 = 0;

    /*Check to see if we were able to generate locking info for each criteria. If so, convert that info to an
    unsigned long and add it to the sum */
    if ((machineID.ip_addr != NULL) && (machineID.ip_addr[0] != '\0'))/*checking for presence*/
        temp1 = strtoul(machineID.ip_addr, (char **)NULL, 10);

    temp2 += temp1 + machineID.disk_id;
    if ((machineID.host_name != NULL) && (machineID.host_name[0] != '\0'))
        temp1 = strtoul(machineID.host_name,(char **)NULL,10);
    temp2 += temp1;
    if ((machineID.ethernet != NULL) && (machineID.ethernet[0] != '\0'))
        temp1 = strtoul(machineID.ethernet, (char **)NULL, 10);
```

```

temp2 += temp1 + machineID.nw_ipx + machineID.nw_serial;
if ((machineID.portserv_addr != NULL) && (machineID.portserv_addr[0] != '\0'))
temp1 = strtoul(machineID.portserv_addr, (char **)NULL, 10);
temp2 += temp1;
temp2 = temp2 / 200; /*just to customise hostid */

temp = temp2 + 10;

return temp; /*return long */
}

```

Registering the Custom Host ID Function on the Server

The function used to register the function with the server is **VLssetHostIdFunc()**, which we call from within **VLsServerVendorInitialize()**. **VLsServerVendorInitialize()** is called when the server first starts to run. Here you inform the server of the name of the function which it can use to return the custom host ID by calling **VLssetHostIdFunc()**. Below is an example using a custom host ID function named “**getCustomHostID()**”. This code should be put into a separate “c” file.

```

extern long getCustomHostId();

LSERV_STATUS VLsServerVendorInitialize(void)
{
    VLssetHostIdFunc(&getCustomHostId);
    return(LSERV_STATUS_SUCCESS);
}

```

Registering the Custom Host ID Function on the Client

Here you need to call **VLssetHostIdFunc()** in the client application. in the same manner as was done in **VLsServerVendorInitialize()** above.

```

main(int argc, char ** argv){
    VLsInitialize( );
    VLssetHostIdFunc( );
    VLsRequest( );
}

```

Building the Server

Build the new customized **lserv** by linking it to files that contain code for **getCustomHostId()** and **VLServerVendorInitialize()** using *Custom32.mak*.

In this step the object files for the “c” files generated in the first two steps need to be linked with the server library.

Creating an Updated Client ID Generator

You will need to create an updated client ID generator (*echoid.exe*). The file, *myechoid.c*, takes the host ID from the **getCustomHostId()** function and prints it in hex. Sample code is shown below:

```
extern long getCustomHostId();

