

Sentinel LM Programmer's Reference Manual



Copyright © 2005, SafeNet, Inc. (Baltimore)
All rights reserved.

All attempts have been made to make the information in this document complete and accurate. SafeNet, 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.

SafeNet, Sentinel, and Sentinel LM are either trademarks or registered trademarks of Safenet, Inc. All other product names referenced herein are trademarks or registered trademarks of their respective manufacturers.

CONFIDENTIAL INFORMATION

Sentinel LM protection system is designed to protect your applications 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.

Part Number 001640-001, Revision D
Software versions 8.0 and later

Revision	Action/Change	Date
A	Initial Release	December 2004
B	Updated for UNIX releases	April 2005
C	Updated for Solaris release	June 2005
D	Update for this release.	July 2005

SafeNet Sales Offices

Australia +61 3 9882 8322	Brazil +55 11 6121 6455	China +86 10 88519191	Finland +358 20 500 7800
France +33 1 41 43 29 00	Germany +49 1803 7246269	Hong Kong +852 3157 7111	India +91 11 26917538
Japan (Tokyo) + 81 45 6405733	Korea +82 31 705 8212	Mexico +52 55 5575 1441	Netherlands +31 73 658 1900
Singapore +65 6297 6196	Taiwan 886-2-27353736	UK (Camberley) +44 0 1276 608000	U.S. (Massachusetts) +1 978.539.4800
U.S. (New Jersey) +1 201.333.3400	U.S. (Virginia) +1 703.279.4500	U.S. (Irvine, California) +1 949.450.7300	U.S. (San Jose, California) + (408) 452 7651
U.S. (Torrance, California) +1 310.533.8100			

International Quality Standard Certification



The Beijing, China; Irvine, California, U.S.A; and Rotterdam, The Netherlands facilities are certified to the latest, globally-recognized ISO 9001:2000 standard. The certificate number is: CERT-02982-2003-AQ-HOU-RAB Rev 3.

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.

FCC Notice to Users



The hardware key has passed the FCC Self-authorization process of Computers and Computer Peripherals. FCC Part 15 Class B Specifications.

This equipment has been tested and found to comply with the limits for a class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

In order to maintain compliance with FCC regulations, shielded cables must be used with this equipment. Operation with non-approved equipment or unshielded cables is likely to result in interference to radio and TV reception. The user is cautioned that changes and modifications made to the equipment without the approval of manufacturer could void the user's authority to operate this equipment.

Contents

Preface xix

Sentinel LM Documentation Resources	xix
<i>Who Should Read This Manual?</i>	xx
Conventions Used in This Manual.....	xxi
About This Manual	xxii
Contacting Technical Support.....	xxiii
Export Considerations	xxv
We Welcome Your Comments	xxvi

Chapter 1 – Introduction 1

Using the Sentinel LM Application Library	1
Licensing on Stand-alone and Network Computers	3
Client API Example.....	3
<i>Example</i>	4
Language Interfaces Supported.....	5
Special Use of Win32 for Generating Tools	5
Debugging Your Client Application	5
Disabling Licensing	6

Chapter 2 – Protecting Your Application with the Application Library 7

Stand-alone Application Protection	8
Network Application Protection	8
<i>Adding APIs to Your Source Code</i>	8
<i>Application Identification</i>	9

<i>Automatic License Server Detection</i>	10
<i>Special Licensing Cases</i>	12
Integrated Application Protection.....	12
<i>Dynamic Switching Between Stand-alone and Network Licensing</i>	13
Examples of Dynamic Switching.....	14
Linking with the Correct Library.....	14
<i>Windows Static Linked Libraries</i>	14
<i>Windows Dynamic Linked Libraries and Import Libraries</i>	16
<i>UNIX Libraries</i>	16
Testing and Debugging Your Application.....	17
<i>Disabling Licensing</i>	17
<i>Library Tracing</i>	17
Sample Programs	18
<i>Sample Program Summary</i>	18
<i>Customization Samples</i>	19
Notes on Security	19
<i>Protecting Against Time Tampering</i>	20

Chapter 3 – Sentinel LM Client API..... 21

Basic Client Licensing Functions	24
<i>Quick Client Licensing Functions</i>	24
VLSlicense	24
VLSdisableLicense	28
<i>Standard Client Licensing Functions</i>	29
VLSinitialize	29
LSRequest.....	31
LSUpdate	35
LSRelease	39
VLScleanup	41
<i>Advanced Client Licensing Functions</i>	41
VLSrequestExt	42
VLSrequestExt2.....	45
VLSreleaseExt.....	45
VLSbatchUpdate	47
Challenge-response Mechanism	50
Client Configuration Functions	52

VLSsetContactServer	53
VLSgetContactServer	56
VLSsetServerPort	57
VLSgetServerPort	58
VLSinitMachineID	58
VLSgetMachineID	60
VLSgetNumberedMachineID	61
VLSmachineIDtoLockCode	62
VLSgetServerNameFromHandle	63
VLSinitServerList	64
VLSgetServerList	65
VLSinitServerInfo	66
VLSsetHostIdFunc	66
VLSsetBroadcastInterval	67
VLSgetBroadcastInterval	68
VLSsetTimeoutInterval	68
VLSgetTimeoutInterval	69
VLSsetHoldTime	70
VLSsetSharedId/ VLSsetTeamId	71
VLSsetSharedIdValue/ VLSsetTeamIdValue	73
Local vs. Remote Renewal of License Tokens	75
VLSdisableLocalRenewal	76
VLSenableLocalRenewal	77
VLSisLocalRenewalDisabled	77
VLSgetRenewalStatus	78
VLSsetRemoteRenewalTime	81
VLSdisableAutoTimer	82
Client Query Functions	83
VLSgetClientInfo	86
VLSgetHandleInfo	89
VLSgetLicInUseFromHandle	90
Feature Query Functions	91
VLSgetFeatureInfo	100
VLSgetVersions	103
VLSgetFeatureFromHandle	104
VLSgetVersionFromHandle	105

VLSgetTimeDriftFromHandle	107
VLSgetFeatureTimeLeftFromHandle	108
VLSgetKeyTimeLeftFromHandle	110
Client Utility Functions.....	111
VLSdiscover.....	112
VLSaddFeature.....	115
VLSaddFeatureToFile.....	117
VLSdeleteFeature	119
VLSgetLibInfo.....	121
VLSshutDown	122
VLSwhere.....	124
Trial License Related Functions.....	125
VLSgetTrialPeriodLeft	126
Getting License Server Information.....	127
VLSservInfo Struct.....	128
Retrieving Information About Time Tampering -VLStimeTamperInfo Struct	129
Retrieving Information About a License Server (VLSgetServInfo)	130
VLSservInfo Data Structure	131
License Revocation Function	132
VLSrevokeLicense.....	132
Error Handling	136
VLSerrorHandle	137
LSGetMessage	137
VLSsetErrorHandler	138
VLSsetUserErrorFile	140
Tracing Sentinel LM Operation	141
Chapter 4 – License Code Generation API	143
License Code Generation Functions	144
CodeT Struct.....	148
Basic Functions	156
VLScgInitialize.....	157
VLScgCleanup.....	157
VLScgReset.....	158
Functions Which Retrieve or Print Errors.....	158
VLScgGetNumErrors	159

VLScgGetErrorLength.....	160
VLScgGetErrorMessage	160
VLScgPrintError.....	162
Functions for Setting the Fields in CodeT Struct	162
VLScgSetCodeLength.....	167
VLScgAllowFeatureName	168
VLScgSetFeatureName.....	169
VLScgAllowFeatureVersion.....	170
VLScgSetFeatureVersion.....	170
VLScgAllowLicenseType	171
VLScgSetLicenseType	171
VLScgAllowTrialLicFeature.....	172
VLScgSetTrialDaysCount	173
VLScgAllowAdditive.....	173
VLScgSetAdditive.....	174
VLScgAllowKeyLifetime.....	175
VLScgSetKeyLifetime	175
VLScgAllowStandAloneFlag	176
VLScgAllowNetworkFlag	177
VLScgAllowPerpetualFlag.....	177
VLScgSetStandAloneFlag	178
VLScgAllowLogEncryptLevel.....	179
VLScgSetLogEncryptLevel.....	179
VLScgAllowSharedLic/ VLScgAllowTeamCriteria.....	180
VLScgSetSharedLicType/ VLScgSetTeamCriteria.....	181
VLScgAllowShareLimit/ VLScgAllowTeamLimit	182
VLScgSetShareLimit/ VLScgSetTeamLimit	183
VLScgAllowCommuterLicense	184
VLScgSetCommuterLicense	185
VLScgAllowCommuterMaxCheckoutDays	186
VLScgAllowNumKeys.....	187
VLScgSetNumKeys	187
VLScgAllowLockModeQuery	188
VLScgSetClientServerLockMode	189
VLScgAllowRedundantFlag.....	190
VLScgSetRedundantFlag	190

<i>VLScgAllowMajorityRuleFlag</i>	191
<i>VLScgSetMajorityRuleFlag</i>	191
<i>VLScgAllowMultipleServerInfo</i>	193
<i>VLScgSetNumServers</i>	193
<i>VLScgAllowServerLockInfo</i>	194
<i>VLScgSetServerLockInfo1</i>	195
<i>VLScgSetServerLockMechanism1</i>	196
<i>VLScgSetServerLockMechanism2</i>	197
<i>VLScgSetServerLockInfo2</i>	198
<i>VLScgAllowLockMechanism</i>	199
<i>VLScgSetClientLockMechanism</i>	200
<i>VLScgAllowClientLockInfo</i>	201
<i>VLScgSetClientLockInfo</i>	201
<i>VLScgSetNumClients</i>	202
<i>VLScgAllowClockTamperFlag</i>	203
<i>VLScgSetClockTamperFlag</i>	204
<i>VLScgAllowOutLicType</i>	205
<i>VLScgSetOutLicType</i>	205
<i>VLScgSetLicType</i>	207
<i>VLScgAllowHeldLic</i>	208
<i>VLScgSetHoldingCrit</i>	208
<i>VLScgAllowCodegenVersion</i>	209
<i>VLScgSetCodegenVersion</i>	210
<i>VLScgAllowCapacityLic</i>	210
<i>VLScgSetCapacityFlag</i>	211
<i>VLScgAllowCapacity</i>	212
<i>VLScgSetCapacityUnits</i>	213
<i>VLScgSetCapacity</i>	215
<i>VLScgAllowMultiKey</i>	216
<i>VLScgSetKeyType</i>	216
<i>VLScgAllowSecrets</i>	217
<i>VLScgSetSecrets</i>	218
<i>VLScgSetNumSecrets</i>	219
<i>VLScgAllowVendorInfo</i>	220
<i>VLScgSetVendorInfo</i>	220
<i>VLScgAllowKeysPerNode</i>	221

<i>VLScgSetKeysPerNode</i>	222
<i>VLScgAllowSiteLic</i>	223
<i>VLScgSetSiteLicInfo</i>	223
<i>VLScgSetNumSubnets</i>	224
<i>VLScgAllowNumFeatures</i>	225
<i>VLScgSetNumFeatures</i>	226
<i>VLScgAllowSoftLimit</i>	227
<i>VLScgSetSoftLimit</i>	227
<i>VLScgAllowKeyLifeUnits</i>	228
<i>VLScgSetKeyLifetimeUnits</i>	229
<i>VLScgAllowKeyHoldUnits</i>	230
<i>VLScgSetKeyHoldtimeUnits</i>	230
<i>VLScgAllowKeyHoldtime</i>	231
<i>VLScgSetKeyHoldtime</i>	232
<i>VLScgAllowLicBirth</i>	233
<i>VLScgSetLicBirthMonth</i>	233
<i>VLScgSetLicBirthDay</i>	234
<i>VLScgSetLicBirthYear</i>	235
<i>VLScgAllowLicExpiration</i>	236
<i>VLScgSetLicExpirationMonth</i>	237
<i>VLScgSetLicExpirationDay</i>	238
<i>VLScgSetLicExpirationYear</i>	239
<i>VLScgSetNumericType</i>	240
<i>VLScgSetLoadSWLicFile</i>	241
<i>VLScgAllowGracePeriodFlag</i>	241
<i>VLScgSetGracePeriodFlag</i>	242
<i>VLScgAllowGracePeriod</i>	242
<i>VLScgSetGracePeriodDays</i>	243
<i>VLScgSetGracePeriodHours</i>	243
<i>VLScgAllowOverdraftFlag</i>	244
<i>VLScgSetOverdraftFlag</i>	244
<i>VLScgAllowOverdraft</i>	245
<i>VLScgSetOverdraftHours</i>	245
<i>VLScgSetOverdraftUsers</i>	246
<i>VLScgAllowLocalRequestLockCritFlag</i>	246
<i>VLScgSetLocalRequestLockCritFlag</i>	247

<i>VLScgAllowLocalRequestLockCrit</i>	247
<i>VLScgSetLocalRequestLockCrit</i>	248
License Generation Function	248
<i>VLScgGenerateLicense</i>	249
License Decode Function	250
<i>VLScgDecodeLicense</i>	250
License Meter Related Functions	252
<i>VLScgGetLicenseMeterUnits</i>	252
<i>VLScgGetTrialLicenseMeterUnits</i>	253
License Revocation Ticket Decode Function	254
<i>VLScgDecodeLicenseRevocationTicket</i>	254
<i>The VLSrevocationTicketInfoT Structure</i>	256
Chapter 5 – Redundancy API	259
Redundancy Functions and API	260
<i>VLsaddFeature</i>	262
<i>VLsaddFeatureExt</i>	264
<i>VLsaddFeatureToFile</i>	265
<i>VLsaddServerToPool</i>	267
<i>VLschangeDistbCrit</i>	268
<i>VLsdelServerFromPool</i>	269
<i>VLsdiscoverExt</i>	272
<i>VLsgetDistbCrit</i>	275
<i>VLsgetDistbCritToFile</i>	277
<i>VLsgetFeatureInfoToFile</i>	279
<i>VLsgetHostName</i>	281
<i>VLsgetLeaderServerName</i>	282
<i>VLsgetHostAddress</i>	284
<i>VLsgetLicSharingServerList</i>	285
<i>VLsgetPoolServerList</i>	287
<i>VLssetBorrowingStatus</i>	288
<i>VLssetServerLogState</i>	290
Chapter 6 – License Queuing API	293
License Queuing Example Code	293

License Queuing Functions	296
<i>VLSqueuePreference Struct</i>	297
<i>VLSserverInfo Struct</i>	298
<i>VLSgetQueuedClientInfo Struct</i>	298
<i>VLSqueuedRequest and VLSqueuedRequestExt</i>	300
<i>VLSgetQueuedClientInfo</i>	306
<i>VLSremoveQueuedClient</i>	308
<i>VLSremoveQueue</i>	310
<i>VLSgetHandleStatus</i>	311
<i>VLSupdateQueuedClient</i>	312
<i>VLSgetQueuedLicense</i>	314
<i>VLSinitQueuePreference</i>	317
 Chapter 7 – Commuter License API	319
Commuter License Related Functions	319
<i>VLSCommuterInfo</i>	320
<i>VLSgetCommuterInfo</i>	322
<i>VLSgetAndInstallCommuterCode</i>	323
<i>VLSuninstallAndReturnCommuterCode</i>	325
<i>VLScleanExpiredCommuterCode</i>	325
Get Commuter Locking Code from Remote Computer (<i>VLSgetMachineIDString</i>)	326
lock_selector Values	327
Checking Out a Remote Authorization (<i>VLSgetCommuterCode</i>)	328
Installing a Remote Commuter Authorization (<i>VLSinstallCommuterCode</i>)	330
 Chapter 8 – Capacity License API	333
Capacity License Related Functions	333
<i>VLSrequestExt2</i>	334
<i>VLSgetFeatureInfoExt</i>	342
<i>VLSgetCapacityList</i>	344
<i>VLSgetClientInfoExt</i>	346
<i>VLSdeleteFeatureExt</i>	348
<i>VLSgetCapacityFromHandle</i>	350
<i>VLSsetTeamId</i>	350
<i>VLSsetTeamIdValue</i>	350

Chapter 9 – Upgrade License API 353

Upgrade License Code Generator API.....	353
<u>codeT Struct</u>	355
<i>VLSucgInitialize</i>	357
<i>VLSucgCleanup</i>	357
<i>VLSucgReset</i>	358
<i>VLSucgGetNumErrors</i>	359
<i>VLSucgGetErrorLength</i>	360
<i>VLSucgGetErrorMessage</i>	361
<i>VLSucgPrintError</i>	362
<i>VLSucgAllowBaseFeatureName</i>	363
<i>VLSucgSetBaseFeatureName</i>	364
<i>VLSucgAllowBaseFeatureVersion</i>	365
<i>VLSucgSetBaseFeatureVersion</i>	366
<i>VLSucgAllowUpgradeCode</i>	367
<i>VLSucgSetUpgradeCode</i>	368
<i>VLSucgAllowUpgradeFlag</i>	369
<i>VLSucgSetUpgradeFlag</i>	370
<i>VLSucgAllowUpgradeVersion</i>	371
<i>VLSucgSetUpgradeVersion</i>	372
<i>VLSucgAllowUpgradeCapacity</i>	373
<i>VLSucgSetUpgradeCapacityUnits</i>	374
<i>VLSucgSetUpgradeCapacity</i>	376
<i>VLSucgGenerateLicense</i>	377
<i>VLSucgGetLicenseMeterUnits</i>	379
<i>VLgenerateUpgradeLockCode</i>	380
Upgrade License Decode API	381
<u>ulcCode Struct</u>	382
<i>VLsdecodeUpgradelockCode</i>	383
<i>VLsucgDecodeLicense</i>	384

Chapter 10 – Usage Log Functions 387

<i>VLschangeUsageLogFileName</i>	387
<i>VLsgetUsageLogFileName</i>	388

Chapter 11 – Utility Functions	389
<i>VLSScheduleEvent</i>	389
<i>VLSDisableEvents</i>	390
<i>VLSeventSleep</i>	391
 Chapter 12 – System Initialization Functions	393
<i>sntlInitStandaloneSystem</i>	393
<i>sntlInitNetworkSystem</i>	396
Lsgeninit Utility	398
 Appendix A – Sample Applications	397
 Appendix B – Customization Features.....	399
Initializing the Server	401
<i>VLServerVendorInitialize</i>	401
<i>VLSeventAddHook</i>	401
Protecting Against Time Clock Changes	404
<i>VLConfigureTimeTamper</i>	405
<i>VLISClockSetBack</i>	407
Encrypting License Codes	407
<i>VLSEncryptLicense</i>	408
<i>VLSDecryptLicense</i>	410
Encrypting Messages	412
<i>VLSEncryptMsg</i>	412
<i>VLSDecryptMsg</i>	414
Changing the Default Port Number	415
<i>VLChangePortNumber</i>	415
Customizing the Host ID	416
<i>Creating the Custom Host ID Function</i>	417
<i>Registering the Custom Host ID Function on the Server</i>	418
<i>Registering the Custom Host ID Function on the Client</i>	418
<i>Building the Server</i>	419
<i>Creating an Updated Client ID Generator</i>	419
<i>Using a Customized Host ID</i>	419
Customizing Upgrade Licenses	420

<i>VLSEncryptUpgradeLicense</i>	420
<i>VLSDecryptUpgradeLicense</i>	421
Setting License Server Information	421
<i>Setting Vendor Specific Information in a License Server (VLSsetServerInfo)</i>	421
Customizing Stand-alone License File Names (VLSsetFileName)	422
Using a Custom Locking Code	423
<i>Step 1 - Rebuilding License Server</i>	424
Compiler Required	424
Files Required	424
Required Changes to Server Source Code	425
Steps to Rebuilding the License Server	425
<i>Step 2 - Rebuilding echoid.exe</i>	426
Compiler Required	426
Files Required for echoid.exe	426
Required Changes to echoid.exe	427
Steps to Rebuilding echoid.exe	427
<i>Step 3 - Modifying Client Application</i>	427
<i>Overall Process of Using a Rebuilt License Server and Rebuilt echoid.exe</i>	428
Adding Additional Security to Licenses Generated by WlscGen	428
 Appendix C – Sentinel LM Error and Result Codes	431
 Appendix D – Error Codes for License Generation Functions....	453
 Appendix E – Error Codes for Upgrade License Functions	461
 Appendix F – File Formats	467
License Code File Format	467
Configuration File Format	468
Log File Format	472
 Index	475

Preface

Thank you for choosing the Sentinel LM™—a world-class license management solution for your software!

The *Sentinel LM Programmer's Reference Manual* contains information on using the Sentinel LM Application Library.

Sentinel LM Documentation Resources

The Sentinel LM product includes several manuals, all designed to work in conjunction with each other.

Manual	What's in it?	Who should read it?
<i>Sentinel LM Developer's Guide</i>	All the steps necessary to protect, package, and ship a stand-alone or network application protected with Sentinel LM-Shell or the Sentinel LM Application Library.	Developers using Sentinel LM-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>Sentinel LM Programmer's Reference Manual</i>	Description of the Sentinel LM Application Library.	Developers who are using the Sentinel LM Application Library to protect their applications. This manual assumes you are familiar with the C programming language. Other language interfaces are also available.

Manual	What's in it?	Who should read it?
<i>Sentinel LM System Administrator's Online 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.
<i>Sentinel LM Release Notes</i>	Information on the features added to the current release. Also contains late-breaking information that was not available when the manuals were completed.	Any developer using Sentinel LM.


Who Should Read This Manual?

The *Sentinel LM Programmer's Reference Manual* provides detailed information about the APIs that constitute the Sentinel LM Application Library. It has been especially designed for developers who need to write their own code using the Sentinel LM Application Library to protect their application.

To be able to understand the *Sentinel LM Programmer's Reference Manual*, you should be familiar with the C programming language, although other language interfaces are available.

Conventions Used in This Manual

Please note the following conventions used in this manual:

Convention	Meaning
Select	Use the arrow keys or mouse to select an item on a menu, a field in a window or an item in a list.
Click	Press the primary mouse button once. The primary mouse button is typically the left button, but may be reassigned to the right button.
<code>Courier</code>	Denotes syntax, prompts and code examples. Bold Courier type represents characters that you type; for example: logon .
Bold Lettering	Words in boldface type represent keystrokes, menu items, window names or fields.
<i>Italic Lettering</i>	Words in italic type represent file names and directory names.
	This warning icon flags any potential pitfalls that we think you may need to be careful of.
Note:	Used for highlighting notes related to a specific section.
Tip:	Denotes tips that should be remembered while trying to achieve a specific result.

About This Manual

This manual provides detailed information about the APIs that constitute the Sentinel LM Application Library. The APIs are enlisted in the chapters described below:

Chapter/Appendix	Description
Chapter 1 – Introduction	Shows how Sentinel LM is put together.
Chapter 2 – Protecting Your Application with the Application Library	Provides instructions and information on client library functions and compiling applications.
Chapter 3 – Sentinel LM Client API	Provides a complete reference of all client functions.
Chapter 4 – License Code Generation API	Allows you to write a program to generate license codes.
Chapter 5 – Redundancy API	Summarizes the redundancy functions.
Chapter 6 – License Queuing API	Explains the license queuing functions.
Chapter 7 – Commuter License API	Summarizes the commuter license related functions.
Chapter 8 – Capacity License API	Summarizes the capacity license related functions.
Chapter 9 – Upgrade License API	Explains the upgrade licensing functions.
Chapter 10 – Usage Log Functions	Explains the usage log functions.
Chapter 11 – Utility Functions	Summarizes functions for the UNIX platform.
Appendix A – Sample Applications	Lists source code for the sample programs and utilities.
Appendix B – Customization Features	Lists the features that can be customized.
Appendix C – Sentinel LM Error and Result Codes	Lists client function return codes and their description.
Appendix D – Error Codes for License Generation Functions	Lists license generation function return codes.

Chapter/Appendix	Description
Appendix E – Error Codes for Upgrade License Functions	Lists upgrade license generation function return codes.
Appendix F – File Formats	Summarizes all the file formats, like license code file format, configuration file format, and log file format.

Contacting Technical Support

We are committed to supporting Sentinel LM. If you have questions, need additional assistance, or encounter a problem, please contact Technical Support:

Technical Support Contact Information

Customer Connection Center (C ³)	
<i>http://c3.safenet-inc.com</i>	
Americas	
Internet	<i>http://www.safenet-inc.com/support/index.asp</i>
E-mail	support@safenet-inc.com
United States	
Telephone	(800) 545-6608
Europe	
E-mail	eutechsupport@safenet-inc.com
France	
Telephone	0825 341000
Fax	+44 (0) 1276 608080
Germany	
Telephone	01803 7246269
Fax	+44 (0) 1276 608080

Technical Support Contact Information (Continued)

United Kingdom	
Telephone	+44 (0) 1276 608000
Fax	+44 (0) 1276 608080
Pacific Rim	
E-mail	support@safenet-inc.com
Australia and New Zealand	
Telephone	(61) 3 9882 8322
Fax	(61) 3 9882 0588
China	
Telephone	(86) 10 8851 9191
Fax	(86) 10 6872 7342
India	
Telephone	(91) 11 2691 7538
Fax	(91) 11 2633 1555
Taiwan and Southeast Asia	
Telephone	(886) 2 27353736
Fax	(886) 2 27352383

Export Considerations

SafeNet, Inc. offers products that are based on encryption technology. The Bureau of Industry and Security (BIS) in the U.S. Department of Commerce administers the export controls on SafeNet's commercial encryption products.

Rules governing exports of encryption can be found in the Export Administration Regulations (EAR), 15 CFR Parts 730-774, which implements the Export Administration Act ("EAA" 50 U.S.C. App. 2401 et seq.).

Important Note: BIS requires that each entity exporting products be familiar with and comply with their obligations described in the Export Administration Regulations. Please note that the regulations are subject to change. We recommend that you obtain your own legal advice when attempting to export any product that uses encryption. In addition, some countries may restrict certain levels of encryption imported into their country. We recommend consulting legal counsel in the appropriate country or the applicable governmental agencies in the particular country.

We Welcome Your Comments

To help us improve future versions of Sentinel LM documentation, we want to know about any corrections, clarifications or further information you would find useful. When you contact us, please include the following information:

- The title and version of the guide you are referring to
- The version of the Sentinel LM software you are using
- Your name, company name, job title, phone number, and e-mail address

Send us e-mail at:

`techpubs@safenet-inc.com`

Chapter 1

Introduction

Sentinel LM 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 protected application, and a license server to receive and act on authorization requests. Access to the license server is made possible by an Application Programming Interface (API). API functions are implemented in the Sentinel LM 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 Sentinel LM Application Library

The Sentinel LM Client Library is used to integrate Sentinel LM API calls to 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 a license 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 the hostname of the machine running Sentinel LM protected application, 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 Network 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. Sentinel LM 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 Sentinel LM 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 Sentinel LM.

The first call is `VLSInitialize`, this API initializes the client library. `VLSInitialize` is called 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 *FeatureName* which identifies your product and *Version* which specifies the version number of that product. The feature name and version are also contained in the license code, and must match before authorization to run the program can be given.

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 (VLSinitialize())
{
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 */
VLScleanup();
}
```

Language Interfaces Supported

Different language interfaces are supported by Sentinel LM to allow you to incorporate Sentinel LM Application Library calls in applications coded in different programming languages. Among the language interfaces supported are Microsoft Visual C/C++, Microsoft Visual Basic, Microsoft .NET (C# and Visual Basic), Java, COMObjects, PowerBuilder, Borland C, and Delphi. Check the *Interface* directory in the Sentinel LM directory for the latest language interfaces.

Other interfaces are available, and will continue to become available over time. Contact your SafeNet representative for information on new interfaces and specific versions supported. If your application does not use one of the supported interfaces, see the *Sentinel LM Developer's Guide* for information on using the Sentinel LM-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 SafeNet'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 Sentinel LM system are encouraged to install the Windows version of Sentinel LM 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 Sentinel LM 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 “Tracing Sentinel LM Operation” on page 141.) 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 Sentinel LM license server must be started using the appropriate option(s). See the online *System Administrator's Guide* for details.

Disabling Licensing

The macro `NO_LICENSE` in the *lserv.h* file can be set to completely disable licensing for debugging. This replaces all Sentinel LM function calls with void statements. Don't forget to re-enable licensing before preparing your application for shipment.

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 Sentinel LM 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 Sentinel LM Application Library to protect your application in the following ways:

- Stand-alone
- Network
- Integrated

For a full discussion of the Sentinel LM Application Library calls, refer to other chapters in this book.

Stand-alone Application Protection

When Sentinel LM protects your stand-alone application, it embeds within it the license management function that checks for a valid license code before the application will run. Very simply, if a valid license code exists, the application will run; if Sentinel LM cannot find a valid license code, the application will not run.

You may also restrict the application to run only on a specific computer by locking the application to that computer.

For more information on stand-alone licensing and on using Sentinel LM-Shell, refer to the *Sentinel LM Developer's Guide*.

Network Application Protection

To protect an application that is to be run on multiple computers Sentinel LM moves the license management function outside of the protected application itself and uses an external license server to verify that a valid license code exists before granting authorization to run the application.

Note: The license server can run on any computer on the network, including a computer on which the protected application is run.

The license server keeps track of all Sentinel LM licenses and handles requests from network users who want to run your application, granting authorization to the requesters to allow them to run the application, and denying requests when all licenses are in use.

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. We will discuss using the following calls:

- VLInitialize
- LSRequest
- VLSdiscover
- LSUpdate
- LSRelease
- VLScleanup

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 VLInitialize.

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. The feature name and version information must match what you give the license code generator when you generate a license code authorizing use of this application.

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 293.

Automatic License Server Detection

If you provide no information to Sentinel LM protected application on the location of a license server, a Sentinel LM-licensed application uses a broadcast mechanism to determine the existence of an active Sentinel LM 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, Sentinel LM 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, Sentinel LM 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 Sentinel LM cannot find an `LSHOST` environment variable or *LSHOST/lshost* file, or if it cannot find the license servers specified in that variable or file, Sentinel LM uses its broadcast mechanism to find any license server on the local subnet which contains the desired feature/version.

When there are multiple Sentinel LM 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 Sentinel LM client library will return an error, and/or the application will terminate.

This situation can be avoided by using the Sentinel LM client library call, `VLSdiscover`, to locate all of the Sentinel LM license servers on the local subnet, and query each of them individually for a license. You will need to call `VLSsetContactServer` 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 Sentinel LM 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 recommended 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 users 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, “Sentinel LM Client API,” on page 21.

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 a compiler.

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`

Integrated Application Protection

Sentinel LM provides an integrated library that allows an application to switch between stand-alone and network licensing. The benefit of using the Sentinel LM integrated library lies in the fact that the license type (stand-alone/ network) can be decided when the application is run, not when the application is compiled.

Dynamic Switching Between Stand-alone and Network Licensing

One of the most important benefits Sentinel LM offers is *dynamic switching* when an application is protected using the Sentinel LM integrated library and when the LSFORCEHOST environment variable is not set to any value. Dynamic switching occurs when an application is protected with the integrated library and it is not defined at that time whether the application will obtain a stand-alone license on the computer on which it is running (stand-alone mode) or from a license server (network mode). At the time the application is run, the decision is made based on the availability of a license. In an effort to make dynamic switching more flexible and consistent, a change has been made in the way the LSHOST environment variable is interpreted by Sentinel LM.

The LSHOST environment variable is now interpreted as stating a *preference* for where a license will be obtained:

- If LSHOST on the client computer is not set to anything, the application protected with the integrated library will search first for a stand-alone license on that computer; if it is not found, the application will start looking for network license servers for a network license.
- If LSHOST is set to anything other than NO-NET, the application will look first for a license server on the computers named by LSHOST; if it cannot find a license server on the computers defined by LSHOST that can grant the requested license, Sentinel LM will do a broadcast on the network to look for a network license server that has the license. If the appropriate license server is not found on the network, Sentinel LM will start looking for a stand-alone license.

If LSHOST is set to NO-NET, Sentinel LM will first look for a stand-alone license; if it cannot find one, it will start looking for a license on a network license server.

Tip: You can use the LSFORCEHOST environment variable to force Sentinel LM to look for a license on one specific computer. Or to force Sentinel LM to **ONLY** look for a stand-alone license, set LSFORCEHOST to NO-NET.

Examples of Dynamic Switching

- If LSHOST is set to NO-NET followed by a list of computers containing license servers:

```
LSHOST NO-NET:ACCTNG1:MIS2:ORION
```

If a stand-alone license is found, the application will use it. If a stand-alone license is not found, Sentinel LM switches to searching the computers on the network named in the LSHOST environment in the order listed for a license server containing a network license that can be granted.

- If LSHOST is set to a single computer name, Sentinel LM starts looking for a network license from a license server on that computer. If it doesn't find a license, it performs a broadcast on the network to look for a network license server that has the license. If an appropriate license server is not found on the network, Sentinel LM switches to stand-alone mode and looks for a stand-alone license on the computer on which the application is running.

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*, *shell32.lib*, *ole32.lib*, *oleaut32.lib*, *uuid.lib*, *odbc32.lib*, *odbc32.lib*, and *netapi32.lib*. Please see the *sample32.mak* make file in the Sentinel LM \demo\MsvcDev\Samples directory for details on how to link your application with the Sentinel LM client library.

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

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

Windows Static Libraries

Library	Description
<i>Isapiw32.lib</i>	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.
<i>Issrv32.lib</i>	This library is the same as <i>Isapiw32.lib</i> .
<i>Isclws32.lib</i>	The network client library for Windows applications. This library allows your application to acquire licenses via network only.
<i>Isnnet32.lib</i>	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

Windows Dynamic Libraries and Import Libraries

Library	Description
<i>lsapiw32.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>lsapiw32.lib</i>	This library is the import library for <i>lsapiw32.dll</i> , (Microsoft format).
<i>lssrv32.dll</i>	This library is the same as <i>lsapiw32.dll</i> .
<i>lsclws32.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>lsapiw32.dll</i> , you may use the <i>lsapiw32.lib</i> import library supplied with the installation.
<i>lsnnet32.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>lsapiw32.dll</i> , you may use the <i>lsapiw32.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 executable 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 *Sentinel LM* shipment directory. 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.

Testing and Debugging Your Application

The Sentinel LM client library has built-in default responses to all exceptional conditions that may arise. Custom exception handlers can also be registered with the Sentinel LM Application Library. Each error condition handler must be explicitly registered. The default error handlers will be used for those errors for which no explicit handlers are registered. See Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Disabling Licensing

The macro, `NO_LICENSE`, in the *lserv.h* file can be set to completely disable software licensing, for instance during debugging. This replaces all Sentinel LM function calls with void statements. Be sure to enable before shipping your application.

Library Tracing

You can also enable the tracing of internal operation of the Sentinel LM client library by calling `VLSsetTraceLevel`. See Chapter 3, “Tracing Sentinel LM Operation,” on page 141.

Sample Programs

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 Sentinel LM. For Windows platforms, the file is called *sample32.mak*.

Below is a list of the available sample programs, utilities, and Sentinel LM components. However, for the very latest list of such files included on the release, see the `\MsvcDev\Samples` and `\MsvcDev\Custom` subdirectories in the Sentinel LM installation directory.

Sample Program Summary

The following table lists the sample programs, the features illustrated in each, and on which platforms the programs are available.

Sample Programs, Features, and Platforms

Programs	Features	Platforms
bounce	Simple function macros	Windows NT/2000/XP, Windows 95/98/ME
dots1	Simple function macros	UNIX, Windows NT/2000/XP, Windows 95/98/ME
tutor1	Simple function macros	UNIX
timer	Simple function macros and using timer signals	UNIX
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:

Customization Sample Files

Component	File(s)
Linking	Makefile
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>
Challenge-response	<i>crexamp.c</i> , <i>chalresp.[c h]</i> , <i>md4.[c h]</i>

On the Windows platforms the following components/files are available:

Customization Sample Files on Windows

Component	File(s)
the license server	<i>lservdown.[c dsp]</i> , <i>lserv.h</i>
licence generator	<i>echoid32.dsp</i> , <i>echomain.c</i>
lsdecode	<i>lsde.o</i>
lslic	<i>lslic.[c dsp]</i>
lsmon	<i>lsmon.[c dsp]</i>
lswhere	<i>lswhere.[c dsp dsw ncb opt]</i>
Challenge-response	<i>crexamp.c</i> , <i>chalresp.[c h]</i> , <i>md4.[c h]</i>

Notes on Security

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

Sentinel LM 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, “Sentinel LM Client API,” on page 21 or refer to *Windows License Code Generator Help* for more 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 Sentinel LM license server is configured to detect tampering of the system clock.

The Sentinel LM 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.

Chapter 3

Sentinel LM Client API

Using the Sentinel LM 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.

Note: Throughout this manual, we refer to getting license codes and returning or releasing license codes. Although it is convenient to think of license management this way, it is important to realize that Sentinel LM does not physically transfer license codes from the license server to the client or vice versa, but instead grants or denies permission to use a license code depending on the license code contents.

- 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 authorization. `VLSinitialize` should be called only once. During program termination, calls are made to `LSRelease` to release the authorization 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 Sentinel LM Application Programming 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 Sentinel LM 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 136.

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 Sentinel LM network client library to a stand-alone client library.
- **Network:** For any operation requiring a network license server.
- **Integrated:** For both stand-alone and network operations. We recommend you to 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 authorizations can be requested within an application for a feature and feature version. Each authorization must be released and updated separately as the license server treats these authorizations as separate clients. A handle that uniquely identifies an authorization will be returned for each LSRequest call using the argument, *lshandle*. This handle is also used in other Sentinel LM function calls.

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 performing all Sentinel LM functions for that single thread. We would also suggest you not to call different LM functions from separate threads

Available client licensing function calls can be separated into the following categories:

- Basic client licensing functions
- Capacity
- Challenge-response
- Client configuration
- Client query
- Client utility
- Commuter
- Error handling
- Feature query
- Queuing
- Redundancy
- Revocation
- System Initialization
- Tracing Sentinel LM operation

Basic Client Licensing Functions

Quick Client Licensing Functions

The following table summarizes the quick client functions:

Quick Client Licensing Function

Function	Description
VLSlicense	Performs single-call licensing
VLSdisableLicense	Disables single-call licensing

VLSlicense

Client	Server	Static Library	DLL
✓		✓	✓

Initializes contact with the license server, requests authorization and automatically updates the license.

Syntax

```
LS_STATUS_CODE VLSlicense(  
    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 (out)</i>	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 *Sentinel LM Developer's Guide* for details.

Description This function obtains a 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 Sentinel LM 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:

VLSlicense Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none">• <i>featureName</i> is NULL• <i>version</i> is NULL Both feature and version cannot be NULL at the same time.
VLS_CALLING_ERROR	<ul style="list-style-type: none">• <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, version and capacity.
VLS_TRIAL_LIC_EXHAUSTED	Trial license has expired.
LS_LICENSE_EXPIRED	License has expired.
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.