long main(int argc, char ** argv){
    long customid;
    customid=getCustomHostId();
    printf("0x%lX", customid);
}
```

Using a Customized Host ID

The sequence of events for an application using a custom ID is as follows:

1. Generate client node lock and/or server node locked licenses with the custom host ID as returned by *myechoid.exe*.
2. Rebuild and execute the customized **lserv**.
3. In the client application set the host ID function to **getCustomHostId()**.
Now the client side host ID has been changed.
4. Add the client node lock license to the server.

When an application tries to request a key for a client node-locked license, the server then verifies the client host ID as sent in the request message and compares it with the host ID in the license.

5. In the case of server locking to a customized host ID, when a server-locked license is added to the server, it executes the **VLServerVendorInitialize()** function and gets the host ID for the server then checks it against the host ID in the license.

Appendix C - Error and Result Codes for Client Functions

Client Function Return Codes

The following tables list LSAPI and SentinelLM client function return codes and their default actions:

Table C-1: LSAPI Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
0xC800100B	056	LS_BAD_INDEX	Bad index	Invalid index specified in LSEnumProviders() or any query functions.
0xC8001001	046	LS_BADHANDLE	Bad handle	Handle given to function represents an invalid licensing system context.
0xC800100E	059	LS_BUFFER_TOO_SMALL	Input buffer too small, string truncated.	Input buffer provided to function is not large enough to store the license server's name. Need to input a larger buffer.
0xC8001002	047	LS_INSUFFICIENTUNITS	Could not locate enough licensing resources.	Not enough sufficient resources to satisfy LSRequest() .
0xC800100D	058	LS_LICENSE_EXPIRED	Feature cannot run due to time restriction on it. Contact your software vendor.	Licensing agreement for this feature has expired.
0xC8001003	048	LS_LICENSESYSNOTAVAILABLE	Licensing System not available.	Licensing system itself is unavailable.
0xC8001004	049	LS_LICENSETERMINATED	License terminated because renewal time expired.	LSupdate() failed. License expired due to time-out.
0xC800100C	057	LS_NO_MORE_UNITS	No additional units are available.	Additional licenses/units requested are unavailable.

Table C-1: LSAPI Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
0xC80010009	034	LS_NO_MSG_TEXT	The specified filename can not be found on license server.	LSGetMessage() unable to retrieve message text.
0xC8001008	053	LS_NO_NETWORK	Unable to talk to the host specified. Verify client/server communication.	Network communication problems encountered.
0xC800100F	060	LS_NO_SUCCESS	No success in achieving the target.	No success in achieving the target.
0xC8001005	044	LS_NOAUTHORIZATIONAVAILABLE	Could not find the specified client for the feature.	License server does not recognize this feature name.
0xC8001006	051	LS_NOLICENSESAVAILABLE	All licensing keys are currently in use.	License server has no more license codes available for this request. All licenses are in use.
0xC8001007	047	LS_NORESOURCES	Could not locate enough licensing resources.	Insufficient resources (such as memory) are available to complete the request. An error occurred in attempting to allocate memory needed by function.
0x0	00	LS_SUCCESS		Successful completion of function call.
0xC800100A	055	LS_UNKNOWN_STATUS	Unknown error code, cannot provide error message.	Unknown or unrecognized status code was passed to LSGetMessage() .

Table C-2: SentinelLM Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
19	019	VLS_ADD_LIC_FAILED	Failed to add license string to the license server.	Dynamic license addition failed. Default: Display error message, return error code.
39	039	VLS_ALL_UNITS_RELEASED	All the keys issued to the feature have been returned.	The client asked VLSreleaseExt() API to return a specific number of units, it returned all the issued units.
42	042	VLS_AMBIGUOUS_HANDLE	The status of the handle is ambiguous.	The status of LS_HANDLE is ambiguous. It is not exclusively active or exclusively queued.
6	006	VLS_APP_NODE_LOCKED	Feature not licensed to run on this machine.	Server-locked feature cannot be issued a floating license code. Default: Display error message, return error code.
2	002	VLS_APP_UNNAMED	Feature name or version cannot be NULL.	No feature name provided with function call. Default: Display error message, return error code.
25	025	VLS_BAD_SERVER_MESSAGE	Could not understand message received from the license server. Verify Client and License server versions match.	An error has occurred in decrypting (or decoding) a network message at the client end. Probably an incompatible or unknown license server, or a version mismatch.
11	011	VLS_CALLING_ERROR	Error in calling the function. Check the calling parameters.	Error in calling a SentinelLM function. Default: Display error message, return error code.
45	045	VLS_CLIENT_NOT_AUTHORIZED	Client is not authorized for the specified action.	Client not authorized to make the specified request.

Table C-2: SentinelLM Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
26	026	VLS_CLK_TAMP_FOUND	Request denied due to clock tamper detection.	The license server has found evidence of tampering of the system clock, and it cannot service the request since the license for this feature has been set to be time-tamper proof.
20	020	VLS_DELETE_LIC_FAILED	Failed to delete feature from the license server.	Dynamic license deletion failed. Default: Display error message, return error code.
36	036	VLS_FINGERPRINT_MISMATCH	Machine's fingerprint mismatched.	The fingerprint identification of requesting computer does not match with the system.
3	003	VLS_HOST_UNKNOWN	Unknown license server host.	License server host does not seem to be on the network. Invalid host name specified. Default: Display error message, return error code.
12	012	VLS_INTERNAL_ERROR	Internal error in licensing or accessing feature.	SentinelLM internal error. Failure occurred in setting timer. (Timer is only attempted to be set if timer is available for platform and if license requires timer for updates.) Default: Display error message, return error code.
28	028	VLS_INVALID_DOMAIN	Cannot perform this operation on the domain name specified.	The domain of license server is different from that of client.
21	021	VLS_LOCAL_UPDATE	The last update was done locally.	The last update was done locally.
35	035	VLS_LOG_FILE_NAME_NOT_CHANGED	Cannot change specified log filename on license server.	Log file name was not changed.
34	034	VLS_LOG_FILE_NAME_NOT_FOUND	The specified log filename can not be found on license server.	Log file name not recognized by license server.

Table C-2: SentinelLM Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
24	024	VLS_MULTIPLE_VENDORID_FOUND	Feature licensed by multiple vendors.	The license system has licenses for the same <i>feature</i> , <i>version</i> , and it is not clear from the requested operation which license the requestor is interested in.
7	007	VLS_NO_KEY_TO_RETURN	Attempt to return a non-existent key for feature.	LSrelease() was called before the license code was issued. Default: Display error message.
1	001	VLS_NO_LICENSE_GIVEN	Unable to obtain licensing key.	Other internal error not listed above. Default: Display error message, return error code.
9	009	VLS_NO_MORE_CLIENTS	No more clients to report.	VLSgetClientInfo() has no more clients to report. Default: No action.
10	010	VLS_NO_MORE_FEATURES	No more features to report.	LSgetFeatureInfo() has no more features to report. Default: No action.
17	017	VLS_NO_RESPONSE_TO_BROADCAST	Probably no license servers running on this subnet.	No license servers responded to the VLSdiscover() call. Default: Display error message, return error code.
4	004	VLS_NO_SERVER_FILE	License server hostname not specified. Set environment variable LSHOST to name the license server.	Client not initialized with the name of the license server host. No license server has been set and unable to determine which license server to use. Default: Get the host name interactively from the user.