VLSlicense Error Codes (Continued)

Error Code	Description
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_NO_NETWORK	Failed to initialize Winsock wrapper. (Only applicable if using network-only library.) Generic error indicating network failure.
VLS_INTERNAL_ERROR	An internal error has occurred in the processing.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

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:

VLSdisableLicense Error Codes

Error Code	Description
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.
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.

VLSdisableLicense Error Codes (Continued)

Error Code	Description
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 Sentinel LM files.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Standard Client Licensing Functions

The following table summarizes the standard client functions:

Standard Client Licensing Functions

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

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 Sentinel LM 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.

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

VLSInitialize Error Codes

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

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLScleanup” on page 41.

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 char    *unitsReqd,
    unsigned char    *logComment,
    unsigned char    *challenge,
    LSHANDLE         *lshandle );

```

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.
<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> (IN/OUT)	The number of licenses required. The license server verifies that the requested number of units exist and may reserve those units. 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.

Argument	Description
<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. Maximum of 100 characters.
<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 50.
<i>lshandle (OUT)</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 is computed if a non-NULL value is passed for the *challenge* argument. At minimum, the *featureName* 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:

LSRequest Error Codes

Error Code	Description
VLS_CALLING_ERROR	<ul style="list-style-type: none"> <i>lshandle</i> is NULL. <i>challenge</i> argument is non-NULL, but cannot be understood. <p>Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.</p>
VLS_APP_UNNAMED	<ul style="list-style-type: none"> <i>featureName</i> is NULL <i>version</i> is NULL. <p>Both feature name and version cannot be Null at the same time.</p>
VLS_NO_LICENSE_GIVEN	<ul style="list-style-type: none"> <i>unitsReqd</i> is zero <i>lshandle</i> is not a valid handle. <p>License is only available at license server that does not match mode settings, e.g. network license available when stand-alone mode, etc.</p>
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature and version.
LS_INSUFFICIENTUNITS	<ul style="list-style-type: none"> License server does not currently have sufficient licensing units for requested feature to grant license. The <i>units_reqd</i> parameter of the call contains the hard limit of the feature for which authorization was requested if this request exceeded the hard limit of the license.
LS_LICENSE_EXPIRED	License is expired.
VLS_TRIAL_LIC_EXHAUSTED	Trial license expired or trial license usage exhausted.

LSRequest Error Codes (Continued)

Error Code	Description
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	An internal error has occurred in the processing.
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, “Sentinel LM Error and Result Codes,” on page 431.

See Also

- “VLSrequestExt2” on page 45
- “Challenge-response Mechanism” on page 50
- “VLSsetTimeoutInterval” on page 68
- “VLSqueuedRequest and VLSqueuedRequestExt” on page 300

LSUpdate

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 license renewal. You do not need to call LSUpdate unless you disable automatic license renewals. Even when automatic license renewal is active, you should check periodically on the update status by calling VLSgetRenewalStatus. (See “VLSgetRenewalStatus” on page 78.)

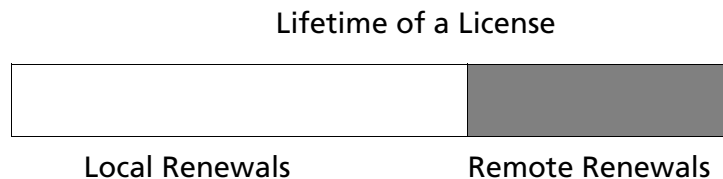
If automatic license renewals are disabled (if the platform you are working on doesn’t support timers or you don’t want to rely on timers to renew the

license), the client must call LSUpdate periodically to renew its license and inform the license server of its continued need for a license.

If you do call LSUpdate manually to verify the client is still attached to the license server, you should disable automatic renewals before calling LSUpdate.

Local Vs. Remote License Renewal

In order to reduce network traffic and communication overhead, Sentinel LM 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 Sentinel LM API also provides the function, VLSdisableLocalRenewal, which forces all future LSUpdate requests to go to the license server.

VLSgetRenewalStatus provides information on whether the last successful update was local or remote. See page 75 for these and other related function calls.

Note: 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:

LSUpdate Error Codes

Error Code	Description
VLS_CALLING_ERROR	<ul style="list-style-type: none"> <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, version and capacity.
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.

LSUpdate Error Codes (Continued)

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

See Also

- “VLSbatchUpdate” on page 47
- “VLSsetTimeoutInterval” on page 68
- “VLSdisableLocalRenewal” on page 76
- “VLSenableLocalRenewal” on page 77
- “VLSisLocalRenewalDisabled” on page 77
- “VLSgetRenewalStatus” on page 78
- “VLSsetRemoteRenewalTime” on page 81

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   *log_comment);

```

Argument	Description
<i>lshandle</i>	The handle returned by the corresponding LSRequest.
<i>units_consumed</i>	Number of units required to be 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. Maximum of 100 characters is allowed.

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:

LSRelease Error Codes

Error Code	Description
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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLScleanup

Client	Server	Static Library	DLL
✓		✓	✓

Cleans up the client library.

Syntax `LS_STATUS_CODE VLScleanup(void);`

This function has no arguments.

Description After all Sentinel LM 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, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSinitialize” on page 29.

Advanced Client Licensing Functions

The following table summarizes the advanced client functions:

Advanced Client Licensing Functions

Function	Description
VLSinitialize	Initializes the client library.
VLSrequestExt	Requests an authorization license.
VLSrequestExt2	Supports capacity and non-capacity requests.
VLSreleaseExt	Releases an authorization license.
VLScleanup	Called when finished using the client library.
VLSbatchUpdate	Updates several license codes at once.

VLSrequestExt

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLSrequestExt(  
    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>	<ul style="list-style-type: none">• Cannot be Null• 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.
<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>unitsReqd</i> (IN/OUT)	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.

Argument	Description
<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 50.
<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 401.

Description Use VLSrequestExt when using license server hooks. Before calling VLSrequestExt, you must call VLSinitServerInfo. (See “VLSinitServerInfo” on page 66.)

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

VLSrequestExt Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none"> <i>featureName</i> is NULL <i>version</i> is NULL Both feature name and version cannot be Null at the same time.
VLS_CALLING_ERROR	<ul style="list-style-type: none"> <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	<ul style="list-style-type: none"> <i>unitsReqd</i> is zero <i>lshandle</i> is not a valid handle.

VLSrequestExt Error Codes (Continued)

Error Code	Description
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature, version and capacity.
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.

VLSrequestExt Error Codes (Continued)

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

See Also

- “Challenge-response Mechanism” on page 50
- “VLSeventAddHook” on page 401

VLSrequestExt2

See “VLSrequestExt2” on page 334.

VLSreleaseExt

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```

LS_STATUS_CODE VLSreleaseExt (
LS_HANDLE      lshandle,
unsigned long   *unused1,
unsigned char   *logComment,

```

```
VLSserverInfo      *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. Maximum of 100 charactersb.
<i>serverInfo</i>	This information is passed to the license server for use in server hook functions. See “VLSeventAddHook” on page 401.

Returns

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

VLSreleaseExt Error Codes

Error Code	Description
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 hostName 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, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSeventAddHook” on page 401

VLSbatchUpdate

Client	Server	Static Library	DLL
✓		✓	✓

Updates several license authorization at once. Currently the client library defaults to automatic license renewal. You do not need to call this unless you disable the automatic license renewal. Please note that this does not updates capacity authorizations.

Syntax

```
LS_STATUS_CODE VLSbatchUpdate(
    int          *numHandles,
    LS_HANDLE    *lshandle,
    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:

VLSbatchUpdate Error Codes

Error Code	Description
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, version and capacity.
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.

VLSbatchUpdate Error Codes (Continued)

Error Code	Description
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

- “LSUpdate” on page 35
- “Challenge-response Mechanism” on page 50
- “VLSsetTimeoutInterval” on page 68
- “VLSdisableLocalRenewal” on page 76
- “VLSenableLocalRenewal” on page 77
- “VLSisLocalRenewalDisabled” on page 77
- “VLSsetRemoteRenewalTime” on page 81

Challenge-response Mechanism

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

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. 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.

Sentinel LM 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 217, “VLScgSetNumSecrets” on page 219, and “VLScgSetSecrets” on page 218.

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.

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.
VLSgetNumberedMachineID	Obtains the CID or Ethernet fingerprint at the specified index location.
VLSmachineIDtoLockCode	Computes the <i>machineID</i> locking code.
VLSgetServerNameFrom Handle	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.

Client Configuration Functions (Continued)

Function	Description
VLSsetTimeoutInterval	Configures timeout behavior.
VLSgetTimeoutInterval	Retrieves timeout behavior parameters.
VLSsetHoldTime	Sets license hold time.
VLSsetSharedId/ VLSsetTeamId	Redefines shared ID functions.
VLSsetSharedIdValue/ VLSsetTeamIdValue	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 License Tokens” on page 75.

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 or IP address of the computer running the license server.

Description

Each licensed application must be aware of the location of a Sentinel LM 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, Sentinel LM 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 the local subnet to determine the existence and location of a Sentinel LM license server. If a client makes a Sentinel LM 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 Sentinel LM client library returns an error.

After a license is successfully requested (via `LSRequest` or its variants) Sentinel LM 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, Sentinel LM will always communicate with the license server from which it obtained the license, regardless of intervening `VLSsetContactServer` calls. The license server name set by `VLSsetContactServer` 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.

`VLSsetContactServer` 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.

The roles are summarized in the table below:

Linked with	Server Name	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.

In case of *libnonet.a*, Sentinel LM 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:

VLSsetContactServer Error Codes

Code	Description
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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSgetContactServer” on page 56.

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> (out)	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 the Sentinel LM 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:

VLSgetContactServer Error Codes

Error Code	Description
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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSsetContactServer” on page 53.

VLSsetServerPort

Client	Server	Static Library	DLL
✓		✓	✓

Sets the port number.

Syntax `int VLSsetServerPort(int);`

Description Defines the license server's communication port.

Returns Does not return anything.

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 *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: 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          0x3FF
```

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

VLSinitMachineID Error Codes

Error Code	Description
VLS_MACHINE_FAILURE_CODE	<i>machineID</i> is NULL.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetMachineID

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLSgetMachineID(  
    unsigned long    lock_selector_in,  
    VLSmachineID     *machineID,  
    unsigned long    *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. VLSgetMachineID allows you to use an Ethernet address as a locking criterion for UNIX computers.

Returns The status code, VLScg_SUCCESS, is always returned. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetNumberedMachineID

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLSgetNumberedMachineID(  
    unsigned long    lock_selector_in,  
    VLSmachineID     *machineID,  
    unsigned long    *lock_selector_out,  
    int              flag,  
    int              index,  
    int              reserved);
```

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.
<i>flag</i>	Specify VLS_GET_ETHERNET for multiple Ethernet. Specify VLS_GET_CID for cascaded CID keys.
<i>index</i>	Refers to the index of the device (starts from zero).
<i>reserved</i>	This parameter is reserved for future use.

Description This function:

- Sets the machineID struct for the current host (like VLSgetMachineID)
- Obtains the CID or Ethernet fingerprint at the specified index location. It can be useful in following cases:
 - ❑ Multiple Ethernet cards
 - ❑ Cascaded multiple CID keys

Returns The status code, VLScg_SUCCESS, is always returned. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSmachineIDtoLockCode

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE  VLSmachineIDtoLockCode(  
VLSmachineID    *machineID,  
unsigned long    lock_selector,  
unsigned long    *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. lockCode is an OUT parameter.

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, LS_SUCCESS, is returned if successful and if *lock_selector* is zero. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetServerNameFromHandle

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```

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

```

Argument	Description
<i>handle_id</i>	The handle returned by LSRequest or VLSrequestExt
<i>outBuf</i> (OUT)	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:

VLSgetServerNameFromHandle Error Codes

Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

VLSinitServerList

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE  VLSinitServerList(  
char            *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 the list of license servers specified along with enumerating the license servers running in the local subnet. However, if the VLSsetContactServer function is called, the VLSinitServerList settings will be overridden.

The *serverList* parameter should be in the same format as the last parameter of the VLSdiscover call, and have the same syntax. See “VLSdiscover” on page 112 for description of *optionFlag*. This 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:

VLSinitServerList Error Codes

Error Code	Description
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetServerList

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLSgetServerList(
char           *outBuf,
int            outBufSz);
```

Argument	Description
<i>outBuf</i> (OUT)	User allocated buffer to receive license server name
<i>outBufSz</i>	Size 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:

VLSgetServerList Error Codes

Error Code	Description
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, "Sentinel LM Error and Result Codes," on page 431.

VLSinitServerInfo

Client	Server	Static Library	DLL
✓		✓	✓

Syntax

```
LS_STATUS_CODE VLSinitServerInfo(  
    VLServerInfo *serverInfo);
```

Argument	Description
<i>serverInfo</i>	User allocated buffer that will contain initialized VLServerInfo.

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.

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> .

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:

VLSsetHostIdFunc Error Codes

Error Code	Description
<code>LS_NORESOURCES</code>	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSsetBroadcastInterval

Client	Server	Static Library	DLL
✓		✓	✓

Sets the broadcast interval.

Syntax

```
LS_STATUS_CODE VLSsetBroadcastInterval(
    long        interval);
```

Argument	Description
<i>interval</i>	The total time interval for broadcast (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:

VLSsetBroadcastInterval Error Codes

Error Code	Description
<code>LS_NORESOURCES</code>	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

***VL*SgetBroadcastInterval**

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the broadcast interval.

Syntax `long VLSgetBroadcastInterval (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 VLSgetBroadcastInterval returns the *total* length of broadcast attempts.

***VL*SsetTimeoutInterval**

Client	Server	Static Library	DLL
✓		✓	✓

Sets the timeout interval.

Syntax `LS_STATUS_CODE VLSsetTimeoutInterval(
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 re-sends the message a few times. Each time, the length of the timeout *interval* is slightly longer than the previous. VLSsetTimeoutInterval 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:

VLSsetTimeoutInterval Error Codes

Error Code	Description
LS_NORESOURCES	An error occurred in attempting to allocate memory needed by function.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetTimeoutInterval

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the timeout interval.

Syntax `long VLSgetTimeoutInterval ();`

Description When a licensed application sends a request to a license server and the license server does not respond, it re-sends 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 LSRequest or VLSqueuedRequest call for it to be applicable. Once a license is requested using LSRequest, the hold time is set for that application, and VLSsetHoldTime will not change it.

Returns

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

VLSsetHoldTime Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none"><i>featureName</i> is NULL<i>version</i> is NULL Both feature name and version cannot be NULL at the same time.
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, “Sentinel LM Error and Result Codes,” on page 431.

VLSsetSharedId/ VLSsetTeamId

Client	Server	Static Library	DLL
✓		✓	✓

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

Syntax

In case of normal license:

```
LS_STATUS_CODE VLSsetSharedId(
    int sharedId,
    unsigned long (*mySharedIdFunc) (char *));
```

In case of capacity license:

```
LS_STATUS_CODE VLSsetTeamId(
    int teamId,
    unsigned long (*mySharedIdFunc) (char *));
```

Argument	Description
<i>sharedId/ teamId</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

VLSsetSharedId/VLSsetTeamId must be used to register a customized *sharedID/teamID* function with the Sentinel LM 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.

VLSsetSharedId should be used in case of normal license and VLSsetTeamId should be used in case of capacity 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 flexibility 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, Sentinel LM 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.

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

VLSsetSharedId/ VLSsetTeamId Error Codes

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

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSsetSharedIdValue/ VLSsetTeamIdValue

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 In case of normal license:

```
LS_STATUS_CODE VLSsetSharedIdValue(
    int          sharedId,
    char          *sharedIdValue );
```

In case of capacity license:

```
LS_STATUS_CODE VLSsetTeamIdValue(
    int          teamId,
    char          *teamIdValue );
```

Argument	Description
<i>sharedId/ teamId</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/ VLSsetTeamId` and can be used to register a customized *sharedId/ teamId* value with the Sentinel LM client library. You can explicitly provide the *sharedId/ teamId* itself using this function. The value of the *sharedId/ teamId* must be passed through the character buffer. All *sharedId/ teamId* character buffers will be truncated to 32 bytes. If you call both `VLSsetSharedId` and `VLSsetSharedIdValue/ VLSsetTeamId` and `VLSsetTeamIdValue`, `VLSsetSharedId/ VLSsetTeamId` has priority and the value set by `VLSsetSharedIdValue/ VLSsetTeamIdValue` will be ignored.

The same concept applies to `VLSsetSharedIdValue/ VLSsetTeamIdValue` function as `VLSsetSharedId/ VLSsetTeamId` function. The difference between `VLSsetSharedId/ VLSsetTeamId` and `VLSsetSharedIdValue/ VLSsetTeamIdValue` lies in the fact that `VLSsetSharedId/ VLSsetTeamId` function will make the `VLSrequestExt` internally send different IDs as returned by the Developer-Defined functions, whereas `VLSsetSharedIdValue/ VLSsetTeamIdValue` 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/ VLSsetTeamId`. If no developer defined function is found then the priority is passed to the `SharedIdValue` as set by `VLSsetSharedIdValue/ VLSsetTeamIdValue` function. If neither the developer defined function nor the developer defined `SharedIdValue` is found, the default functions are used.

Note: VLSsetSharedIdValue should be used in case of normal license and VLSsetTeamIdValue should be used in case of capacity license.

Returns

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

VLSsetSharedIdValue Error Codes

Error Code	Description
VLS_CALLING_ERROR	An error occurred in the use of an internal buffer.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Local vs. Remote Renewal of License Tokens

The license token (also known as a *key*) issued by the license server to a client upon request has to be renewed by calling LSUpdate within the period of the license lifetime, or if you are using auto-timers, this will be done automatically for you. 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 35.

The following function calls relate to license renewal:

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.

License Renewal Functions (Continued)

Function	Description
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 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, “Sentinel LM Error and Result Codes,” on page 431.

See Also ■ “LSUpdate” on page 35
 ■ “VLSenableLocalRenewal” on page 77

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 key life-time. Resets the license renewal for all future license renewals made using LSUpdate 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, “Sentinel LM Error and Result Codes,” on page 431.

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” on page 35
- “VLSdisableLocalRenewal” on page 76

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.

Returns Returns the following error codes:

VLSisLocalRenewalDisabled Error Codes

Error Code	Description
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. 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 status codes:

VLSgetRenewalStatus Error Codes

Error Code	Description
VLS_NO_LICENSE_GIVEN	Generic error indicating that license was not updated.
LS_LICENSETERMINATED	Cannot update the license because the license has already expired.

VLSgetRenewalStatus Error Codes (Continued)

Error Code	Description
VLS_NO_SUCH_FEATURE	License server does not have license that matches requested feature, version and capacity.
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.

VLSgetRenewalStatus Error Codes (Continued)

Error Code	Description
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.
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” on page 35.

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>	<i>Version</i> of the feature.
<i>timeInSecs</i>	Time in seconds. Default time is 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 LSRequest call for it to be applicable. Once a license is requested using 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:

VLSsetRemoteRenewalTime Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none"> <i>featureName</i> is NULL <i>version</i> is NULL Both feature name and version cannot be NULL at the same time.
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, “Sentinel LM Error and Result Codes,” on page 431.

- See Also**
- “LSRequest” on page 31
 - “LSUpdate” on page 35

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:

VLSdisableAutoTimer Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

Client Query Functions

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

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.
VLSgetLicInUseFrom Handle	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, “Sentinel LM Error and Result Codes,” on page 431.

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 authorized by the Sentinel LM 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      key_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;
int       is_commuted;
long      structSz;
unsigned long  team_capacity;
unsigned long  total_resv_team_capacity;
unsigned long  reserved_team_capacity_in_use;
unsigned long  unreserved_team_capacity_in_use;
unsigned long  user_capacity_from_reserved;
unsigned long  user_capacity_from_unreserved;
int         total_team_tokens_resv;
int         reserved_team_tokens_in_use;
int         unreserved_team_tokens_in_use;
} VLSclientInfo;
```

Member	Description
<i>user_name</i>	The login name of the user using the application, where MAXLEN is set to 64 characters. 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, where MAXLEN is set to 64 characters. If the user does not belong to an explicitly named group, <i>DefaultGrp</i> is returned.

Member	Description
<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.
<i>host_name</i>	Name of the host/computer where the user is running the application, where MAXLEN is set to 64 characters. 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, where MAXLEN is set to 64 characters. 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. The maximum length of the string is set to 64 characters.
<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.
<i>structSz</i>	Size of VLSclientInfo structure.

Member	Description
<i>team_capacity</i>	Total capacity of the team.
<i>total_resv_team_capacity</i>	Total capacity reserved for the team.
<i>reserved_team_capacity_in_use</i>	Capacity given to clients from reserved capacity.
<i>unreserved_team_capacity_in_use</i>	Capacity given to clients from unreserved capacity.
<i>user_capacity_from_reserved</i>	Capacity given to users from reserved capacity.
<i>user_capacity_from_unreserved</i>	Capacity given to users from unreserved capacity.
<i>total_team_tokens_resv</i>	Total reserved units.
<i>reserved_team_tokens_in_use</i>	Units given to teams from reserved pool.
<i>unreserved_team_tokens_in_use</i>	Units given to teams from unreserved pool.

VLSgetClientInfo

Client	Server	Static Library	DLL
✓		✓	✓

Returns information about a client feature.

Syntax

```
LS_STATUS_CODE VLSgetClientInfo(  
    char          *featureName,  
    char          *version,  
    int           index,  
    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 (out)</i>	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 Sentinel LM 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:

VLSgetClientInfo Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none">• <i>featureName</i> is NULL• <i>version</i> is NULL Both feature name and version cannot be NULL at the same time.
VLS_CALLING_ERROR	<ul style="list-style-type: none">• <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, version and capacity.
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_UNNING	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.

VLSgetClientInfo Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetHandleInfo

Client	Server	Static Library	DLL
✓		✓	✓

Returns information about a client feature.

Syntax

```

LS_STATUS_CODE  VLSgetHandleInfo(
LS_HANDLE       lshandle,
VLSclientInfo   *clientInfo);

```

Argument	Description
<i>lshandle</i>	The handle returned by LSRequest or VLSrequestExt
<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:

VLSgetHandleInfo Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

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.

Syntax

```
LS_STATUS_CODE VLSgetLicInUseFromHandle(  
    LS_HANDLE    lshandle,  
    int          *totalKeysIssued);
```

Argument	Description
<i>lshandle</i>	The handle returned by any Request API call.
<i>totalKeysIssued</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. The information in the handle is *not* updated subsequently. For more information, use VLSgetFeatureInfo.

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

VLSgetLicInUseFromHandle Error Codes

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

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSgetFeatureInfo” on page 100.

Feature Query Functions

The following table summarizes the feature query functions:

Feature Query Functions

Function	Description
<code>VLSgetFeatureInfo</code>	Retrieves feature licensing information from the license server.
<code>VLSgetVersions</code>	Retrieves licensed version information for a feature.
<code>VLSgetFeatureFromHandle</code>	Returns the feature name corresponding to the handle.

Feature Query Functions

Function	Description
VLSgetVersionFrom Handle	Returns the version corresponding to the handle.
VLSgetTimeDriftFrom Handle	Returns the difference in seconds between the estimated current time on the license server and the estimated time on the client.
VLSgetFeatureTime LeftFromHandle	Returns the difference in seconds between the estimated current time on the license server and the estimated feature expiration time on the license server.
VLSgetKeyTimeLeft FromHandle	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 Sentinel LM license server is returned in the following structure.

Syntax

```
typedef struct feature_info_struct {  
    long    structSz  
    char    feature_name[VLS_MAXFEALEN];  
    char    version[VLS_MAXFEALEN];  
    int     lic_type;  
    int     trial_days_count;  
    long    birth_day  
    long    death_day;  
    int     num_licenses;  
    int     total_resv;  
    int     lic_from_resv;  
    int     qlic_from_resv;  
    int     lic_from_free_pool;  
    int     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_MAXCLLOCKLEN];
long         key_life_time;
int          sharing_limit;
int          soft_num_licenses;
int          is_standalone;
int          check_time_tamper;
int          is_additive;
int          isRedundant;
int          majority_rule;
int          num_servers;
int          isCommuter;
int          log_encrypt_level;
int          elan_key_flag;
long         conversion_time;
long         avg_queue_time;
long         queue_length;
int          tot_lic_reqd;
int          isELMEnabled ;
int          commuted_keys;
int          commuter_keys_left;
char         server_locking_info[VLS_MAXSRVLOCKLEN];
int          capacity_flag;
unsigned long capacity;
unsigned long total_resv_capacity;
unsigned long in_use_capacity_from_reserved;
unsigned long in_use_capacity_from_unreserved;
long         commuter_max_checkout_days;
long         grace_period_flag;
long         grace_period_calendar_days;
long         grace_period_elapsed_hours;
long         overdraft_flag;
long         overdraft_hours;
long         overdraft_users;
long         overdraft_users_in_use;
```

```
int    local_request_lockcrit_flag;  
int    local_request_lockcrit_required;  
int    local_request_lockcrit_float;  
int    local_request_lockcrit_min_num;  
int    isGraceLicense;  
int    license_version;  
} VLSfeatureInfo;
```

Member	Description
<i>structSz</i>	Size of VLSfeatureInfo structure.
<i>feature_name</i>	Name of the feature whose information is retrieved. Maximum 24 characters.
<i>version</i>	Feature <i>version</i> . Maximum 11 characters.
<i>lic_type</i>	Type of license either trial or normal.
<i>trial_days_count</i>	Number of trial days.
<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 unreserved licenses issued to queued clients.

Member	Description
<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
<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).

Member	Description
<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. The maximum length of vendor_info string can be 395 characters.
<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: 0x8ef38b91(20#) 0xa4c7188d 0x51f8c94a(10#).
<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.
<i>soft_num_licenses</i>	The soft limit (for alerts) on the number of concurrent users of this feature.
<i>is_standalone</i>	The license type as any of the following: <ul style="list-style-type: none">• VLS_STANDALONE_MODE - for a stand-alone license• VLS_NETWORK_MODE - for a network license• VLS_PERPETUAL_MODE - for a perpetual license

Member	Description
<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>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>commuted_keys</i>	Number of commuter keys that have been checked out.
<i>commuter_keys_left</i>	Number of computer keys left.
<i>server_locking_info</i>	Stores the server locking information.
<i>capacity_flag</i>	The capacity flag can be one of the following constants: <ul style="list-style-type: none"> • VLS_CAPACITY_NONE - for no capacity • VLS_CAPACITY_NON_POOLED - for non-pooled capacity • VLS_CAPACITY_POOLED - for pooled capacity
<i>capacity</i>	Total capacity of the license.
<i>total_resv_capacity</i>	Total reserved capacity.

Member	Description
<i>in_use_capacity_from_reserved</i>	Capacity given to clients from reserved capacity.
<i>in_use_capacity_from_unreserved</i>	Capacity given to clients from unreserved capacity.
<i>commuter_max_checkout_days</i>	The maximum number of days a commuter license can be checked out for.
<i>grace_period_flag</i>	A flag that decides whether grace period for a license will be provided or not. It can be any of the following constants: <ul style="list-style-type: none">• VLS_NO_GRACE_PERIOD - grace period not allowed (default)• VLS_STANDARD_GRACE_PERIOD - grace period allowed
<i>grace_period_calendar_days</i>	The number of days the grace period will last for. It can be a value between 1 to 180 days.
<i>grace_period_elapsed_hours</i>	The number of hours the grace period will last for. It can be a value between 1 to 1440 hours.
<i>overdraft_flag</i>	The flag that decides whether overdraft users are allowed or not. It can be any of the following constants: <ul style="list-style-type: none">• VLS_NO_OVERDRAFT - overdraft users are not allowed (default)• VLS_STANDARD_OVERDRAFT - overdraft users are allowed
<i>overdraft_hours</i>	The number of hours provided to each overdraft user. It can be a value between 1 to 720.
<i>overdraft_users</i>	The number of overdraft users allowed. It can be any of the following values: <ul style="list-style-type: none">• Between 1 to maximum number of concurrent users (not exceeding 30,000)• VLS_INFINITE_KEYS (any number of users)

Member	Description
<i>overdraft_users_in_use</i>	The number of overdraft users currently using the application
<i>local_request_lockcrit_flag</i>	The flag that sets the local license request locking criteria.
<i>local_request_lockcrit_required</i>	The necessary number of locking license criteria that must be met for making the licenses available to a local client.
<i>local_request_lockcrit_float</i>	The desired number of locking license criteria that must be met for making the licenses available to a local client
<i>local_request_lockcrit_min_num</i>	<p>The minimum number of locking license criteria that must be met for making the licenses available to a local client</p> <p>For example, the “required” criteria can be disk ID, the “floating” criteria can be ethernet card and CID key, and the “minimum” criteria can be any 2 of the above.</p>
<i>isGraceLicense</i>	Set to VLS_TRUE only when the grace period is available for the local request

Member	Description
<i>license_version</i>	<p>The Sentinel LM license version. The possible values are:</p> <ul style="list-style-type: none">• VLS_LICENSE_VERSION_800 - For Sentinel LM 8.0.0 licenses• VLS_LICENSE_VERSION_LATEST - For the latest version of Sentinel LM licenses• VLS_LICENSE_VERSION_7301 - For Sentinel LM 7.3.0.1 licenses• VLS_LICENSE_VERSION_730 - For Sentinel LM 7.3.0 licenses• VLS_LICENSE_VERSION_700 - For Sentinel LM 7.0.0 licenses• VLS_LICENSE_VERSION_TOO_OLD - For licenses older than Sentinel LM 7.0.0• VLS_LICENSE_VERSION_NOT_DEFINED - When the license version is not defined

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 the *VLSfeatureInfo* structure being passed to this call before actually calling it.

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 Sentinel LM license server, an appropriate error code is returned. For a complete list of error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Returns

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

VLSgetFeatureInfo Error Codes

Error Code	Description
VLS_CALLING_ERROR	<ul style="list-style-type: none">• featureInfo 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, “Sentinel LM Error and Result Codes,” on page 431.

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 (out)</i>	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:

VLSgetVersions Error Codes

Error Code	Description
VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature, version and capacity.
VLS_APP_UNNAMED	<i>featureName</i> is NULL.

VLSgetVersions Error Codes (Continued)

Error Code	Description
VLS_CALLING_ERROR	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
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, “Sentinel LM Error and Result Codes,” on page 431.

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>	Handle returned by license request API.
<i>buffer (out)</i>	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:

VLSgetFeatureFromHandle Error Codes

Error Code	Description
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<ul style="list-style-type: none"> <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, “Sentinel LM Error and Result Codes,” on page 431.

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 or VLSrequestExt.
<i>buffer (OUT)</i>	Buffer to hold the feature version. Space allocated by caller.
<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:

VLSgetVersionFromHandle Error Codes

Error Code	Description
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	<ul style="list-style-type: none"><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, “Sentinel LM Error and Result Codes,” on page 431.

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 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 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:

VLSgetTimeDriftFromHandle Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

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 or VLSrequestExt.
<i>secondsUntilTheFeature Expires</i>	Caller allocates memory for *out* 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 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.

VLSgetFeatureTimeLeftFromHandle 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: VLSgetFeatureTimeLeftFromHandle 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:

VLSgetFeatureTimeLeftFromHandle Error Codes

Error Code	Description
LS_BADHANDLE	Invalid handle.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature, version and capacity.
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, “Sentinel LM Error and Result Codes,” on page 431.

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 or VLSrequestExt.
<i>secondsUntilTheFeature 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 token (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 token lifetime mentioned in the license is 2 minutes, 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:

VLSgetKeyTimeLeftFromHandle Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

Client Utility Functions

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

Client Utility Functions

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

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 Sentinel LM license servers on the local subnet will respond. The space-separated name list of the responding Sentinel LM 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 left-most 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:

VLSdiscover Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

Examples To get a list of all the Sentinel LM 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);
```

See Also “VLSsetBroadcastInterval” on page 67.

VLsaddFeature

Client	Server	Static Library	DLL
✓		✓	✓

Adds licensing information about a feature.

Syntax

```
LS_STATUS_CODE  VLsaddFeature(  
    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.

Note: The feature is not permanently added to the license server, therefore the feature will not be on the license server when the license server is shut-down and restarted.

Returns

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

VLSaddFeature Error Codes

Error Code	Description
<code>VLS_CALLING_ERROR</code>	Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
<code>LS_NO_SUCCESS</code>	<i>licenseString</i> is NULL.
<code>VLS_ADD_LIC_FAILED</code>	Generic error indicating the feature has not been added.
<code>VLS_BAD_DISTB_CRIT</code>	Invalid distribution criteria.
<code>VLS_CLK_TAMP_FOUND</code>	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.
<code>VLS_NO_SERVER_RUNNING</code>	License server on specified host is not available for processing the license operation requests.
<code>VLS_NO_SERVER_RESPONSE</code>	Communication with license server has timed out.
<code>VLS_HOST_UNKNOWN</code>	Invalid <i>hostName</i> was specified.
<code>VLS_NO_SERVER_FILE</code>	The license server has not been set and is unable to determine which license server to use.
<code>VLS_BAD_SERVER_MESSAGE</code>	Message returned by the license server could not be understood.
<code>LS_NO_NETWORK</code>	Generic error indicating that the network is unavailable in servicing the license operation.
<code>LS_NORESOURCES</code>	An error occurred in attempting to allocate memory needed by this function.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

See Also

“VLSdeleteFeature” on page 119.

VLSaddFeatureToFile

Client	Server	Static Library	DLL
✓		✓	✓

Adds licensing information about a feature.

Syntax

```
LS_STATUS_CODE VLSaddFeatureToFile(  
    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.

Note: The 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:

VLSaddFeatureToFile Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSdeleteFeature” on page 119.

VLSdeleteFeature

Client	Server	Static Library	DLL
✓		✓	✓

Deletes licensing information about a feature.

Syntax

```
LS_STATUS_CODE VLSdeleteFeature(  
    unsigned char *featureName,  
    unsigned char *version,  
    unsigned char *unused1,  
    LS_CHALLENGE *unused2);
```

Argument	Description
<i>featureName</i>	Name of the feature.
<i>version</i>	Version of the feature.
<i>unused2</i>	Unused.
<i>unused3</i>	Unused.

Description Deletes licensing information from the license server’s internal tables, for the given *featureName* and *version*. This call 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:

VLSdeleteFeature Error Codes

Error Code	Description
VLS_APP_UNNAMED	<ul style="list-style-type: none">• <i>featureName</i> is NULL• <i>version</i> is NULL. Both feature name and version cannot be NULL at the same time.
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, version and capacity.
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.

VLSdeleteFeature Error Codes (Continued)

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

See Also “VLSaddFeature” on page 115.

VLSgetLibInfo

Returns information about the Sentinel LM 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];
} LS_LIBVERSION
```

Member	Description
<i>ulInfoCode</i>	Unused.
<i>szVersion</i>	The version of the Sentinel LM 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:

VLSgetLibInfo Error Codes

Codes	Description
LS_NORESOURCES	<i>pInfo</i> is NULL.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSshutDown

Client	Server	Static Library	DLL
✓		✓	✓

Shuts down license server at specified *hostname*.

- Syntax

```
LS_STATUS_CODE VLSshutDown(  
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		Win 95/98	WinNT/2000 (Admin)	UNIX (non-root)	Unix (root)
Client	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/2000 (Admin)	Yes	Yes	Yes	Yes
	UNIX (root)	Yes	Yes	Yes	Yes

Returns

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

VLSshutDown Error Codes

Error Codes	Description
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.

VLSshutDown Error Codes (Continued)

Error Codes	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

VLSwhere

Client	Server	Static Library	DLL
✓		✓	✓

Retrieves the names of the computers on the local subnet (beyond running) the Sentinel LM 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,  
    char              *server_names,
```

```
int broadcastFlag);
```

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

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:

VLSwhere Error Codes

Error Codes	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

Trial License Related Functions

The following table summarizes the trial license related functions:

Trial License Related Functions

Function	Description
VLSgetTrialPeriodLeft	Returns the remaining time left in a trial license.

VLSgetTrialPeriodLeft

Syntax

```
int VLSgetTrialPeriodLeft(
    unsigned char *feature_name,
    unsigned char *version,
    unsigned long *trialperiod,
    unsigned char *unused1);
```

Argument	Description
<i>feature_name</i>	Name of the feature.
<i>version</i>	Version of the feature. Must be unique.
<i>trialperiod</i> (OUT)	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 LS_SUCCESS is returned if stand-alone library is used. Otherwise, it will return the following error codes:

VLSgetTrialPeriodLeft Error Codes

Error Codes	Description
VLS_CALLING_ERROR	<ul style="list-style-type: none">feature_name is NULL.version is NULLtrialperiod is NULL Both feature name and version cannot be NULL at the same time.
VLS_SEVERE_INTERNAL_ERROR	An Irrecoverable internal error has occurred in processing.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing license operation request.
VLS_HOST_UNKNOWN	Invalid hostname was specified.
LS_NO_NETWORK	Generic error indicating that the network is unavailable for servicing the licensing operation.

VLSgetTrialPeriodLeft Error Codes

Error Codes	Description
VLS_NO_SERVER_RESPONSE	Communication with license server has timed out.
VLS_BAD_SERVER_MESSAGE	Message from license server could not be understood.
VLS_INTERNAL_ERROR	An internal error has occurred in processing.
VLS_NO_TRIAL_INFO	No Trial usage info.
VLS_TRIAL_LIC_EXHAUSTED	Trial license expired or trial license usage exhausted.
VLS_TRIAL_INFO_FAILED	Trial usage query failed
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	An error has occurred in decrypting (or decoding) a network message. Probably an incompatible or unknown server, or a version mismatch.

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

Getting License Server Information

Developers sometimes need to know the details about the license servers running on a customer’s computer to see if conflicts are occurring between license servers provided by different developers or to detect a specific license server. The VLSservInfo structure contains the server information. A new API call, VLSgetServInfo, now provides a data structure into which information about a specific license server can be requested or obtained by a client application.

VLSservInfo Struct

```
typedef struct{
    long    structSz;
    int     major_no;
    int     minor_no;
    int     revision_no;
    int     build_no;
    unsigned char locale[VLS_SERV_LOCALE_STR_LEN];
    unsigned char vendor_info[VLS_SERV_VNDINFO_STR_LEN];
    unsigned char platform[VLS_SERV_PLATFORM_STR_LEN];
    unsigned long lock_mask;
    unsigned char unused1[VLS_SERV_UNUSED1_STR_LEN];
    long     unused2;
    VLStimeTamperInfo tmtmpr_info;
    VLSmachineID     machine_id;
} VLSservInfo;
```

Argument	Description
<i>structSz</i>	Size of the structure. Must be set by the user.
<i>major_no</i>	The major number of the server.
<i>mainor_no</i>	The minor number of the server.
<i>revision_no</i>	The revision number of the server.
<i>build_no</i>	The build number of the server.
<i>locale</i>	The locale for which the server was built.
<i>vendor_info</i>	Vendor specified license server identification. This can be customized through VLSsetServInfo API. Default is null string
<i>platform</i>	The platform for which the server was built.
<i>lock_mask</i>	Lock selector used in computing the machine ID of the server machine.

Argument	Description
<i>unused1</i>	Reserved. Uses NULL as the value.
<i>unused2</i>	Reserved. Uses NULL as the value.
<i>tmtmpr_info</i>	Contains the time tampering related information on the server machine.
<i>machine_id</i>	Machine ID structure. To be used in conjunction with <i>lock_mask</i> to obtain the locking code of the server machine. See <i>VLSmachineIDtoLockCode</i> API for details.

Retrieving Information About Time Tampering - VLStimeTamperInfo Struct

The Sentinel LM license server is configured to detect tampering of the system clock. You also have the option of implementing your own functionality to retrieve the time tamper information using the *VLStimeTamperInfo* struct.