14	014	VLS_NO_SERVER_RESPONSE	License server not responding.	The license server is not responding due to communication has timed out. Default: Display error message, return error code.
5	005	VLS_NO_SERVER_RUNNING	Cannot talk to the license server. Verify license server is running.	No license server seems to be running on the remote host. License server on specified host is not available for processing the license operation requests. Default: Display error message, return error code.

Client Function Return Codes

Table C-2: SentinelLM Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
44	044	VLS_NO_SUCH_CLIENT	Could not find the specified client for the feature.	The client specified is not found on the license server.
18	018	VLS_NO_SUCH_FEATURE	No license string is available.	The license server does not recognize the given feature. Default: Display error message, return error code.
38	038	VLS_NO_UPDATES_SO_FAR	The updates for the specified feature have not been made so far.	No updates have been made so far.
43	043	VLS_NOMORE_QUEUE_RESOURCES	Could not locate enough resources to queue for license feature.	Could not queue the client because the queue is full.
27	027	VLS_NOT_AUTHORIZED	Unauthorized operation requested.	The specified operation is not permitted - authorization failed.
40	040	VLS_QUEUED_HANDLE	The specified handle is a queued handle.	The LS_HANDLE is a queued handle.
22	022	VLS_REMOTE_UPDATE	The last update was done remotely.	The last update was performed by contacting the SentinelLM license server.
8	008	VLS_RETURN_FAILED	Cannot return key for feature.	LSrelease() failed to return the issued license code. Default: Display error message, return error code.
13	013	VLS_SEVERE_INTERNAL_ERROR	Severe internal error in licensing or accessing feature.	SentinelLM severe internal error. An error occurred while attempting to retrieve system time. Default: Display error message, return error code.
37	037	VLS_TRIAL_LIC_EXHAUSTED	Duration or usage of a trial license is exhausted.	Trial license usage exhausted or trial license has expired.
16	016	VLS_UNKNOWN_SHARED_ID	Unknown shared id specified.	The supplied sharing criteria is unknown. Default: Display error message, return error code.

Client Function Return Codes

Table C-2: SentinelLM Client Function Return Codes

SentinelLM Error Number	Shell Error Number	Return Code	Default Message	Description
15	015	VLS_USER_EXCLUDED	User/machine excluded from running the given feature.	The user/computer is excluded by group reservations. Default: Display error message, return error code.
23	023	VLS_VENDORIDMISMATCH	Feature licensed by a different vendor.	The license system has those resources that could satisfy the request, but the vendor code of requested application does not match with that of the application licensed by the license server.

Appendix D - Error and Result Codes for License Generation Functions

License Generation Function Return Codes

The following table lists SentinelLM license generation function return codes and their default actions (where applicable):

Table D-1: License Generation Function Return Codes

Return Code	Description
VLScg_BAD_HANDLE	Bad file handle.
VLScg_DECRYPT_FAIL	Decryption failed for license string.
VLScg_DYNAMIC_DECRYPT_FAILURE	Decryption failed for dynamically added license string.
VLScg_EXCEEDS_MAX_STRLEN	Length of <value> is greater than <value>.
VLScg_EXCEEDS_MAX_VALUE	Value entered (<value>) exceeds the maximum allowed value. The maximum value can be <value>.
VLScg_EXPIRED_LICENSE	Your SW license file has expired.
VLScg_FAIL	Operation failed.
VLScg_FIXED_STR_ERROR	Default fixed string error.
VLScg_INTERNAL_ERROR	Internal error.
VLScg_INVALID_BIRTH_YEAR	Start year cannot be less than <value>.
VLScg_INVALID_CHARS	Invalid characters - \"<value>\".

Table D-1: License Generation Function Return Codes (Continued)

Return Code	Description
VLScg_INVALID_CHKSUM	Checksum validation failed for license string. Please verify the license string.
VLScg_INVALID_DATE	<value> is not valid in <value>, <value>.
VLScg_INVALID_DEATH_YEAR	Expiration year cannot be less than <value>.
VLScg_INVALID_EXP_DATE	Expiration Date must be greater than Start Date.
VLScg_INVALID_EXP_MONTH	License Expiration Month must be greater than Start Month.
VLScg_INVALID_EXP_YEAR	License Expiration Year must be greater than Start Year.
VLScg_INVALID_HANDLE	Invalid handle entered.
VLScg_INVALID_HEX_TYPE	Wrong value entered - \"<value>\". Should be hexadecimal.
VLScg_INVALID_INPUT	Invalid input - \"<value>\".
VLScg_INVALID_INT_TYPE	Expected an integer value, found \"<value>\".
VLScg_INVALID_IP_TYPE	Wrong value entered - \"<value>\". IP address should be specified in dot form.
VLScg_INVALID_LICTYPE	Invalid License Type.
VLScg_INVALID_RANGE	Value \"<value>\" violates the valid range of input.
VLScg_INVALID_TRIAL_COUNT	Invalid Trial License Count.
VLScg_INVALID_TRIALDAYS	Invalid Trial Days.
VLScg_INVALID_VENDOR_CODE	Invalid Vendor Code. Please contact your SentinelLM distributor.
VLScg_INVALID_YEAR	Invalid year entered - \"<value>\".
VLScg_LESS_THAN_MIN_VALUE	Value entered (<value>) is less than the minimum supported value. The minimum value is <value>.
VLScg_LICMETER_ACCESS_ERROR	Error accessing SentinelLM license meter(s). Please make sure the Sentinel System Driver is properly installed and a license meter is attached to the parallel port.

Table D-1: License Generation Function Return Codes (Continued)

Return Code	Description
VLScg_LICMETER_CORRUPT	Your SentinelLM license meter(s) are corrupted.
VLScg_LICMETER_COUNTER_TOOLOW	Too few units (Normal License Count=<value>/ Trial License Count=<value>) left in your SentinelLM license meter(s) to generate requested license. <value> units required.
VLScg_LICMETER_DECREMENT_OK	Your SentinelLM license meter(s) have been decremented by <value> units. You now have <value> units left out of an initial count of <value> units.
VLScg_LICMETER_EMPTY	All <value> units of your SentinelLM license meter(s) have been used up. License generation will fail.
VLScg_LICMETER_EXCEPTION	Unknown exception (<value>) in accessing SentinelLM license meter(s).
VLScg_LICMETER_VERSION_MISMATCH	Your SentinelLM license meter has an invalid version (<value>.<value>). Expected <value>.<value>.
VLScg_MALLOC_FAILURE	Out of heap memory.
VLScg_MAX_LIMIT_CROSSED	Maximum limit crossed.
VLScg_NO_ENABLE_FEATURE	Enable feature not specified.
VLScg_NO_FEATURE_NAME	Feature Name must be specified. It cannot be empty.
VLScg_NO_NETWORK_AUTHORIZATION	Server does not recognize this network.
VLScg_NO_RESOURCES	No resources left.
VLScg_NOT_MULTIPLE	Value of <value> should be a multiple of <value>.
VLScg_PORTSERV_ACCESS_ERROR	Error accessing SentinelLM license server(s) for a commuter license.
VLScg_PORTSERV_CORRUPT	Your SentinelLM license server(s) for commuter licensing is corrupted.

Table D-1: License Generation Function Return Codes (Continued)

Return Code	Description
VLScg_PORTSERV_EXCEPTION	Unknown exception (<value>) in accessing SentinelLM license server(s) for commuter licenses.
VLScg_PORTSERV_VERSION_MISMATCH	Your SentinelLM license server has an invalid version (<value>.<value>) for commuter licenses. Expected <value>.<value>.
VLScg_PREMATURE_TERM	Premature termination of license string. Please check.
VLScg_REMAP_DEFAULT	Failed to remap default strings from configuration file for license \"<value>\".
VLScg_RESERV_STR_ERR	\"<value>\" is a reserved string.
VLScg_SECRET_DECRYPT_FAILURE	Decryption failed for secrets. Verify the configuration file for readable licenses.
VLScg_SHORT_STRING	License string \"<value>\" too small to parse.
VLScg_SIMPLE_ERROR	Error in license string. Please check.
VLScg_SUCCESS	Successful completion of function call.
VLScg_TRIAL_SUCCESS	Your SentinelLM Trial license meter(s) have been decremented by <value> units. You now have <value> units left.