```
typedef struct timetampering_info_struct{
    long    structSz;
    time_t  lastTime;
    time_t  currTime;
    long    grace_period;
    int     percentViolationAllowed;
    int     numViolationForError;
    int     numViolationFound;
    int     percentViolationFound;
    unsigned long  clkSetBackTime;
} VLStimeTamperInfo;
```

Argument	Description
structSz	Size of the structure. Must be set by the user.
lastTime	The last known good time when no clock tampering was detected.
currTime	Current time on the server.
grace_period	If Sentinel LM finds the system clock has been set back by less than grace_period seconds, it will not be counted as a violation.
percentViolation Allowed	Percentage of system files that must be found in violation of the grace period before concluding that the system clock has been set back. Used on UNIX systems only.
numViolationFor Error	Number of system files that must be found in violation of the grace period before concluding that the system clock has been set back. Used on UNIX systems only.
numViolationFound	Actual number of system files found in violation of the grace period. Used on UNIX systems only.
percentViolation Found	Percentage of system files found in violation of the grace period. Used on UNIX systems only.
clkSetBackTime	The actual amount of time by which the clock has been set back. This value is zero if no time tampering has been detected.

Retrieving Information About a License Server (VLSgetServInfo)

Returns information regarding the given license server, including version, locale, platform, and the locking information of the computer on which the license server is running. After a successful call, the VLSservInfo data structure will contain the information returned from the license server.

This call will also return the locking information for the computer on which the license server is running. This can be used to generate lock codes as the *echoID.exe* utility does.

Syntax

```
LS_STATUS_CODE VLSgetServInfo(  
    unsigned char *server_name,  
    VLSServInfo *srv_info,  
    unsigned char *unused1,  
    unsigned long *unused2);
```

Argument	Description
<i>server_name</i>	The name of the server you would like to retrieve information from. You can pass this as NULL if you want this API call to pick up the server name if previously set by VLSSetContactServer or LSFORCEHOST. If set to NULL but no server name is found, an error code will be returned. If not set to NULL, this value is independent of the LSHOST, LSFORCEHOST, and VLSSetContactServer values.
<i>srv_info</i>	This points to the VLSServInfo data structure, which is populated with information returned from the server such as platform, locale, build versions, and locking information (see below). You may not set this to NULL.
<i>unused1</i>	Reserved. Uses NULL as the value.
<i>unused2</i>	Reserved. Uses NULL as the value.

Returns

The status code LS_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for the failure. Possible errors returned by this call include VLS_NOT_SUPPORTED.

For a complete list of error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSServInfo Data Structure

The VLSServInfo data structure contains the information returned by VLSgetServInfo:

```
typedef struct{
    long    structSz;
    int     major_no;
    int     minor_no;
    int     revision_no;
    int     build_no;
    unsigned char locale[VLS_SERV_LOCALE_STR_LEN];
    unsigned char vendor_info[VLS_SERV_VNDINFO_STR_LEN];
    unsigned char platform[VLS_SERV_PLATFORM_STR_LEN];
    unsigned long lock_mask;
    unsigned char unused1[VLS_SERV_UNUSED1_STR_LEN];
    long    unused2;
    VLStimeTamperInfo tmtmpr_info;
    VLSmachineID      machine_id;
} VLSservInfo;
```

License Revocation Function

The license revocation option has been provided to allow transfer and revocation of licenses already deployed with customers.

VLSrevokeLicense

Syntax

```
intVLSrevokeLicense(
    unsigned char    *server_name,
    unsigned char    *feature_name,
    unsigned char    *feature_version,
    unsigned long    capacity,
    unsigned char    *password,
    int              *units_to_revoke,
    unsigned long    *capacity_to_revoke,
    unsigned char    *rvk_ticket,
    int              *length_of_rvk_ticket,
    unsigned char    *log_comment,
    unsigned long    *reserved1,
    unsigned char    *reserved2);
```

Argument	Description
<i>server_name</i>	A pointer to the name of the license server where the license to be revoked is currently deployed. An error will be returned if an invalid name or a null value is provided.
<i>feature_name</i>	A pointer to the feature name of the license to be revoked.
<i>feature_version</i>	A pointer to the feature version of the license to be revoked.
<i>capacity</i>	Not supported.
<i>password</i>	A pointer to the password used for authenticating the client.
<i>units_to_revoke</i>	A pointer to the number of tokens to be revoked. An error will be returned by the API if the requested value exceeds the number of licenses available. The number of licenses available for revocation are returned in this parameter.
<i>capacity_to_revoke</i>	A pointer to the capacity units to be revoked. However, revocation of a capacity license is not supported in this release of Sentinel LM. Make the value of this parameter as 0 or pass NULL.
<i>rvk_ticket</i>	A pointer to the license revocation ticket out buffer. The memory needs to be allocated and freed by the programmer. The buffer allocated should be at least 1024 bytes long.

Argument	Description
<i>length_of_rvk_ticket</i>	A pointer to the size of the license revocation ticket.
<i>log_comment</i>	A string that is written by the license manager to the comment field of the usage log file.
<i>reserved1</i>	Reserved for future use. You need to pass null in order to use this function currently.
<i>reserved2</i>	Reserved for future use. You need to pass null in order to use this function currently.

Description This function is used for revoking the hard limit or the number of tokens for a license.

When successful, it returns the license revocation ticket, which can be decoded using the `-lrt` option of the `lsdecode` utility.

The license revocation ticket is encoded using the secret specified at the time of license creation. The same secret is to be provided for decoding the ticket (using the `-secret` option in `lsdecode`).

Note: License with infinite tokens will be revoked completely with the condition that no token should be in use currently. The license will become unusable after revocation.

Returns The status code `LS_SUCCESS` is returned if licenses revoked successfully. Otherwise, it will return the following error codes:

VLSrevokeLicense Error Codes

Error Codes	Description
<code>VLS_REVOKE_ERR_NO_FEATURE</code>	License with the given feature/version does not exist on the license server.
<code>VLS_REVOKE_ERR_CORRUPT_MESSAGE</code>	The message received by the license server was corrupted.
<code>VLS_REVOKE_ERR_OUT_OF_VALID_RANGE</code>	The revocation request was beyond the range.

VLSrevokeLicense Error Codes

Error Codes	Description
VLS_REVOKE_ERR_MD5_PLUGIN_LOAD_FAIL	An error was encountered while loading the MD5 plug-in DLL at the license server.
VLS_REVOKE_ERR_MD5_PLUGIN_EXEC_FAIL	An error was encountered in executing the authentication plug-in.
VLS_REVOKE_ERR_INSUFFICIENT_FEATURE_LICENSES	The feature has less number of total licenses.
VLS_REVOKE_ERR_INSUFFICIENT_DEFAULT_GROUP	The default group does not have sufficient licenses. You can reconfigure your user reservation file.
VLS_REVOKE_ERR_INSUFFICIENT_FREE_IN_DEFAULT	Currently the required number of licenses are not free for revocation in the default group.
VLS_REVOKE_ERR_INVALID_SESSION_ID	An invalid session ID was sent by the client in packet.
VLS_REVOKE_ERR_INVALID_PASSWORD	The password supplied for revocation was invalid.
VLS_REVOKE_ERR_INTERNAL_SERVER	Revocation Failed! - Internal Server Error.
VLS_REVOKE_ERR_INFINITE_GRP_DIST	It is not possible to revoke infinite licenses with group distribution enabled.
VLS_REVOKE_ERR_INFINITE_LIC_IN_USE	All licenses must be free for infinite revocation.
VLS_REVOKE_ERR_INFINITE_LIC_FINITE_REQ	The license has infinite keys. Only infinite revocation request allowed.
VLS_REVOKE_ERR_TICKET_GENERATION	Failed to generate the revocation ticket.
VLS_REVOKE_ERR_CODEGEN_VERSION_UNSUPPORTED	License revocation is not supported on the older versions of the License Code Generator.
VLS_REVOKE_ERR_REDUNDANT_LIC_UNSUPPORTED	License revocation is not supported for redundant licenses.

VLSrevokeLicense Error Codes

Error Codes	Description
VLS_REVOKE_ERR_CAPACITY_LIC_UNSUPPORTED	License revocation is not supported for capacity licenses.
VLS_REVOKE_ERR_UNEXPECTED_AUTH_CHLG_PKT	Unexpected error! The challenge packet was received from the license server.
VLS_REVOKE_ERR_TRIAL_LIC_UNSUPPORTED	License revocation is not supported for trial licenses.

Error Handling

The following table summarizes the error-handling functions:

Error-Handling Functions

Function	Description
VLErrorHandle	Toggles default error handling on or off.
LSGetMessage	Prints error messages corresponding to specified error code.
VLSsetErrorHandler	Registers custom error handlers.
VLSsetUserErrorFile	Configures the display of error messages.

Sentinel LM 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 Sentinel LM, their descriptions, and the default built-in actions, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431. The Sentinel LM 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 <code>VLS_ON</code> . To turn off error handling, use <code>VLS_OFF</code> . Default: <code>VLS_ON</code> .

Description

If the value of *flag* is the constant, `VLS_ON`, error handling is enabled. If *flag* is `VLS_OFF`, error handling is disabled. If called with `VLS_OFF` and `LSGetMessage` is used, the feature name and version are hidden. 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, “Sentinel LM Error and Result Codes,” on page 431.

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 or VLSrequestExt.
value	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*.

Returns

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

LSGetMessage Error Codes

Error Code	Description
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, “Sentinel LM Error and Result Codes,” on page 431.

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);
```

Description

In some situations, the default responses may not be suitable. Therefore, Sentinel LM 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 Sentinel LM 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:

VLSsetErrorHandler Error Codes

Code	Description
VLS_CALLING_ERROR	<ul style="list-style-type: none"> <i>myErrorHandler</i> parameter is NULL <i>LSErrorType</i> is an invalid error type.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

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;  
  
LS_STATUS_CODE VLSsetUserErrorFile(  
    VLS_ERR_FILE    msgFile,  
    char LSFAP      *filePath);
```

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 <i>stderr</i> .

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:

VLSsetUserErrorFile Error Codes

Code	Description
VLS_CALLING_ERROR	Could not open <i>msgFile</i> .

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Tracing Sentinel LM Operation

Client	Server	Static Library	DLL
✓		✓	✓

Enables tracing of the internal operation of the Sentinel LM 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

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/ME, Windows NT, Windows 2000, Windows 2003 or Windows XP.

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 “Functions of the CodeT Struct,” on page 163.
- 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:

- *CodeT* Struct
- Basic functions
- Functions which retrieve or print errors
- Functions which set flags and data fields of *codeT* struct
- License generation functions
- License meter related functions
- License revocation ticket decode function

Example:

```
*****
/*  Copyright (C) 2004 SafeNet, Inc.                      */
/*                      All Rights Reserved                */
/*                      */
/*This Module contains Proprietary Information of */
/*SafeNet, 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 - lscgen32.lib
*/
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);

/* 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() */
```

CodeT Struct

Description Holds the licensing information that is set using VLScgSetXXXX APIs and passes the same to VLScgGenerateLicense API to generate the corresponding license string. Contains the decoded information from the license string as returned by VLScgDecodeLicense API.

Syntax

```
typedef struct {
/* List of flags to be set by external callers: */
int  code_type ;
int  additive ;
int  client_server_lock_mode;
int  holding_crit ;
int  sharing_crit
int  server_locking_crit1[VLScg_MAX_NUM_SERVERS];
int  server_locking_crit2[VLScg_MAX_NUM_SERVERS];
int  client_locking_crit[VLScg_MAX_NUM_NL_CLIENTS];
int  standalone_flag;
int  out_lic_type;
int  clock_tamper_flag;

/* List of data fields to be set by external callers:*/
char feature_name      [VLScg_MAX_CODE_COMP_LEN+1];
char feature_version   [VLScg_MAX_CODE_COMP_LEN+1];
int  birth_day        ;
int  birth_month      ;
int  birth_year       ;
int  death_day        ;
int  death_month      ;
int  death_year       ;
int  num_servers      ;
```

```

char server_lock_info1 [VLScg_MAX_NUM_SERVERS]
    [VLScg_MAX_SERVER_LOCK_INFO_LEN+1];
char server_lock_info2 [VLScg_MAX_NUM_SERVERS]
    [VLScg_MAX_SERVER_LOCK_INFO_LEN+1];
char nl_client_lock_info[VLScg_MAX_NUM_NL_CLIENTS]
    [VLScg_MAX_NL_CLIENT_INFO_LEN+1];
unsigned num_keys[VLScg_MAX_NUM_FEATURES];
unsigned soft_limit ;
unsigned keys_per_node [VLScg_MAX_NUM_NL_CLIENTS];
int  num_subnets      ;
char site_lic_info [VLScg_MAX_NUM_SUBNETS]
    [VLScg_MAX_SUBNET_INFO_LEN+1];
unsigned share_limit;
int  key_life_units ;
unsigned long  key_lifetime  ;
int  key_hold_units ;
unsigned long  key_holdtime  ;
int  num_secrets      ;
char secrets [VLScg_MAX_NUM_SECRETS]
    [VLScg_MAX_SECRET_LEN+1];
char vendor_info      [VLScg_MAX_VENINFOLEN+1];

/* New additions */
int  licType ;
int  trialDaysCount;
int  use_auth_code;
int  numeric_type;

/* for codegen_version >= 7 */
time_t conversion_time;
int isRedundant;
int majority_rule;
int isCommuter;
int commuter_max_checkout_days;
int log_encrypt_level;
int elan_key_flag;

/* Fields for internal use, or unused */
int  vendor_code      ;

```

```
int  version_num      ;
int  licensing_crit   ;
unsigned long meter_value      ;
int  num_features;
int  key_type;

/* Fields for capacity licensing */
int  capacity_flag;
int  capacity_units;
unsigned long capacity;

/* Fields for grace licensing */
int  grace_period_flag;
int  grace_period_calendar_days;
int  grace_period_elapsed_hours;

/* Fields for overdraft licensing */
int  overdraft_flag;
int  overdraft_hours;
int  overdraft_users;
int  overdraft_users_isPercent;

/* Fields for commuter, perpetual, and grace licensing */
int  local_request_lockcrit_flag;
int  local_request_lockcrit_required;
int  local_request_lockcrit_float;
int  local_request_lockcrit_min_num;
} codeT;
```

Member	Description
<i>code_type</i>	Pointer to CodeT struct
<i>additive</i>	License type can be additive or exclusive.
<i>client_server_lock_mode</i>	Locking mode can be: <ul style="list-style-type: none"> • VLScg_FLOATING- Server is locked • VLScg_BOTH_NODE_LOCKED - Clients and server are locked • VLScg_DEMO_MODE - Demo license (no locking) • VLScg_CLIENT_NODE_LOCKED - Only clients are locked
<i>holding_crit</i>	Criterion for held licenses can be: <ul style="list-style-type: none"> • VLScg_HOLD_NONE • VLScg_HOLD_VENDOR • VLScg_HOLD_CODE
<i>sharing_crit</i>	Criterion for sharing of non-capacity licenses, can be: <ul style="list-style-type: none"> • VLScg_NO_SHARING • VLScg_USER_SHARING • VLScg_HOSTNAME_SHARING • VLScg_XDISPLAY_SHARING • VLScg_VENDOR_SHARING Criterion for sharing of capacity licenses, can be: <ul style="list-style-type: none"> • VLScg_NO_TEAM • VLScg_USER_BASED_TEAM • VLScg_HOSTNAME_BASED_TEAM • VLScg_XDISPLAY_BASED_TEAM • VLScg_VENDOR_BASED_TEAM
<i>server_locking_crit1</i>	<ul style="list-style-type: none"> • Server lock selector/criterion (group 1) • Allows 2 hostid's per server
<i>server_locking_crit2</i>	<ul style="list-style-type: none"> • Server lock selector/criterion (group 2) • Allow 2 hostid's per server

Member	Description
<i>client_locking_crit</i>	<ul style="list-style-type: none">• Client lock selector/criterion• Allow 1 hostid per client
<i>standalone_flag</i>	Specifies if the license is stand-alone, network, or perpetual.
<i>out_lic_type</i>	Specifies if the license is <ul style="list-style-type: none">• Encrypted• Expanded readable• Concise readable
<i>clock_tamper_flag</i>	If set then the license does not allow time tampering
<i>feature_name</i>	Name of the feature
<i>feature_version</i>	Version of the feature
<i>birth_day</i>	day of the month (1-31)
<i>birth_month</i>	1 - 12 or JAN - DEC
<i>birth_year</i>	2003 to...; minimum birth year can be 2003.
<i>death_day</i>	max day of the month (1-31)
<i>death_month</i>	1 - 12 or JAN - DEC
<i>death_year</i>	2003 to... ; minimum death year can be 2003.
<i>num_servers</i>	Identifies the number of license servers. 1serverallowed for single server application and maximum 11 servers allowed for redundant server application.
<i>server_lock_info1</i>	Stores information in ascii
<i>server_lock_info2</i>	Stores information in ascii
<i>num_nl_clients</i>	Number of nodelocked clients, maximum of 7 nodelocked clients allowed.
<i>nl_client_lock_info</i>	Stores information in ascii
<i>num_keys</i>	Number of concurrent keys
<i>soft_limit</i>	0 to num_keys

Member	Description
<i>keys_per_node</i>	Number of keys allotted to each client for a network mode license
<i>num_subnets</i>	The number of subnet specifications provided for the site.
<i>site_lic_info</i>	Stores information in binary
<i>share_limit/team-limit</i>	<ul style="list-style-type: none"> • Number of clients/users who can share a single license key. • Used as team limit in case of capacity license.
<i>key_life_units</i>	Determines lifetime least count
<i>long key_lifetime</i>	Absolute value in minutes
<i>key_hold_units</i>	Flag which determines heldtime least count
<i>key_holdtime</i>	Absolute value in minutes
<i>num_secrets</i>	Number of Challenge response secrets
<i>secrets</i>	Stores information in ascii
<i>vendor_info</i>	The vendor-defined information string. The maximum length of vendor_info string can be 395 characters.
<i>licType</i>	Trial or Normal license type
<i>trialDaysCount</i>	Life of trial license.
<i>use_auth_code</i>	For multi-keys or short numeric codes
<i>numeric_type</i>	For short numeric codes <ul style="list-style-type: none"> • 0 - non-numeric • 1 - general short numeric • 2 - general numeric • 10 and above specific type for codegen_version=7
<i>isRedundant</i>	Validates if the license is actually redundant.
<i>majority_rule</i>	Checks whether majority rule is on or off.
<i>isCommuter</i>	Commuter licenses.

Member	Description
<i>commuter_max_checkout_days</i>	The maximum number of days a commuter license can be checked out for. It can be: <ul style="list-style-type: none">• A value between 1 to 1827 days for long licenses.• A value between 1 to 60 days for short licenses.• Zero for unlimited check-out (up to the license expiration)
<i>log_encrypt_level</i>	For encryption level in the license code.
<i>vendor_code</i>	Vendor identification code
<i>version_num</i>	Version number
<i>meter_value</i>	Fields for multi_key for short numeric codegen version >=2
<i>num_features</i>	Number of features in case of multi key
<i>key_type</i>	Single key/Multi key for short numeric only
<i>capacity_flag</i>	Specifies if the license is a capacity or non-capacity license. Values can be: <ul style="list-style-type: none">• VLScg_CAPACITY_NONE• VLScg_CAPACITY_NON_POOLED• VLScg_CAPACITY_POOLED
<i>capacity_units</i>	Flag which determines capacity least count
<i>long_capacity</i>	The capacity of this license.
<i>grace_period_flag</i>	A flag that decides whether grace period for a license will be provided or not. It can be any of the following constants: <ul style="list-style-type: none">• VLScg_NO_GRACE_PERIOD - grace period not allowed (default)• VLScg_STANDARD_GRACE_PERIOD - grace period allowed
<i>grace_period_calendar_days</i>	The number of days the grace period will last for. It can be a value between 1 to 180 days. The default value is 2 days.

Member	Description
<i>grace_period_elapsed_hours</i>	The number of hours the grace period will last for. It can be a value between 1 to 1440 hours. The default value is 20 hours.
<i>overdraft_flag</i>	The flag that decides whether overdraft users are allowed or not. It can be: <ul style="list-style-type: none">• VLScg_NO_OVERDRAFT (default)• VLScg_STANDARD_OVERDRAFT (overdraft users are allowed).
<i>overdraft_hours</i>	The number of hours provided to each overdraft user. It can be a value between 1 to 720 hours. The default value is 2 hours.
<i>overdraft_users</i>	The number of overdraft users allowed. It can be a value between 1 to maximum concurrent users (not exceeding 30,000) or VLScg_INFINITE_KEYS (any number of users). The default value is 10 users.
<i>overdraft_users_isPercent</i>	The number of overdraft users as a percentage of the maximum concurrent users.

Member	Description
<i>local_request_lockcrit_flag</i>	The flag that sets the local license request locking criteria as: <ul style="list-style-type: none">• <code>VLScg_LOCAL_REQUEST_LOCKCRIT_USEDEFAULT</code> - Sets <i>local_request_lockcrit_required</i> to disk ID (<code>VLS_LOCK_DISK_ID</code>) <i>local_request_lockcrit_float</i> to 0, and <i>local_request_lockcrit_min_num</i> to 1. This is the default setting.• <code>VLS_LOCAL_REQUEST_LOCKCRIT_DEFINED</code> - Sets <i>local_request_lockcrit_required</i> to a user-defined value.
<i>local_request_lockcrit_required</i>	The necessary number of locking criteria that must be met for making the licenses available to a local client
<i>local_request_lockcrit_float</i>	The desired number of locking criteria that must be met for making the licenses available to a local client
<i>local_request_lockcrit_min_num</i>	The minimum number of locking criteria that must be met for making the licenses available to a local client. For example, the “required” locking criteria can be disk ID, the “floating” criteria can be ethernet card and CID key, and the “minimum” criteria can be any two of the above.

Basic Functions

The following table summarizes the basic functions for this library:

Basic Functions

Function	Description
<code>VLScgInitialize</code>	Initializes the handle.
<code>VLScgCleanup</code>	Destroys the created handle.
<code>VLScgReset</code>	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:

VLScgInitialize Error Codes

Error Code	Description
VLScg_MAX_LIMIT_CROSSED	No more handles left.
VLScg_BAD_HANDLE	Call VLScgCleanup to free the resources associated with invalid handle.
VLScg_LICMETER_NOT_SUPPORTED	Your Sentinel LM License Meter is not supported.

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

See Also

“VLScgCleanup” on page 157

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 Codes for License Generation Functions,” on page 453.

VLScgReset

- Syntax**
- ```
int VLScgReset(
 VLScg_HANDLE iHandleP,
 codeT *codeP);
```

| Argument        | Description                           |
|-----------------|---------------------------------------|
| <i>iHandleP</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 Codes for License Generation Functions,” on page 453.

**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:

**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> (OUT) | 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*. You can have only one int memory for this API. Hence the code would be:

```
int errNo;
VLScg_HANDLE handle;
VLScgGetNumErrors(handle, &errNo);
```

### Returns

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

**VLScgGetNumErrors Error Codes**

| Error Code         | Description                    |
|--------------------|--------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

### 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> .  |

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:

VLScgGetErrorLength Error Codes

| Error Code         | Description                    |
|--------------------|--------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

### 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><br>(OUT) | 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:

#### VLScgGetErrorMessage Error Codes

| Error Code         | Description                    |
|--------------------|--------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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:

VLScgPrintError Error Codes

| Error Code         | Description                    |
|--------------------|--------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## Functions for Setting the Fields in CodeT Struct

The following table summarizes the functions used to set flags and data fields of the *codeT* 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.

---

## Functions of the CodeT Struct

| Function                                                                                               | Description                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLScgSetCodeLength                                                                                     | Sets the license code length.                                                                                                                                                                                      |
| VLScgAllowFeatureName<br>VLScgSetFeatureName                                                           | Sets the name of the feature to be licensed.                                                                                                                                                                       |
| VLScgAllowFeatureVersion<br>VLScgSetFeatureVersion                                                     | Sets the version number to be licensed.                                                                                                                                                                            |
| VLScgAllowLicenseType<br>VLScgSetLicenseType                                                           | Controls the license type.                                                                                                                                                                                         |
| VLScgAllowTrialLicFeature<br>VLScgSetTrialDaysCount                                                    | Sets the number of trial days.                                                                                                                                                                                     |
| VLScgAllowAdditive<br>VLScgSetAdditive                                                                 | Sets the license to exclusion or additive.                                                                                                                                                                         |
| VLScgAllowKeyLifeUnits<br>VLScgSetKeyLifetimeUnits                                                     | Sets unit of time used to specify time between license renewals.                                                                                                                                                   |
| VLScgAllowStandAloneFlag<br>VLScgAllowNetworkFlag<br>VLScgSetStandAloneFlag<br>VLScgAllowPerpetualFlag | Sets whether the license will be stand-alone, network, or perpetual.                                                                                                                                               |
| VLScgAllowLogEncryptLevel<br>VLScgSetLogEncryptLevel                                                   | Controls the network license encryption level for the license server's usage log file.                                                                                                                             |
| VLScgAllowSharedLic/<br>VLScgAllowTeamCriteria<br>VLScgSetSharedLicType/<br>VLScgSetTeamCriteria       | <ul style="list-style-type: none"> <li>• Enables shared licenses and sets sharing criteria for non-capacity license.</li> <li>• Enables team licenses and sets team criteria for capacity license.</li> </ul>      |
| VLScgAllowShareLimit/<br>VLScgAllowTeamLimit<br>VLScgSetShareLimit/<br>VLScgSetTeamLimit               | <ul style="list-style-type: none"> <li>• Sets the number of users that can share a non-capacity license.</li> <li>• Sets the number of team members that can share a token in case of capacity license.</li> </ul> |
| VLScgAllowCommuterLicense<br>VLScgSetCommuterLicense                                                   | Enables commuter licenses to be checked out.                                                                                                                                                                       |

**Functions of the CodeT Struct (Continued)**

| Function                                                | Description                                                                                                           |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| VLScgAllowNumKeys<br>VLScgSetNumKeys                    | Sets the number of concurrent licenses allowed.                                                                       |
| VLScgAllowLockModeQuery<br>VLScgSetClientServerLockMode | Sets locking mode for the license server computer. Installs client server lock mode in codeP.                         |
| VLScgAllowRedundantFlag<br>VLScgSetRedundantFlag        | Controls whether the license will be used with redundant license servers.                                             |
| VLScgAllowMajorityRuleFlag<br>VLScgSetMajorityRuleFlag  | Controls whether the majority of redundant license servers must be running.                                           |
| VLScgAllowMultipleServerInfo<br>VLScgSetNumServers      | Fields for information on various license servers.                                                                    |
| VLScgAllowServerLockInfo<br>VLScgSetServerLockInfo1     | Sets license server primary locking code. Installs license server lock code in primary lock.                          |
| VLScgSetServerLock<br>Mechanism1                        | Sets license server primary fingerprint criteria. Installs license server's fingerprint criteria in primary lock.     |
| VLScgSetServerLock<br>Mechanism2                        | Sets license server secondary fingerprint criteria. Installs license server's fingerprint criteria in secondary lock. |
| VLScgSetServerLockInfo2                                 | Sets license server secondary locking code. Installs server lock code in secondary lock.                              |
| VLScgAllowLockMechanism<br>VLScgSetClientLockMechanism  | Sets client's fingerprint criteria.                                                                                   |
| VLScgAllowClientLockInfo<br>VLScgSetClientLockInfo      | Sets the client locking code.                                                                                         |
| VLScgSetNumClients                                      | Sets the number of client locking codes to be specified.                                                              |
| VLScgAllowClockTamperFlag<br>VLScgSetClockTamperFlag    | Controls action on detection of clock being set back on the machine.                                                  |
| VLScgAllowOutLicType<br>VLScgSetOutLicType              | Sets the license output format.                                                                                       |

### Functions of the CodeT Struct (Continued)

| Function                                                       | Description                                                                                                                                  |
|----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| VLScgSetLicType                                                | Sets the license type.                                                                                                                       |
| VLScgAllowHeldLic<br>VLScgSetHoldingCrit                       | Enables/disables license hold time and determines where that hold time is specified.                                                         |
| VLScgAllowCodegenVersion<br>VLScgSetCodegenVersion             | Sets the version of license codes to generate. Checks if the current license code setting allows multiple features.                          |
| VLScgAllowMultiKey<br>VLScgSetKeyType                          | Controls whether a license will be single or multi-feature.                                                                                  |
| VLScgAllowSecrets<br>VLScgSetSecrets<br>VLScgSetNumSecrets     | Sets the value of the specified challenge-response secrets.<br>Sets the total number of secrets for the challenge-response.                  |
| VLScgAllowVendorInfo<br>VLScgSetVendorInfo                     | Sets vendor-defined information in the license.                                                                                              |
| VLScgAllowKeysPerNode<br>VLScgSetKeysPerNode                   | Sets the number of license tokens per node for the specified number of clients.                                                              |
| VLScgAllowSiteLic<br>VLScgSetSiteLicInfo<br>VLScgSetNumSubnets | Sets address of subnets licensed application will be restricted to.<br>Sets the number of subnets the licensed application is restricted to. |
| VLScgAllowNumFeatures<br>VLScgSetNumFeatures                   | Sets the number of features.                                                                                                                 |
| VLScgAllowSoftLimit<br>VLScgSetSoftLimit                       | Sets soft limit number.                                                                                                                      |
| VLScgAllowKeyHoldUnits<br>VLScgSetKeyHoldtimeUnits             | Sets units of time to be used to specify license hold time.                                                                                  |
| VLScgAllowKeyLifetime<br>VLScgSetKeyLifetime                   | Sets time between license renewals.                                                                                                          |
| VLScgAllowKeyHoldtime<br>VLScgSetKeyHoldtime                   | Sets the time a license will be held.                                                                                                        |

**Functions of the CodeT Struct (Continued)**

| Function                                                                                                                             | Description                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLScgAllowLicBirth<br>VLScgSetLicBirthMonth<br>VLScgSetLicBirthDay<br>VLScgSetLicBirthYear                                           | Sets the month of the license start date (the month should be specified in the range of 0-11). Sets the day of the license start date.<br>Sets the year of the license start date. |
| VLScgAllowLicExpiration<br>VLScgSetLicExpirationMonth<br>VLScgSetLicExpirationDay<br>VLScgSetLicExpirationYear                       | Sets month license expires. The months should be specified in the range of 0-11.<br>Sets day month the license expires.<br>Sets the year the license expires.                      |
| VLScgSetNumericType                                                                                                                  | Sets the value of numeric type.                                                                                                                                                    |
| VLScgSetLoadSWLicFile                                                                                                                | Sets and loads the software license file ( <i>lscgen.lic</i> ).                                                                                                                    |
| VLScgAllowGracePeriodFlag<br>VLScgSetGracePeriodFlag<br>VLScgAllowGracePeriod<br>VLScgSetGracePeriodDays<br>VLScgSetGracePeriodHours | Set the grace period for the commuter license.                                                                                                                                     |
| VLScgAllowOverdraftFlag<br>VLScgSetOverdraftFlag<br>VLScgAllowOverdraft<br>VLScgSetOverdraftHours<br>VLScgSetOverdraftUsers          | Sets the overdraft period.                                                                                                                                                         |

## VLScgSetCodeLength

**Syntax**

```
int VLScgSetCodeLength(
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>    | <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"> <li>VLScg_SHORT_CODE_STRING = "0"</li> <li>VLScg_LONG_CODE_STRING = "1"</li> <li>VLScg_SHORT_NUMERIC_CODE = "2"</li> </ul> |

**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:

**VLScgSetCodeLength Error Codes**

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

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**VLScgAllowFeatureName**

**Syntax**

```
int VLScgAllowFeatureName(
 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 `VLScgAllowXXX` 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.

---

**Note:** 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:

#### VLScgSetFeatureName Error Codes

| Error Code              | Description                                                               |
|-------------------------|---------------------------------------------------------------------------|
| 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 | Returned if the length of string passed exceeds the maximum length of 24. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX 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.

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

#### VLScgSetFeatureVersion Error Codes

| Error Code                           | Description                                                                                     |
|--------------------------------------|-------------------------------------------------------------------------------------------------|
| <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. Maximum length of the version is 11 characters. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowLicenseType

### Syntax

```
int VLScgAllowLicenseType(
 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 `VLScgAllowXXX` 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(
 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>    | <i>Flag</i> is used to set the <i>code_type</i> member of <i>codeT</i> struct.<br>The values are: <ul style="list-style-type: none"><li>• VLScg_NORMAL_LIC_STRING - Non-trial license = "0"</li><li>• VLScg_TRIAL_LIC_STRING - Trial license = "1"</li></ul> |

**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:

**VLScgSetLicenseType Error Codes**

| Error Code                | Description                                       |
|---------------------------|---------------------------------------------------|
| 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<br>VLScg_NORMAL_LIC_STRING |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 VLScgAllowXXX 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX 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

**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"><li>• VLScg_ADDITIVE = "0"</li><li>• VLScg_EXCLUSIVE = "1"</li></ul> |

**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.

---

**Note:** An additive license can't be added on an exclusive license.

---

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

#### VLScgSetAdditive Error Codes

| Error Code                | Description                                |
|---------------------------|--------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX 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.

## 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. See “VLScgSetKeyLifetimeUnits” on page 229. |

### 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:

**VLScgSetKeyLifetime Error Codes**

| Error Code                             | Description                               |
|----------------------------------------|-------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If information is a non-negative integer. |
| <code>VLScg_NOT_MULTIPLE</code>        | If value is not a correct multiple.       |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds 3.                       |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is less than or equal to 0.      |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 `VLScgAllowXXX` 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(
 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 VLScgAllowXXX 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.

## VLScgAllowPerpetualFlag

### Syntax

```
int VLScgAllowPerpetualFlag(
 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 VLScgAllowXXX 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                                                                                                                                                                                                            |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iHandle  | The instance handle for this library.                                                                                                                                                                                  |
| codeP    | The pointer to the codeT struct.                                                                                                                                                                                       |
| flag     | Flag values are used to set the standalone_flag of codeT struct. Legal values are: <ul style="list-style-type: none"><li>VLScg_NETWORK_STRING</li><li>VLScg_STANDALONE_STRING</li><li>VLScg_PERPETUAL_STRING</li></ul> |

**Description**

Sets whether license will be for stand-alone or network computer.

Stand-alone and network licenses cannot be used interchangeably.

**Returns**

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

### VLScgSetStandAloneFlag Error Codes

| Error Code                | Description                                      |
|---------------------------|--------------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## VLScgAllowLogEncryptLevel

### Syntax

```
int VLScgAllowLogEncryptLevel(
 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 VLScgAllowXXX 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(
 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>    | Allowed value are: <ul style="list-style-type: none"><li>• "0"</li><li>• "1"</li><li>• "2"</li><li>• "3"</li><li>• "4"</li></ul> |

### 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:

**VLScgSetLogEncryptLevel Error Codes**

| Error Code                | Description                                  |
|---------------------------|----------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not a decimal number.            |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds VLScg_MAX_ENCRYPTION_LEVEL. |
| 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 Codes for License Generation Functions,” on page 453.

**VLScgAllowSharedLic/ VLSAllowTeamCriteria**

**Syntax** In case of non-capacity license:

```
int VLScgAllowSharedLic(
 VLScg_HANDLE iHandle,
 codeT *codeP);
```

In case of capacity license:

```
int VLScgAllowTeamCriteria(
 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. |

---

**Note:** We recommend you use VLScgAllowSharedLic for non-capacity license and VLScgAllowTeamCriteria for capacity license.

---

**Returns** The VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX should be called. If VLScgAllowXXX returns 1 then the

corresponding VLScgSetXXX function can be called. Otherwise, it will return 0 as false.

## VLScgSetSharedLicType/ VLScgSetTeamCriteria

### Syntax

In case of non-capacity license:

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

In case of capacity license:

```
int VLScgSetTeamCriteria(
 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>    | This <i>flag</i> enables shared licenses and specifies the sharing criteria. Legal values are: <ul style="list-style-type: none"> <li>VLScg_NO_SHARING_STRING = "0"</li> <li>VLScg_USER_SHARING_STRING = "1"</li> <li>VLScg_HOSTNAME_SHARING_STRING = "2"</li> <li>VLScg_XDISPLAY_SHARING_STRING = "3"</li> <li>VLScg_VENDOR_SHARING_STRING = "4" - Vendor defined / customized. Need to customize the client library for this.</li> </ul> |

### 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.

**Tip:** We recommend you use `VLScgSetSharedLicType` for non-capacity license and `VLScgSetTeamCriteria` for capacity license.

**Returns**

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

**VLScgSetSharedLicType/ VLScgSetTeamCriteria Error Codes**

| Error Code                | Description                                            |
|---------------------------|--------------------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not numeric.                               |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds<br>VLScg_VENDOR_SHARING_STRING.       |
| VLScg_LESS_THAN_MIN_VALUE | If the value is lower than<br>VLScg_NO_SHARING_STRING. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**VLScgAllowShareLimit/ VLScgAllowTeamLimit**

**Syntax**

In case of non-capacity license:

```
int VLScgAllowShareLimit(
 VLScg_HANDLE iHandle,
 codeT *codeP);
```

In case of capacity license:

```
int VLScgAllowTeamLimit(
 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. |

---

**Tip:** We recommend you use `VLScgAllowShareLimit` for non-capacity license and `VLScgAllowTeamLimit` for capacity license.

---

**Returns**

The `VLScgAllowXXX` 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/VLScgSetTeamLimit

**Syntax**

In case of non-capacity license:

```
int VLScgSetShareLimit(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *decimalNUM);
```

In case of capacity license:

```
int VLScgSetTeamLimit(
 VLScg_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.

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 non-capacity license or when the team limit has been reached for a capacity license.