VLScg_TRIALMETER_EMPTY	All <value> units of your SentinelLM Trial license meter(s) have been used up.
VLScg_UNKNOWN_LOCK	Unknown lock mechanism - \"<value>\"
VLScg_VALUE_LARGE	Value \"<value>\" : too large.
VLScg_VENDOR_ENCRYPTION_FAIL	Vendor-customized encryption failed.

Appendix E - Error Codes for Redundancy, Queuing and Commuter Functions

Return Codes

The following tables list SentinelLM redundancy, queuing, and commuter return codes and their descriptions:

Table E-1: Redundancy, Queuing, and Commuter Return Codes

Return Code	Description
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	Buffer is not large enough to store license server's name.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for requested feature to grant a license.
LS_LICENSE_EXPIRED	License has expired.
LS_LICENSETERMINATED	Cannot update the license because the license has already expired.
LS_NO_AUTHORIZATION	License server does not recognize this feature name.

Table E-1: Redundancy, Queuing, and Commuter Return Codes

Return Code	Description
LS_NO_SUCCESS	Failed to retrieve computer names on local subnet.
LS_NO_SUCH_FEATURE	<i>feature_version</i> is non-existent.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_NON_REDUNDANT_SERVER_CONTACTED	Sets LSHOST to a non-redundant license server.
LS_NONNETWORK	Generic error indicating network failure.
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by this function.
VLS_ACTIVE_HANDLE	<i>lshandle</i> is an active handle.
VLS_ADD_LIC_FAILED	Generic error indicating the feature has not been added.
VLS_AMBIGUOUS_HANDLE	<i>lshandle</i> is an ambiguous handle; it is not exclusively active or exclusively queued.
VLS_APP_NODE_LOCKED	Requested feature is node locked, but request was issued from an unauthorized machine.
VLS_APP_UNNAMED	Specified feature is NULL.
VLS_BAD_DISTB_CRIT	Invalid distribution criteria.
VLS_BAD_HOSTNAME	<i>hostName</i> is not valid.
VLS_BAD_SERVER_MESSAGE	Message returned by license server could not be understood.
VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_CLIENT_NOT_AUTHORIZED	Client not authorized to remove queue.
VLS_CLK_TAMP_FOUND	License server has determined that the client's system clock has been modified. The license for this feature has time-tampering protection enabled, so the license operation is denied.
VLS_CONF_FILE_ERROR	Error in configuration file.

Table E-1: Redundancy, Queuing, and Commuter Return Codes

Return Code	Description
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.
VLS_FEATURE_INACTIVE	Feature is inactive on specified license server.
VLS_FINGERPRINT_MISMATCH	Client-locked; locking criteria does not match.
VLS_HOST_UNKNOWN	Invalid <i>hostName</i> was specified.
VLS_INVALID_DOMAIN	The domain of the license server is different from that of the client.
VLS_INVALID_IP_ADDRESS	<i>IP_address</i> is not valid.
VLS_LEADER_NOT_PRESENT	Unknown leader.
VLS_MAJORITY_RULE_FAILURE	Majority rule failure prevents token from being issued.
VLS_MULTIPLE_VENDORID_FOUND	The license server has licenses for the same feature and version from multiple vendors. It is ambiguous which feature is requested.
VLS_NO_LICENSE_GIVEN	Invalid handle specified.
VLS_NO_RESPONSE_TO_BROADCAST	No license servers have responded.
VLS_NO_SERVER_FILE	The license The license server has not been set and is unable to determine which license server to use.
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation requests.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches request feature.
VLS_NOMORE_QUEUE_RESOURCES	Queue is full.
VLS_NON_REDUNDANT_FEATURE	Feature is non-redundant and thus cannot be used in this redundancy-related operation.

Table E-1: Redundancy, Queuing, and Commuter Return Codes

Return Code	Description
VLS_NON_REDUNDANT_SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_NOT_AUTHORIZED	Invalid user.
VLS_ONLY_SERVER	Pool will not exist if this license server is removed.
VLS_POOL_FULL	Pool already has maximum number of license servers. No more license servers can be added.
VLS_QUEUED_HANDLE	<i>lshandle</i> is a queued handle.
VLS_SERVER_ALREADY_PRESENT	Attempted to add a license server that is already in the pool.
VLS_SERVER_NOT_PRESENT	Attempted to delete a license server that is not in the pool.
VLS_SERVER_SYNC_IN_PROGRESS	License server synchronization in process.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
VLS_UNRESOLVED_HOSTNAME	<i>IP_address</i> is valid, but could not be resolved.
VLS_UNRESOLVED_IP_ADDRESS	<i>IP_address</i> is valid, but could not be resolved.
VLS_USER_EXCLUDED	User or machine excluded from accessing requested feature.
VLS_VENDORIDMISMATCH	The vendor identification of the requesting application does not match the vendor identification of the feature for which the license server has the license.

Appendix F - Error and Result Codes for SentinelLM-Shell

SentinelLM-Shell Return Codes

The following table list SentinelLM-Shell return codes and their default actions:

Table F-1: SentinelLM-Shell Return Codes

Shell Error Number	Return Code	Default Message	Description
009	ACTIVATOR_ERROR	Could not launch the Client Activator.	Unknown error occurred when launching the Client Activator.
003	ERROR_LOADING	Error loading program.	The protected application was not loaded into memory prior to execution.
007	HEARTBEAT_FAIL	Heartbeat failed.	Failed to receive a response to a periodic query from the license server.
006	IMPORT_FUNC_ERROR	Error importing library function.	Error importing library function.
005	IMPORT_LIB_ERROR	Error loading import library.	Error loading import library.
001	INIT_ERROR	Initialization error.	A problem occurred when initializing the license manager library.
008	LIC_RENEWAL_FAIL	License renewal failed.	Failed to renew license or portable hardware key removed.
004	MEMORY_ACCESS_ERR	Memory access error.	Error accessing memory at run-time.
012	METER_ACCESS	Meter key access error.	Error accessing the key which meters the SentinelLM-Shell.
010	METER_INIT	License meter driver initialization error.	License meter key driver initialization error.
011	METER_VERSION	License meter version mismatch.	License meter key version mismatch.
002	NO_LICENSE	Could not get a license.	No license was found.

Appendix G - File Formats

This appendix contains the formats for the following files:

- License code
- Configuration
- Log
- Group reservation

The license server looks for these files under the directory specified by the environment variable, LSDEFAULTDIR. If this environment variable is not set, it looks in the directory where the executable resides.

License Code File Format

The license code file contains the encrypted license codes that provides the license server details of licensing agreements with software vendors. There is one license code for each feature licensed by the license server.

All SentinelLM utilities that read or write license codes use the following conventions:

- No more than one license code can be specified on one line of a file.
- A single license code cannot be split across lines.
- A license code must be terminated either by a new line or a pound sign (#).

- If a pound sign (#) is present on a line, all characters following it (until a new line) will be treated as a comment and ignored. Comments may appear anywhere in a license file.

Configuration File Format

A configuration file can be used for specifying alert actions as well as customizing the “fixed” or predefined strings found in a readable license string.

The fixed strings or keywords that can be remapped are:

SHORT	# code_type
LONG	
ADD	# additive
EXCL	
NO_SHR	# sharing_crit
USER_SHR	
HOST_SHR	
XDISP_SHR	
APP_SHR	
NO_HLD	# holding_crit
APP_HLD	
LIC_HLD	
FLOAT	# client_server_lock_mode
ND_LCK	
DEMO	
CL_ND_LCK	
_KEYS	# num_keys suffix
_MINS	# key_holdtime, key_lifetime suffix
#	# comment character
,	# subfield delimiter1
:	# subfield delimiter2

The strings above are used as the default strings to generate the readable license codes unless they are mapped to other strings and specified in the configuration file.

The format of the configuration file is as follows:

```
[feature_name1 feature_version1]
default_string = new_string # comments. This is a remap statement.
...

[feature_name2 feature_version2]
default_string = new_string # comments. This is another remap statement.
...
```

`[feature_name feature_version]` marks the beginning of a new section. All subsequent remap statements apply to readable licenses with this feature and version, until another `[feature_name feature_version]` section is encountered.

In the configuration file comments can be written after the pound sign/hash mark (#) character.

To remap the comment character and the two subfield delimiters used in a readable license, the following format must be used in the corresponding section of the map file:

Item	Description
COMMENT = \$	The comment character used in the readable license string is # now changed to '\$'.
SUBF_DELIM1 = ;	The subfield delimiter used in the readable license # string is ';' not ','.
SUBF_DELIM2 = /	The other subfield delimiter used in the readable license # string is '/' not ':'.

These characters are allowed to be remapped *just in case* you wish to use one or more of these characters in your license code generator data (e.g., in vendor info), which could interfere with parsing of the subfields of a readable license. This remapping should be done when you run the license code generator. Perform the following steps:

1. Write the configuration file.
2. Make sure the license code generator finds the configuration file, and that the appropriate feature and version section exists.
3. The license code generator will generate the remapped license string.
4. Ship the configuration file as well as the readable license to the end user.

5. The end user should make sure that **lsdecode** and/or the license server able to read the configuration file. If either of these are not able to read the configuration file, the license string may not be parsed correctly.

Steps 3 and 5 apply to any remap statement, whether it is the comment character or LONG that is being remapped.

In the configuration file the *feature_name* and *feature_version* can be specified in the following three formats to control the range of applicability of the section:

1. [*feature_name feature_version*] ==>

Subsequent remap statements apply only to *feature_name* and *feature_version*.

For example:

[DOTS 1.0] ==> remapping for version 1.0 of DOTS.

2. [*feature_name* *] ==> remapping for all versions of *feature_name*.

For example:

[DOTS *] ==> remapping for all versions of DOTS.

3. [] or [* *] ==> remapping for all license codes in the license file.

If a particular feature name and version corresponds to more than one [*feature_name feature_version*] section, then the section which describes the feature most accurately is selected and the remap statements under that section are used for remapping.

For example:

If [], [DOTS 2], and [DOTS *] are all specified in the map file, then:

- For DOTS version 2 statements specified below [DOTS 2] will be used.
- For DOTS version 1.0 statements specified below [DOTS *] will be used.
- For TUTOR version 0 statements specified below [] will be used.

[] or [] are invalid and should be written as [] (no space between the two square brackets).

[**] is invalid and should be written as [* *] in the configuration file.

Furthermore, for statements associated with a particular feature and version, *only* the statements within the applicable section will be used. If some statements are missing from [DOTS *] but are given in [* *], the ones in [* *] will *not* be used for DOTS 1.0.

An example configuration file is shown below:

```

[]                                # all features
SHORT = SH                        # short code
COMMENT = #                       # comment char remains the same
LONG = Ln
_KEYS = _keys
_MINS = _minutes
[DOTS *]                          # mapping for all versions of DOTS
SHORT = short
_KEYS = _number_of_keys
LONG = long_code
_MINS = _minutes
[DOTS 1]                          # mapping for version 1 of DOTS
SHORT=SHORT_CODE
LONG = LONG_CODE
FLOAT = FLOATING
_KEYS=_NUM_LICENSES
SUBF_DELIM1= ;                    # comma remapped to a semi-colon
[STARS 2]                         # stars version 2
_MINS = _MINUTES
LONG=LONG_CODE
SHORT=SHORT_CODE
_KEYS=_LICENSE
SUBF_DELIM2 = / # colon remapped to '/'
COMMENT = @ # comment delimiter

```

For parsing errors in readable license strings, the license server gives the line number of the string, the file name, and the cause of error.

The environment variable, LSERVRCNF, can specify the path to the configuration file. The path for <licenseFile>.cnf, is constructed from the license file path the user is using. *licenseFile* can be specified using existing methods such as the -s option, or the LSERVRC environment variable. It is not

an error for the configuration file to be missing. The configuration file can contain information other than remap statements. For instance, alert specifications are also given in this file, so it is a general-purpose configuration file associated with a particular license file.

Log File Format

The license server generates a usage file that logs all license codes issued or denied. License code updates are not recorded. Usage reports can be generated using the SentinelLM utility, **lsusage**. Reports for encrypted log files can be generated by developers only using the **vusage** utility. See the *SentinelLM Developer's Guide* for information on **lsusage** and **vusage**.

Various levels of encryption can be set for the log file entries. You set the encryption level for a particular license code when you generate it, and any log file entry created for that license code will be encrypted at that level. A developer-specified non-zero encryption level overrides any encryption level set by a customer. See the *SentinelLM Developer's Guide* and the *SentinelLM Administrator's Guide* for details.

License codes with an LFE level of 0 will be encrypted using the level specified in the **-lfe** license server switch.

Information is recorded in the log file one entry per line in the following format:

Table G-1: Log Entry Format

Server- LFE	License- LFE	Date	Time- stamp	Feature	Ver	Trans	Numkeys	Keylife	User	Host	LSver	Currency	Comment
----------------	-----------------	------	----------------	---------	-----	-------	---------	---------	------	------	-------	----------	---------

Table G-2: Elements of a Log File

Element	Description
Server-LFE	Customer-defined log file encryption level as specified by the license server -lfe startup option.
License-LFE	Developer-defined log file encryption level as specified during license code generation. If this is non-zero, it overrides the Server-LFE.

Table G-2: Elements of a Log File (Continued)

Element	Description
Date	The date the entry was made, in the format: Day-of-week Month Day Time (hh:mm:ss) Year
Time-stamp	The time stamp of the entry, according to the format set by the mktime() C library call.
Feature	Name of the feature.
Ver	Version of the feature.
Trans	The transaction type. 0 indicates an issue, 1 a denial, and 2 a return.
Numkeys	The number of licenses in use after the current request/release.
Keylife	The time, in seconds, that the license was issued.
User	The user name of the application associated with the entry.
Host	The host name of the application associated with the entry.
LSver	The version of the SentinelLM license server.
Currency	The number of licenses handled during the transaction.
Comment	The text associated with the <i>log_comment</i> string passed in by LSRequest() or LSRelease() .

A typical entry might appear as:

```
3 3 xxxx Tue Jul 06 11:46:27 1999 931286887 99 v1 2 MQ == 632 jsmith engr1 7.00 1 ----- xx xxxx
xxxxxx
```

This entry indicates that *Tuesday, July 06, 1999, at 11:46:27*, the user, *jsmith* finished using an application with the feature *99* and version *1*. The license was returned after using the application for *632* seconds on computer *engr1*. Because this is encrypted to level 3, the number of license tokens remaining after the license was returned is encrypted. The license server version is *7.00*, and *1* license token was used by the application.

If the maximum size of the log file has been specified using the **-z** option, SentinelLM automatically trims the log file so that it will not grow indefinitely. The trimming mechanism ensures that the log file always will have less than 2,000 lines of ASCII text (each line requiring less than 100 bytes).

Index

A

- adding
 - APIs 9
 - feature licensing information 104, 106
 - security 266–268, 269, 272
- advanced client functions 41
- Advanced-API 2
- APIs
 - adding 9
 - advanced 2
 - client 23–119
 - client example 3
 - commuter 249
 - license code generation 121
 - queuing 231
 - quick 1
 - redundancy 201
 - standard 1
- applications
 - sample 259–260
- authenticating the license manager 45–48

B

- basic client licensing functions 25–30
- basic license code generation functions 126
- broadcast intervals
 - retrieving 67
 - setting 66

C

- CHALLENGE structure, defined 46
- challenge-response mechanism 45–48
- CHALLENGERESPONSE structure,
 - defined 46
- CHANGE_PORT_OBJ Makefile
 - variable 276
- changing

- port number default 276–277
- system time 266–268
- client API 23–119
 - example 3
- client configuration functions 2, 52
- client feature information, retrieving 81, 83
- client function return codes 283
- client libraries 24
- client library
 - initializing 31
 - retrieving information 109
 - tracing calls 119
- client query functions 2, 79
- client utility functions 2, 100–112
- clock, detecting changes 17, 266–268
- code struct field setting functions 132–151
- codeT 121
- Commuter Licensing 249
- commuter licensing 249
- configuration files
 - format of 302–306
- contacting Rainbow xix
- conventions
 - syntax xviii
 - typographic xvii
- custom host IDs, creating 277–281
- customizing functions 261
- customizing SentinelLM
 - changing port numbers 276–277
 - creating a custom host ID 277–281
 - detecting time tampering 266–268
 - error handling 116
 - license code encryption 269–272
 - message encryption 272–275

D

- DECRYPT_LIC_OBJ Makefile variable 270

DECRYPT_MSG_OBJ Makefile
variable 273

decrypting
license codes 269–272
messages 272–275
deleting
feature licensing information 107
destroying the handle for lscgen.h 127
disable auto timer 78
displaying error messages 115, 117

E

ENCRYPT_LIC_OBJ Makefile variable 270
ENCRYPT_MSG_OBJ Makefile
variable 273

encrypting
license codes 269–272
messages 272–275
environment variables
LS_MAX_GRP_QLEN 234
LS_MAX_HOLD_SEC 235
LS_MAX_QLEN 234
LS_MAX_WAIT_SEC 235
LSDEFAULTDIR 301
LSERVRC 305
LSERVRCNF 305
LSFORCEHOST 11
LSHOST 15, 53
error codes
client functions 283
license generation functions 291
redundancy, queuing and commuter
functions 295
SentinelLM-Shell functions 299
error handling 113–118
customizing 116
setting 116
error handling functions 2
error message display 117

error messages, displaying 115
errors, retrieving 128–131
event handlers, registering with the
server 263
example files 264

F

feature licensing information
adding 104, 106
deleting 107
retrieving 91
feature names, retrieving 94
feature query functions 2, 86–100
feature time left information
retrieving 97
FeatureName parameter 3, 10
file formats 301–307
configuration 302–306
license codes 301
log 306–307
files
lservrc 301
lshost 53
functions
basic client 25–30
client configuration 2, 52
client query 2, 79
client utility 2, 100–112
customizing 261
error handling 2
feature query 2, 86–100
redundancy 201

H

help
getting xviii
hold time
setting 69
host ID

- customizing 277–281
- setting 66

host names

- retrieving 56
- setting 53

I

- initializing fields of the machineID 58
- initializing the client library 31
- initializing the server 263
- initializing the server info 65

K

key time left information

- retrieving 99

keys

- renewing 38

L

libraries

- client 24
- integrated 24
- network 24
- stand-alone 24
- UNIX 15

license code generation API 121

license codes

- encrypting and decrypting 269–272
- file format 301

license generation function return codes 291

license manager

- authenticating 45–48
- usage logging 306–307

license server

- APIs
 - license code generation 121
- locating 101

LICENSE_LIBS macro 16

licenses

- lifetime of 39
- local vs. remote renewal of 74
- releasing 36, 48
- renewing 38
- requesting 32, 42
- single-call licensing 26
 - disabling 29
- lifetime of a license 39
- local license renewal 75
- locating the license server 101
- log file format 306–307
- LS_LIBVERSION structure, defined 110
- LS_MAX_QLEN 234
- LSAPI client function return codes 284
- lscgen.h handle
 - destroying 127
- LSDEFAULTDIR environment variable 301
- LSERVRC environment variable 305
- lservrc file 301
- LSFORCEHOST environment variable 11
- LSGetMessage 115
- LSHOST environment variable 15, 53
- lshost file 53
- LSRelease 36
- LSRequest 32
- LSUpdate 38
- lsusage utility 306

M

machine names, retrieving 101

macros

- LICENSE_LIBS 16
- NO_LICENSE 6

Makefile 16, 259, 262

Makefile variables

- CHANGE_PORT_OBJ 276
- DECRYPT_LIC_OBJ 270
- DECRYPT_MSG_OBJ 273
- ENCRYPT_LIC_OBJ 270

ENCRYPT_MSG_OBJ 273
messages, encrypting and decrypting 272–
275

P

port numbers
 changing the default 276–277
 retrieving 58
printing errors 128–131
problems
 reporting xxi
programs, sample 259–260
PublisherName parameter 3, 10

Q

quick client functions 25
Quick-API 1

R

redundancy 295
redundant license server 201
registering an event handler 263
releasing licenses 36, 48
remote renewal period 39
remote renewal time, setting 77
renewing license keys 38
reporting problems xxi
requesting licenses 32, 42
retrieving
 broadcast intervals 67
 client feature information 81, 83
 client library information 109
 errors 128–131
 feature licensing information 91
 feature names 94
 feature time left information 97
 license time left information 99
 machine names 101
 server host names 56

server port numbers 58
time drift information 96
time-out intervals 68
version information 93, 95
Returns 37

S

sample applications 259–260
sample programs 259
samples.mak 259
security
 adding 16, 266–268, 269, 272
SentinelLM
 APIs
 client 23–119
 license code generation 121
 architecture 1–2
 customizing
 changing port numbers 276–277
 creating a custom host ID 277–281
 detecting time tampering 266–268
 error handling 116
 license code encryption 269–272
 message encryption 272–275
 license generation function return
 codes 291
 security 16
 adding 266–268, 269, 272
servers
 detecting 10
 initializing 263
 retrieving host names 56
 retrieving port numbers 58
 setting
 host names 53
setting
 broadcast intervals 66
 code struct fields 132–151
 error handling 116

- hold time 69
- host ID 66
- remote renewal time 77
- server names 53
- time-out intervals 68
- shared IDs 70, 72
- shutting down lserv 110
- single-call licensing 26
 - disabling 29
- standard client functions 31
- Standard-API 1
- structure definitions
 - CHALLENGE 46
 - CHALLENGERESPONSE 46
 - LS_LIBVERSION 110
 - VLScientInfo 80
 - VLSfeatureInfo 87
- syntax conventions xviii
- system time, detecting changes 17, 266–268

T

- time clock, detecting changes 17, 266–268
- time drift information
 - retrieving 96
- time-out intervals
 - retrieving 68
 - setting 68
- tracing client-library calls 119
- tracing SentinelLM operation 118
- typographic conventions xvii

U

- UNIX
 - libraries 15
 - Makefile 16, 259
- updating 49
- updating licenses 49
- usage logging 306–307
- using the SentinelLM client API 23

- utilities
 - lsusage 306

V

- variable 305
- variables
 - environment
 - LSDEFAULTDIR 301
 - LSERVRC 305
 - LSFORCEHOST 11
 - LSHOST 15, 53
 - Makefile
 - CHANGE_PORT_OBJ 276
 - DECRYPT_LIC_OBJ 270
 - DECRYPT_MSG_OBJ 273
 - ENCRYPT_LIC_OBJ 270
 - ENCRYPT_MSG_OBJ 273
- version information
 - retrieving 93, 95
- Version parameter 3, 10
- VLSaddFeature 104, 203
- VLSaddFeatureExt 205
- VLSaddFeatureToFile 106, 206
- VLSaddServerToPool 208
- VLSbatchUpdate 49
- VLScgAllowAdditive 135
- VLScgAllowClientLockInfo 172
- VLScgAllowClockTamperFlag 148
- VLScgAllowCodegenVersion 153
- VLScgAllowCommuterLicense 157
- VLScgAllowFeatureName 166
- VLScgAllowFeatureVersion 167
- VLScgAllowHeldLic 139
- VLScgAllowKeyHoldtime 186
- VLScgAllowKeyHoldUnits 183
- VLScgAllowKeyLifetime 185
- VLScgAllowKeyLifeUnits 182
- VLScgAllowKeysPerNode 174
- VLScgAllowLicBirth 187

VLScgAllowLicenseType	151	VLScgSetCodeLength	136
VLScgAllowLicExpiration	190	VLScgSetCommuterLicense	157
VLScgAllowLockMechanism	145	VLScgSetFeatureName	166
VLScgAllowLockModeQuery	169	VLScgSetFeatureVersion	168
VLScgAllowLogEncryptLevel	159	VLScgSetHoldingCrit	139
VLScgAllowMajorityRuleFlag	155	VLScgSetKeyHoldtime	186
VLScgAllowMultiKey	160	VLScgSetKeyHoldtimeUnits	183
VLScgAllowMultipleServerInfo	162	VLScgSetKeyLifetime	185
VLScgAllowNetworkFlag	141	VLScgSetKeyLifetimeUnits	182
VLScgAllowNumFeatures	177	VLScgSetKeysPerNode	174
VLScgAllowNumKeys	179	VLScgSetKeyType	160
VLScgAllowOutLicType	150	VLScgSetLicBirthDay	189
VLScgAllowRedundantFlag	154	VLScgSetLicBirthMonth	188
VLScgAllowSecrets	162	VLScgSetLicBirthYear	189
VLScgAllowServerLockInfo	170	VLScgSetLicenseType	152
VLScgAllowSharedLic	142	VLScgSetLicExpirationDay	191
VLScgAllowShareLimit	193	VLScgSetLicExpirationMonth	191
VLScgAllowSiteLic	175	VLScgSetLicExpirationYear	192
VLScgAllowSoftLimit	180	VLScgSetLicType	138
VLScgAllowStandAloneFlag	140	VLScgSetLoadSWLicFile	195
VLScgAllowTrialLicFeature	144	VLScgSetLogEncryptLevel	159
VLScgAllowVendorInfo	165	VLScgSetMajorityRuleFlag	156
VLScgCleanup	127	VLScgSetNumClients	178
VLScgDecodeLicense	197	VLScgSetNumericType	194
VLScgGenerateLicense	196	VLScgSetNumFeatures	177
VLScgGetErrorLength	129	VLScgSetNumKeys	179
VLScgGetErrorMessage	130	VLScgSetNumSecrets	164
VLScgGetLicenseMeterUnits	198	VLScgSetNumSubnets	176
VLScgGetNumErrors	129	VLScgSetOutLicType	150
VLScgGetTrialLicenseMeterUnits	199	VLScgSetRedundantFlag	154
VLScgInitialize	126	VLScgSetSecrets	163
VLScgPrintError	131	VLScgSetServerLockInfo1	170
VLScgReset	128	VLScgSetServerLockInfo2	171
VLScgSetAdditive	136	VLScgSetServerLockMechanism1	146
VLScgSetClientLockInfo	173	VLScgSetServerLockMechanism2	147
VLScgSetClientLockMechanism	145	VLScgSetSharedLicType	142
VLScgSetClientServerLockMode	169	VLScgSetShareLimit	193
VLScgSetClockTamperFlag	148	VLScgSetSiteLicInfo	175
VLScgSetCodeGenVersion	153	VLScgSetSoftLimit	181

VLScgSetStandAloneFlag 141	VLSgetHostAddress 223
VLScgSetTrialDaysCount 144	VLSgetHostName 220
VLScgSetVendorInfo 165	VLSgetKeyTimeLeftFromHandle 99
VLSchangeDistbCrit 209	VLSgetLeaderServerName 221
VLSchangePortNumber 276	VLSgetLibInfo 109
VLSchangeUsageLogFileName 253	VLSgetLicInUseFromHandle 85
VLSCleanup 37	VLSgetLicSharingServerList 224
VLScleanup 49	VLSgetMachineID 60
VLSclientInfo 80	VLSgetQueuedClientInfo 237
VLSconfigureTimeTamper 266, 267	VLSgetQueuedLicense 245
VLSdecryptLicense 271	VLSgetServerList 64
VLSdecryptMsg 274	VLSgetServerNameFromHandle 62
VLSdeleteFeature 107	VLSgetServerPort 58, 276
VLSdelServerFromPool 210	VLSgetTimeDriftFromHandle 96
VLSdisableAutoTimer 78	VLSgetTimeoutInterval 68
VLSdisableEvents 256	VLSgetTrialPeriodLeft 200
VLSdisableLicense 29	VLSgetUsageLogFileName 254
VLSdiscover 11, 101	VLSgetVersionFromHandle 95
VLSdiscoverExt 212	VLSgetVersions 93
VLSenableLocalRenewal 75	VLSinitialize 31
VLSencryptLicense 269	VLSinitMachineID 58
VLSencryptMsg 273	VLSinitQueuePreference 247
VLSerrorHandle 114	VLSinitServerInfo 65
VLSeventAddHook 263	VLSinitServerList 63
VLSeventSleep 256	VLSisClockSetBack 268
VLSfeatureInfo 87	VLSisLocalRenewalDisabled 75
VLSgetAndInstallCommuterCode 250	VLSlicense 26
VLSgetBroadcastInterval 67	VLSmachineIDtoLockCode 61
VLSgetClientInfo 82	VLSqueuedRequest 232
VLSgetCommuterInfo 249	VLSqueuedRequestExt 232
VLSgetContactServer 56	VLSreleaseExt 48
VLSgetDistbCrit 215	VLSremoveQueue 240
VLSgetDistbCritToFile 217	VLSremoveQueuedClient 238
VLSgetFeatureFromHandle 94	VLSrequestExt 42
VLSgetFeatureInfo 91	VLSscheduleEvent 255
VLSgetFeatureInfoToFile 219	VLSserverVendorInitialize 263
VLSgetFeatureTimeLeftFromHandle 97	VLSsetBroadcastInterval 66
VLSgetHandleInfo 84	VLSsetContactServer 53
VLSgetHandleStatus 241	VLSsetErrorHandler 116

VLSsetHoldTime 13, 69
VLSsetRemoteRenewalTime 77
VLSsetServerPort 276
VLSsetSharedId 70
VLSsetSharedIdValue 72
VLSsetTimeoutInterval 68
VLSsetTraceLevel 119
VLSsetUserErrorFile 117
VLSshutDown 110
VLSuninstallAndReturnCommuterCode 252
VLSupdateQueuedClient 242
VLSwhere 112