**Tip:** We recommend you use `VLScgSetShareLimit` for non-capacity license and `VLScgSetTeamLimit` for capacity license.

---

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

**VLScgSetShareLimit/ VLScgSetTeamLimit Error Codes**

| Error Code                             | Description                         |
|----------------------------------------|-------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If value is not numeric.            |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds maximum.           |
| <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 Codes for License Generation Functions,” on page 453.

**VLScgAllowCommuterLicense**

**Syntax**

```
int VLScgAllowCommuterLicense(
 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 `VLScgAllowXXX` 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(
 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"> <li>VLScg_NOT_ISSUE_COMMUTER_CODES_STRING = "0"</li> <li>VLScg_ISSUE_COMMUTER_LICENSE_CODE_STRING = "1"</li> </ul> |

### Description

Enables commuter licenses.

This function is used to generate license use authorizations for traveling clients. Commuter licensing allows end users to “check out” an authorization from a network served license group and “check it in” when they are done using the application.

### Returns

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

#### VLScgSetCommuterLicense Error Codes

| Error Code                | Description                                                      |
|---------------------------|------------------------------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not numeric.                                         |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds<br>VLScg_ISSUE_COMMUTER_CODES_STRING            |
| VLScg_LESS_THAN_MIN_VALUE | If value is lower than<br>VLScg_NOT_ISSUE_COMMUTER_CODES_STRING. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowCommuterMaxCheckoutDays

**Syntax**

```
int VLScgAllowCommuterMaxCheckoutDays(
 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX should be called. If VLScgAllowXXX returns 1 then the corresponding VLScgSetXXX function can be called. Otherwise, it will return 0 as false.

## VLScgSetCommuterMaxCheckoutDays

**Syntax**

```
int VLScgSetCommuterMaxCheckoutDays(
 VLScg_HANDLE iHandle,
 codeT *codeP),
 char* dayStr);
```

| 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> |                                         |

**Description**

Sets the maximum check-out days allowed for a commuter license.

**Returns**

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

## 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 VLScgAllowXXX 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,
 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>    | 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.)

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

**VLScgSetNumKeys Error Codes**

| Error Code                             | Description                                                                                                                                               |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If value is not a non-negative integer.                                                                                                                   |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value of <i>info</i> exceeds maximum number of license tokens allowed. Maximum value for long codes is 32766 and maximum value for short codes is 254. |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value of <i>info</i> is less than 0.                                                                                                                   |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 `VLScgAllowXXX` 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,
 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>The <i>flag</i> values are:</p> <ul style="list-style-type: none"> <li>VLScg_FLOATING_STRING - License server is locked = "0"</li> <li>VLScg_BOTH_NODE_LOCKED_STRING - Clients and license server are locked = "1"</li> <li>VLScg_DEMO_MODE_STRING - Trial license (no locking) = "2"</li> <li>VLScg_CLIENT_NODE_LOCKED_STRING - Only clients are locked = "3"</li> </ul> |

### 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:

### VLScgSetClientServerLockMode Error Codes

| Error Code                | Description                                  |
|---------------------------|----------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not numeric.                     |
| VLScg_EXCEEDS_MAX_VALUE   | If the value of flag exceeds maximum of 3.   |
| VLScg_LESS_THAN_MIN_VALUE | If the value is lower than the minimum of 0. |

For a complete list of the error codes, see Appendix D, "Error Codes for License Generation Functions," on page 453.

## VLScgAllowRedundantFlag

**Syntax**

```
int VLScgAllowRedundantFlag(
 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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(
 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"><li>VLScg_NON_REDUNDANT_CODE_STRING - Non-redundant license = "0"</li><li>VLScg_REDUNDANT_CODE_STRING - Redundant license = "1"</li></ul> |

**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 Sentinel LM 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:

#### VLScgSetRedundantFlag Error Codes

| Error Code                             | Description                                                          |
|----------------------------------------|----------------------------------------------------------------------|
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds <code>VLScg_REDUNDANT_CODE_STRING</code> .          |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is less than <code>VLScg_NON_REDUNDANT_CODE_STRING</code> . |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowMajorityRuleFlag

### Syntax

```
int VLScgAllowMajorityRuleFlag(
 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 `VLScgAllowXXX` 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(
 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"><li>VLScg_MAJORITY_RULE_FOLLOWS_STRING - Sets the <i>majority_rule_flag</i> = "1"</li><li>VLScg_MAJORITY_RULE_NOT_FOLLOWS_STRING - Unsets the <i>majority_rule_flag</i> = "0"</li></ul> |

**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:

VLScgSetMajorityRuleFlag Error Flag

| Error Code                | Description                                                    |
|---------------------------|----------------------------------------------------------------|
| VLScg_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_FOLLOWS_STRING. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.



## VLScgAllowMultipleServerInfo

### Syntax

```
int VLScgAllowMultipleServerInfo(
 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 VLScgAllowXXX 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.

## VLScgSetNumServers

### Syntax

```
int VLScgSetNumServers(
 VLScg_HANDLE iHandle ,
 codeT *codeP ,
 char *str);
```

| Argument       | Description                             |
|----------------|-----------------------------------------|
| <i>iHandle</i> | The instance handle for this library.   |
| <i>codeP</i>   | The pointer to the <i>codeT</i> struct. |
| <i>str</i>     | Number of servers                       |

### Description

This API sets the number of redundant servers. It can be called for long codes only also the number of servers should be odd. This sets the number of servers for redundancy.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLScgSetNumServers Error Codes**

| Error Code                | Description                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not numeric                                                                              |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds the maximum number of license servers. Maximum number of license servers can be 11. |
| VLScg_LESS_THAN_MIN_VALUE | If the value is less than minimum number of license servers that is equal to or less than 0.         |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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 0 to <i>num_servers-1</i> where <i>num_servers</i> is set using <i>VLScgSetNumServers</i> . <ul style="list-style-type: none"> <li><i>server_lock_info1</i> is an array of 11 elements storing the primary locking codes for 11 servers.</li> </ul> |

### 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:

#### VLScgSetServerLockInfo1 Error Codes

| Error Code                       | Description                                                                                                   |
|----------------------------------|---------------------------------------------------------------------------------------------------------------|
| <i>VLScg_INVALID_HEX_TYPE</i>    | If value is not in hexadecimal format.                                                                        |
| <i>VLScg_EXCEEDS_MAX_VALUE</i>   | If value exceeds the maximum number of license servers. The value set using the API <i>VLScgSetNumServers</i> |
| <i>VLScg_LESS_THAN_MIN_VALUE</i> | If the value is less than minimum number of license servers.                                                  |

For a complete list of the error codes, see Appendix D, "Error Codes for License Generation Functions," on page 453.

## 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>    | Position in array which is storing the primary locking criteria of the 11 servers. Value 0 to num_servers where num_servers is set using VLScgSetNumServers. |

**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:

### VLScgSetServerLockMechanism1 Error Codes

| Error Code                | Description                                                                              |
|---------------------------|------------------------------------------------------------------------------------------|
| VLScg_INVALID_HEX_TYPE    | If criterion is not in hexadecimal format.                                               |
| VLScg_EXCEEDS_MAX_VALUE   | If <i>number of server</i> is too large. The value set using the API VLScgSetNumServers. |
| VLScg_LESS_THAN_MIN_VALUE | If the <i>number of server</i> is lower than minimum.                                    |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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>    | Position in array which is storing the secondary locking criteria of the 11 servers. Its value should also vary from 0 - <i>num_servers</i> where <i>num_servers</i> is set using VLScgSetNumServers. |

### 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 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:

**VLScgSetServerLockMechanism2 Error Codes**

| Error Code                | Description                                                                              |
|---------------------------|------------------------------------------------------------------------------------------|
| VLScg_INVALID_HEX_TYPE    | If criterion is not in hexadecimal format.                                               |
| VLScg_EXCEEDS_MAX_VALUE   | If <i>number of server</i> is too large. The value set using the API VLScgSetNumServers. |
| VLScg_LESS_THAN_MIN_VALUE | If the <i>number of server</i> is lower than minimum.                                    |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 0 to <i>num_servers-1</i> where <i>num_servers</i> is set using VLScgSetNumServers. <ul style="list-style-type: none"><li><i>server_lock_info2</i> is an array of 11 elements storing the secondary locking codes for 11 servers.</li></ul> |

**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:

#### VLScgSetServerLockInfo2 Error Codes

| Error Code                             | Description                                                                          |
|----------------------------------------|--------------------------------------------------------------------------------------|
| <code>VLScg_INVALID_HEX_TYPE</code>    | If value is not in hexadecimal format.                                               |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value is too large. The value set using the API <code>VLScgSetNumServers</code> . |
| <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 Codes for License Generation Functions,” on page 453.

## 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 `VLScgAllowXXX` 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> | <i>client_num</i> is the position in the array storing the locking mechanisms for the clients. The value will vary from 0 to <i>num_nl_clients</i> where <i>num_nl_clients</i> is the number of clients set using <i>VLScgSetNumClients</i> . |

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:

VLScgSetClientLockMechanism Error Codes

| Error Code                       | Description                                                                    |
|----------------------------------|--------------------------------------------------------------------------------|
| <i>VLScg_INVALID_HEX_TYPE</i>    | If value is not in hexadecimal format.                                         |
| <i>VLScg_EXCEEDS_MAX_VALUE</i>   | If value is too large. The value set using the API <i>VLScgSetNumClients</i> . |
| <i>VLScg_LESS_THAN_MIN_VALUE</i> | If the value is lower than minimum.                                            |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.



## 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 VLScgAllowXXX 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.                                                                                                                                                                                                                                           |
| <i>num</i>      | Number of clients: should be from 0 to maximum number of clients specified -1.<br><i>num</i> is the position in the array storing the locking codes for the clients. The value will vary from 0 to <i>num_nl_clients</i> where <i>num_nl_clients</i> is the number of clients set using VLScgSetNumClients. |

**Description** Sets the client locking code.

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

**VLScgSetClientLockInfo Error Codes**

| Error Code                             | Description                                                                         |
|----------------------------------------|-------------------------------------------------------------------------------------|
| <code>VLScg_INVALID_HEX_TYPE</code>    | If value is not in hexadecimal format.                                              |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If number is greater than <i>num_nl_clients</i> - 1. Number of node locked clients. |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If number is less than 0.                                                           |
| <code>VLScg_INVALID_IP_TYPE</code>     | If value is not in dot format.                                                      |
| <code>VLScg_UNKNOWN_LOCK</code>        | If the locking criteria is unknown.                                                 |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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:

#### VLScgSetNumClients Error Codes

| Error Code                             | Description                                   |
|----------------------------------------|-----------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If input is not a non-negative integer.       |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds maximum number of 7 clients. |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is less than 1.                      |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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 `VLScgAllowXXX` 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"><li>VLScg_NO_CHECK_TAMPER_STRING - Do not check clock tamper = "0"</li><li>VLScg_CHECK_TAMPER_STRING - Check clock tamper = "1"</li></ul> |

**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:

**VLScgSetClockTamperFlag Error Codes**

| Error Code             | Description                           |
|------------------------|---------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX 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"><li>VLScg_ENCRYPTED_STRING = “0”</li><li>VLScg_EXPANDED_READABLE_STRING = “1”</li><li>VLScg_CONCISE_READABLE_STRING = “2”</li></ul> |

### 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:

**VLScgSetOutLicType Error Codes**

| Error Code                          | Description                           |
|-------------------------------------|---------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code> | If value is not a decimal number.     |
| <code>VLScg_INVALID_RANGE</code>    | If value is not in the range allowed. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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"><li>VLScg_TRIAL_LIC_STRING = "1"</li><li>VLScg_NORMAL_LIC_STRING = "0"</li></ul> |

**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:

### VLScgSetLicType Error Codes

| Code                   | Description                   |
|------------------------|-------------------------------|
| VLScg_INVALID_LIC_TYPE | If license type is not valid. |

For a complete list of the error codes, see Appendix D, "Error Codes for License Generation Functions," on page 453.

## 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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"><li>• VLScg_HOLD_NONE_STRING = "0" - Held licenses not allowed.</li><li>• VLScg_HOLD_VENDOR_STRING = "1" - Client API specifies hold time.</li><li>• VLScg_HOLD_CODE_STRING = "2" - License code specifies hold time.</li></ul> |

**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:

#### VLScgSetHoldingCrit Error Codes

| Error Code                | Description                                        |
|---------------------------|----------------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## VLScgAllowCodegenVersion

### Syntax

```
int VLScgAllowCodegenVersion(
 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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(
 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 possible values for <i>version_num flag</i> . |

**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:

### VLScgSetCodegenVersion Error Codes

| Error Code              | Description                              |
|-------------------------|------------------------------------------|
| VLScg_INVALID_INT_TYPE  | If value is not numeric.                 |
| VLScg_EXCEEDS_MAX_VALUE | If value exceeds<br>MAX_CODEGEN_VERSION. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowCapacityLic

**Syntax**

```
int VLScgAllowCapacityLic(
 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. |

**Description** Allows the application to check if capacity licensing is allowed or not. For details on capacity licensing, see the *Sentinel LM Developer's Guide*.

**Returns** It will return the following return status:

#### VLScgAllowCapacityLic Return Status

| Return Value | Description                        |
|--------------|------------------------------------|
| 0            | Capacity licensing is not allowed. |
| 1            | Capacity licensing is allowed.     |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgSetCapacityFlag

### Syntax

```
int VLScgSetCapacityFlag(
 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 flag is used to set the capacity_flag of codeT struct. Legal values are- <ul style="list-style-type: none"> <li>VLScg_CAPACITY_NONE_STRING</li> <li>VLScg_CAPACITY_NON_POOLED_STRING</li> <li>VLScg_CAPACITY_POOLED_STRING</li> </ul> |

**Description** Specifies whether the license is a capacity license or not. Also sets the appropriate fields of *codeT* structure to make sure that it is:

- A normal license and not a trial license
- A network license and not a stand-alone license
- Not a held license
- Not a redundant license

- Not a commuter license
- License code format is “Encrypted” only.

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

**VLScgSetCapacityFlag Error Codes**

| Error Code                             | Description                                             |
|----------------------------------------|---------------------------------------------------------|
| <code>VLScg_SUCCESS</code>             | Success                                                 |
| <code>VLScg_INVALID_INT_TYPE</code>    | If value is not numeric                                 |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds <code>VLScg_CAPACITY_POOLED</code>     |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is lower than <code>VLScg_CAPACITY_NONE</code> |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

### VLScgAllowCapacity

**Syntax**

```
int VLScgAllowCapacity(
 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. |

**Description** Allows the application to check whether it is a capacity license or not.

**Returns** It will return the following return status:

**VLScgAllowCapacity Return Status**

| Return Value | Description                   |
|--------------|-------------------------------|
| 0            | It is a non-capacity license. |
| 1            | It is a capacity license.     |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgSetCapacityUnits

### Syntax

```
int VLScgSetCapacityUnits(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *units);
```

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i> | The instance handle for this library.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>codeP</i>   | The pointer to the <i>codeT</i> struct.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>units</i>   | Capacity specification units from 0 to 4. The values are: <ul style="list-style-type: none"><li>• If <i>capacity_units</i> is 0, capacity shall be multiple of 1(s), maximum 1022.</li><li>• If <i>capacity_units</i> is 1, capacity shall be multiple of 10(s), maximum 10220.</li><li>• If <i>capacity_units</i> is 2, capacity shall be multiple of 100(s), maximum 102200.</li><li>• If <i>capacity_units</i> is 3, capacity shall be multiple of 1000(s), maximum 1022000.</li><li>• If <i>capacity_units</i> is 4, capacity shall be multiple of 10000(s), maximum 10220000.</li></ul> |

### Definition

Sets the *capacity\_units* field of *codeT* struct.

### Returns

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

**VLScgSetCapacityUnits Error Codes**

| Error Code                | Description                                             |
|---------------------------|---------------------------------------------------------|
| VLScg_SUCCESS             | Success.                                                |
| VLScg_INVALID_INT_TYPE    | If value is not numeric.                                |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds<br>VLScg_CAPACITY_UNITS_MAX_VALUE      |
| VLScg_LESS_THAN_MIN_VALUE | If value is less than<br>VLScg_CAPACITY_UNITS_MIN_VALUE |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgSetCapacity

**Syntax**

```
int VLScgSetCapacity(
VLScg_HANDLE iHandle ,
codeT *codeP ,
char *capacity);
```

| Argument | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iHandle  | The instance handle for this library.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| codeP    | The pointer to the codeT struct.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| capacity | Controls the capacity <ul style="list-style-type: none"><li>If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022.</li><li>If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220.</li><li>If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200.</li><li>If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000.</li><li>If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000.</li></ul> <b>NOLIMITSTR or EMPTY("/0") String can be used to specify infinite capacity.</b> |

**Definition** Sets the *capacity* field of *codeT* struct.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLScgSetCapacity Error Codes**

| Error Code                | Description                         |
|---------------------------|-------------------------------------|
| VLScg_INVALID_INT_TYPE    | If value is not numeric.            |
| VLScg_NOT_MULTIPLE        | If value is not a correct multiple. |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds maximum.           |
| VLScg_LESS_THAN_MIN_VALUE | If value is lower than minimum.     |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowMultiKey

**Syntax**

```
int VLScgAllowMultiKey(
 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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(
 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>    | <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"><li>VLScg_SINGLE_KEY_STRING = “0”</li><li>VLScg_MULTI_KEY_STRING = “1”</li></ul> |

**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:

#### VLScgSetKeyType Error Codes

| Error Code                             | Description                                                   |
|----------------------------------------|---------------------------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If value is not a decimal number.                             |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds <code>VLScg_MULTI_KEY_STRING</code> .        |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is lower than <code>VLScg_SINGLE_KEY_STRING</code> . |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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 `VLScgAllowXXX` 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. <i>num</i> is the position in the array storing the secrets.<br>The value varies from 0 to <i>num_secrets</i> -1, where <i>num_secrets</i> is set using VLScgSetNumSecrets |

**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:

#### VLScgSetSecrets Error Codes

| Error Code                             | Description                                                                |
|----------------------------------------|----------------------------------------------------------------------------|
| <code>VLScg_INVALID_CHARACTERS</code>  | If string is not valid.                                                    |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds maximum. The value set by <code>VLScgSetNumSecrets</code> |
| <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 Codes for License Generation Functions,” on page 453.

## 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 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:

**VLScgSetNumSecrets Error Codes**

| Error Code                             | Description                                           |
|----------------------------------------|-------------------------------------------------------|
| <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 Codes for License Generation Functions,” on page 453.

**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 `VLScgAllowXXX` 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,
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 395 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:

#### VLScgSetVendorInfo Error Codes

| Error Code                | Description                         |
|---------------------------|-------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX should be called. If VLScgAllowXXX returns 1 then the corresponding VLScgSetXXX function can be called. Otherwise, it will return 0 as false.

### 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>     | Position of client in the array of clients: should be from 0 to the maximum number of clients -1.                              |

**Description**      This function sets the number of *keys* (license tokens) 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:

#### VLScgSetKeysPerNode Error Codes

| Error Code                             | Description                                                                                                                                                     |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If number is not a non-negative integer.                                                                                                                        |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If number is greater than <code>num_nl_clients</code> -1. <code>num_nl_clients</code> is a field of <code>CodeT</code> structure storing the number of clients. |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If number is less than 0.                                                                                                                                       |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgAllowSiteLic

### Syntax

```
int VLScgAllowSiteLic(
 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 `VLScgAllowXXX` 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(
 VLScg_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 wildcard (e.g., *.123.*.28) to specify a range.                                                                                                                                               |
| <i>num</i>     | <i>num</i> is the position of subnet in the array maintaining the subnet info.The value varies from 0 to <i>num_subnets-1</i> where <i>num_subnets</i> is an element of <i>CodeT</i> struct set using <i>VLScgSetNumSubnets</i> . |

- Description

Sets subnet address. See “VLScgSetNumSecrets” on page 219.

Specifies the number of subnets used for site licensing.
- Returns

The status code *VLScg\_SUCCESS* is returned if successful. Otherwise, it will return the following error codes:

**VLScgSetSiteLicInfo Error Codes**

| Error Code                 | Description                                                                 |
|----------------------------|-----------------------------------------------------------------------------|
| <i>VLScg_INVALID_RANGE</i> | 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 Codes for License Generation Functions,” on page 453.

**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 whose value is 7. 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:

#### VLScgSetNumSubnets Error Codes

| Error Code                | Description                                                        |
|---------------------------|--------------------------------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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:

**VLScgSetNumFeatures Error Codes**

| Error Code                | Description                                       |
|---------------------------|---------------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If input is not a decimal number.                 |
| VLScg_EXCEEDS_MAX_VALUE   | If value exceeds<br>VLScg_MAX_NUM_FEATURES.       |
| VLScg_LESS_THAN_MIN_VALUE | If value is lower than<br>VLScg_MIN_NUM_FEATURES. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX 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,
 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 soft limit: should be from 0 to <i>NOLIMITSTR</i> for no limit. <i>NOLIMSTR</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:

**VLScgSetSoftLimit Error Codes**

| Error Code                             | Description                                                                                                                                      |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If information is not a non-negative integer.                                                                                                    |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If information exceeds maximum number of license tokens allowed. Maximum value for long codes is 32766 and maximum value for short codes is 254. |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If information is less than 0.                                                                                                                   |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**VLScgAllowKeyLifeUnits**

**Syntax**

```
int VLScgAllowKeyLifeUnits(
 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 `VLScgAllowXXX` 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(
 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>    | Lifetime specification units of license tokens: from 0 to 3. The values are: <ul style="list-style-type: none"><li>• "0" - Multiple of 1 minute(s), maximum 15 minutes.</li><li>• "1" - Multiple of 10 minute(s), maximum 150 minutes.</li><li>• "2" - Multiple of 30 minute(s), maximum 450 minutes.</li><li>• "3" - Multiple of 60 minute(s), maximum 900 minutes.</li></ul> |

**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” on page 175.

**Returns**

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

### VLScgSetKeyLifetimeUnits Error Codes

| Error Code                | Description                               |
|---------------------------|-------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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.                                                                                                                                                                                                                                                                                                                                             |
| <i>codeP</i>   | The pointer to the <i>codeT</i> struct.                                                                                                                                                                                                                                                                                                                                           |
| <i>info</i>    | Hold time specification units of license tokens: from 0 to 3.<br>The values are: <ul style="list-style-type: none"><li>• "0" - Multiple of 1 minute(s), maximum 15 minutes</li><li>• "1" - Multiple of 10 minute(s), maximum 150 minutes.</li><li>• "2" - Multiple of 30 minute(s), maximum 450 minutes.</li><li>• "3" - Multiple of 60 minute(s), maximum 900 minutes.</li></ul> |

**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:

#### VLScgSetKeyHoldtimeUnits Error Codes

| Error Code                | Description                               |
|---------------------------|-------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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 VLScgAllowXXX function tests whether the corresponding VLScg-SetXXX 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:

### VLScgSetKeyHoldtime Error Codes

| Error Code                | Description                                 |
|---------------------------|---------------------------------------------|
| VLScg_INVALID_INT_TYPE    | If information is 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 Codes for License Generation Functions,” on page 453.



## 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 VLScgAllowXXX 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 ,
 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 0-11 or Jan.-Dec. |

### Description

Sets the month of the license start date. The license start month should be specified in the range of 0-11. VLScgSetLicBirthMonth is not applicable if year is infinite.

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

**VLScgSetLicBirthMonth Error Codes**

| Error Code                             | Description                                          |
|----------------------------------------|------------------------------------------------------|
| <code>VLScg_INVALID_CHARACTERS</code>  | If not a valid string.                               |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds maximum allowed month (exceeds 12). |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is less than 1.                             |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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:

#### VLScgSetLicBirthDay Error Codes

| Error Code                             | Description                             |
|----------------------------------------|-----------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>    | If value is not a non-negative integer. |
| <code>VLScg_INVALID_DATE</code>        | If value is not valid for the month.    |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>   | If value exceeds maximum allowed day.   |
| <code>VLScg_LESS_THAN_MIN_VALUE</code> | If value is less than 1.                |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

### 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., 2003) 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:

**VLScgSetLicBirthYear Error Codes**

| Error Code                            | Description                                         |
|---------------------------------------|-----------------------------------------------------|
| <code>VLScg_INVALID_INT_TYPE</code>   | If value is not a non-negative integer.             |
| <code>VLScg_INVALID_YEAR</code>       | If year is invalid.                                 |
| <code>VLScg_INVALID_BIRTH_YEAR</code> | If year is less than 2003.                          |
| <code>VLScg_EXCEEDS_MAX_VALUE</code>  | If value exceeds maximum allowed year that is 2130. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**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 `VLScgAllowXXX` 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: 0 -11 or Jan.-Dec. |

### Description

Sets month of date license expires. The license expiration month should be specified in the range of 0-11. VLScgSetLicExpirationMonth is not applicable if year is infinite.

### Returns

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

#### VLScgSetLicExpirationMonth Error Codes

| Error Code                | Description                                          |
|---------------------------|------------------------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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:

### VLScgSetLicExpirationDay Error Codes

| Error Code                | Description                             |
|---------------------------|-----------------------------------------|
| 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 Codes for License Generation Functions,” on page 453.

## 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., 2003) to avoid year 2000 problem.<br><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:

### VLScgSetLicExpirationYear Error Codes

| Error Code               | Description                             |
|--------------------------|-----------------------------------------|
| VLScg_INVALID_INT_TYPE   | If value is not a non-negative integer. |
| VLScg_INVALID_YEAR       | If the year is invalid.                 |
| VLScg_INVALID_DEATH_YEAR | If year is less than 2003.              |
| VLScg_EXCEEDS_MAX_VALUE  | If value exceeds 2130.                  |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

## VLScgSetNumericType

**Syntax**

```
int VLScgSetNumericType(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 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>num</i>     | Numeric type values are: <ul style="list-style-type: none"><li>• VLScg_NUMERIC_UNKNOWN = "0"</li><li>• VLScg_NOT_NUMERIC = "1"</li><li>• VLScg_MISC_SHORT_NUMERIC = "2"</li><li>• VLScg_MISC_NUMERIC = "3"</li></ul> |

**Description**

Sets the value *codeP->numeric\_type* to the value of *num* and Checks the user input and saves the value in *code* struct.

**Returns**

The status code *VLScg\_SUCCESS* is returned if successful. Otherwise, it will return the following error codes:

### VLScgSetNumericType Error Codes

| Error Code                       | Description                                 |
|----------------------------------|---------------------------------------------|
| <i>VLScg_EXCEEDS_MAX_VALUE</i>   | Value exceeds the maximum value of 3.       |
| <i>VLScg_LESS_THAN_MIN_VALUE</i> | If value is less than 0.                    |
| <i>VLScg_INVALID_INT_TYPE</i>    | If the value is not a non-negative integer. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.



## 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 Codes for License Generation Functions,” on page 453.

## VLScgAllowGracePeriodFlag

**Syntax**

```
int VLScgAllowGracePeriodFlag(
 VLScg_HANDLE iHandle ,
 codeT *codeP);
```

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

**Returns**

The VLScgAllowXXX 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.

## VLScgSetGracePeriodFlag

**Syntax**

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

| Argument       | Description                                                                                                                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i> | The instance handle for this library.                                                                                                                                                                          |
| <i>codeP</i>   | A pointer to the codeT structure.                                                                                                                                                                              |
| <i>flag</i>    | A flag that will decide whether the VLScgAllowGracePeriod should be called or not. The value can be: <ul style="list-style-type: none"><li>VLScg_NO_GRACE_PERIOD</li><li>VLScg_STANDARD_GRACE_PERIOD</li></ul> |

**Description**

Sets the grace period flag value.

**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 Codes for License Generation Functions,” on page 453.

## VLScgAllowGracePeriod

**Syntax**

```
int VLScgAllowGracePeriod(
 VLScg_HANDLE iHandle,
 codeT *codeP);
```

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

**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 Codes for License Generation Functions,” on page 453.

## VLScgSetGracePeriodDays

**Syntax**

```
int VLScgSetGracePeriodDays(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *calendar_daysStr);
```

| Argument                | Description                                                     |
|-------------------------|-----------------------------------------------------------------|
| <i>iHandle</i>          | The instance handle for this library.                           |
| <i>codeP</i>            | The pointer to the codeT struct.                                |
| <i>calendar_daysStr</i> | The number of days the license can be used in the grace period. |

**Description**

Sets the number of calendar days the license can be used in the grace period.

**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 Codes for License Generation Functions,” on page 453.

## VLScgSetGracePeriodHours

**Syntax**

```
int VLScgSetGracePeriodHours(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *elapsed_hoursStr);
```

| Argument                | Description                                                      |
|-------------------------|------------------------------------------------------------------|
| <i>iHandle</i>          | The instance handle for this library.                            |
| <i>codeP</i>            | The pointer to the codeT struct.                                 |
| <i>elapsed_hoursStr</i> | The number of hours the license can be used in the grace period. |

**Description**

Sets the number of elapsed hours the license can be used in a grace period.

**Return**

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 Codes for License Generation Functions,” on page 453.

### VLScgAllowOverdraftFlag

**Syntax**

```
int VLScgAllowOverdraftFlag(
 VLScg_HANDLE iHandle ,
 codeT *codeP);
```

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

**Returns**

The VLScgAllowXXX 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.

### VLScgSetOverdraftFlag

**Syntax**

```
int VLScgSetOverdraftFlag(
 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 codeT struct.                                                                                                                                                                         |
| <i>flag</i>    | A flag that will decide whether the VLScgSetOverdraftFlag should be called or not. The value can be: <ul style="list-style-type: none"><li>VLScg_NO_OVERDRAFT</li><li>VLScg_STANDARD_OVERDRAFT</li></ul> |

**Description**

Sets the overdraft flag value.

**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 Codes for License Generation Functions,” on page 453.

## VLScgAllowOverdraft

### Syntax

```
int VLScgAllowOverdraft(
 VLScg_HANDLE iHandle,
 codeT *codeP);
```

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

### Description

The VLScgAllowOverdraft function tests whether the corresponding overdraft values are valid or not.

### 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 Codes for License Generation Functions,” on page 453.

## VLScgSetOverdraftHours

### Syntax

```
int VLScgSetOverdraftHours(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *hoursStr);
```

| Argument        | Description                                              |
|-----------------|----------------------------------------------------------|
| <i>iHandle</i>  | The instance handle for this library.                    |
| <i>codeP</i>    | The pointer to the codeT struct.                         |
| <i>hoursStr</i> | The number of hours a single license can be checked out. |

### Description

Sets the number of hours a single license can be checked out in an overdraft setting.

**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 Codes for License Generation Functions,” on page 453.

### VLScgSetOverdraftUsers

**Syntax**

```
int VLScgSetOverdraftUsers(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *usersStr);
```

| Argument        | Description                                                                           |
|-----------------|---------------------------------------------------------------------------------------|
| <i>iHandle</i>  | The instance handle for this library.                                                 |
| <i>codeP</i>    | A pointer to the codeT structure.                                                     |
| <i>usersStr</i> | The number of users or percentage of the hard limit that can be allowed as overdraft. |

**Description** Sets the number of users or percentage of the hard limit that can be allowed as overdraft.

**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 Codes for License Generation Functions,” on page 453.

### VLScgAllowLocalRequestLockCritFlag

**Syntax**

```
int VLScgAllowLocalRequestLockCritFlag(
 VLScg_HANDLE iHandle,
 codeT *codeP);
```

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

**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 Codes for License Generation Functions,” on page 453.

## VLScgSetLocalRequestLockCritFlag

**Syntax**

```
int VLScgSetLocalRequestLockCritFlag(
 VLScg_HANDLE iHandle,
 codeT *codeP,
 char *str);
```

| Argument       | Description                                                                     |
|----------------|---------------------------------------------------------------------------------|
| <i>iHandle</i> | The instance handle for this library.                                           |
| <i>codeP</i>   | The pointer to the codeT struct.                                                |
| <i>str</i>     | The value that defines whether the locking criteria is default or user-defined. |

**Description** Sets the locking criteria as default or user-defined.

**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 Codes for License Generation Functions,” on page 453.

## VLScgAllowLocalRequestLockCrit

**Syntax**

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

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

**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 Codes for License Generation Functions,” on page 453.

### VLScgSetLocalRequestLockCrit

**Syntax**

```
int VLScgSetLocalRequestLockCrit(
 VLScg_HANDLE iHandle,
 codeT *codeP),
 char* str,
```

| Argument       | Description                                                                                                                                                                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i> | The instance handle for this library.                                                                                                                                                                                                                                                                                         |
| <i>codeP</i>   | The pointer to the codeT struct.                                                                                                                                                                                                                                                                                              |
| <i>str</i>     | Allows setting the required, floating, and minimum required locking criteria for the local request. For example, 0x4:0x3FF:2 will set: <ul style="list-style-type: none"><li>• 0x4 as the required locking criteria.</li><li>• 0x3FF as the floating locking criteria.</li><li>• 2 as the minimum locking criteria.</li></ul> |

**Description** Sets the required, floating, and minimum required locking criteria for the local request.

**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 Codes for License Generation Functions,” on page 453.

## License Generation Function

The following table summarizes the license generation function:

License Generation Function

| Function                          | Description                   |
|-----------------------------------|-------------------------------|
| <code>VLScgGenerateLicense</code> | Generates the license string. |



## 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 deallocation 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:

### VLScgGenerateLicense Error Codes

| Error Code                   | Description                                      |
|------------------------------|--------------------------------------------------|
| VLScg_INVALID_VENDOR_CODE    | If vendor identification is illegal.             |
| VLScg_VENDOR_ENCRYPTION_FAIL | If vendor-customized encryption fails.           |
| VLScg_LICMETER_NOT_SUPPORTED | Your Sentinel LM License Meter is not supported. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

# License Decode Function

The following table summarizes the license decode function:

License Decode Function

| Function           | Description                 |
|--------------------|-----------------------------|
| VLScgDecodeLicense | Decodes the license string. |

## VLScgDecodeLicense

**Syntax**

```
int VLScgDecodeLicense(
 VLScg_HANDLE iHandle,
 char *AnyLicenseString,
 char *lic_string,
 int lic_string_length,
 codeT **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> (OUT)  | 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**

VLScgDecodeLicense API is contained in *lsdcod32.lib*. This library needs to be called for using VLScgDecodeLicense API without the license meter.

VLScgDecodeLicense 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 *AnyLicenseString*. This function takes care of all memory allocations it uses.

---

**Note:** When decoding a license via the VLScgDecodeLicense API function, the codeT structure returned does not contain the license secrets—instead the corresponding field contains an empty string.

---

**Returns**

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

**VLScgDecodeLicense Error Codes**

| Error Code                   | Description                                                          |
|------------------------------|----------------------------------------------------------------------|
| VLScg_INVALID_VENDOR_CODE    | If vendor identification is illegal.                                 |
| VLScg_VENDOR_ENCRYPTION_FAIL | If vendor-customized encryption fails.                               |
| VLScg_MALLOC_FAILURE         | Out of heap memory.                                                  |
| VLScg_SHORT_STRING           | License string too small to parse.                                   |
| VLScg_PREMATURE_TERM         | Premature termination of license string. Please check.               |
| VLScg_INVALID_CHARS          | String is not valid.                                                 |
| VLScg_EXCEEDS_MAX_STRING     | Length of the string is greater than the defined limit.              |
| VLScg_REMAP_DEFAULT          | Failed to remap default strings from configuration file for license. |
| VLScg_DECRYPT_FAIL           | Decryption failed for license code.                                  |
| VLScg_INVALID_CHKSUM         | Checksum validation failed for license string.                       |
| VLScg_FIXED_STR_ERROR        | Default fixed string error.                                          |
| VLScg_INVALID_RANGE          | Value violates the valid range of input.                             |
| VLScg_INVALID_INPUT          | Invalid input.                                                       |
| VLScg_INVALID_INT_TYPE       | Value is not numeric.                                                |
| VLScg_LESS_THAN_MIN_VALUE    | Value entered is less than the minimum supported value.              |

**VLScgDecodeLicense Error Codes**

| Error Code                   | Description                                                                         |
|------------------------------|-------------------------------------------------------------------------------------|
| VLScg_LESS_THAN_MAX_VALUE    | Value entered is greater than the maximum supported value.                          |
| VLScg_INVALID_HEX_TYPE       | Wrong value entered. (Should be hexadecimal).                                       |
| VLScg_SECRET_DECRYPT_FAILURE | Decryption failed for secrets. Verify the configuration file for readable licenses. |
| VLScg_SIMPLE_ERROR           | Error in license string. Please check.                                              |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

**License Meter Related Functions**

The following table summarizes the license meter related functions:

**License Meter Related Functions**

| Function                       | Description                                           |
|--------------------------------|-------------------------------------------------------|
| VLScgGetLicenseMeterUnits      | Returns the number of license generation units.       |
| VLScgGetTrialLicenseMeterUnits | Returns the number of trial license generation units. |

**VLScgGetLicenseMeterUnits**

**Syntax**

```
int VLScgGetLicenseMeterUnits(
VLScg_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 Sentinel LM 7.x and 8 for Sentinel LM 7.3.0 and greater). |

**Description** Returns the number of license generation units available in the attached license meter key.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

#### VLScgGetLicenseMeterUnits Error Codes

| Error Code                      | Description                                      |
|---------------------------------|--------------------------------------------------|
| VLScg_LICMETER_EXCEPTION        | Unknown value in accessing the license meter.    |
| VLScg_LICMETER_ACCESS_ERROR     | Error accessing the license meter.               |
| VLScg_LICMETER_CORRUPT          | License meter is corrupted.                      |
| VLScg_LICMETER_VERSION_MISMATCH | License meter has an invalid version.            |
| VLScg_LICMETER_NOT_SUPPORTED    | Your Sentinel LM License Meter is not supported. |

For a complete list of the error codes, see Appendix D, “Error Codes for License Generation Functions,” on page 453.

On platforms that do not support hardware keys, the function returns *V\_FAIL*.

## VLScgGetTrialLicenseMeterUnits

**Syntax** `int VLScgGetTrialLicenseMeterUnits(`

```
VLScg_HANDLE iHandle ,
int units ,
int codegen_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 Sentinel LM 7.x and 8 for Sentinel LM 7.3.0 and greater). |

- 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 Codes for License Generation Functions,” on page 453.

## License Revocation Ticket Decode Function

### VLScgDecodeLicenseRevocationTicket

```
Syntax int VLScgDecodeLicenseRevocationTicket(
char *license_revocation_ticket_buffer,
int license_revocation_ticket_buffer_size,
char *secret_key,
int secret_key_length,
VLSrevocationTicketInfoT
 *revocation_ticket);
```

| Argument                                     | Description                                                                                                                                        |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>license_revocation_ticket_buffer</i>      | A buffer that holds the string data returned by the VLSrevokeLicense API function.                                                                 |
| <i>license_revocation_ticket_buffer_size</i> | A buffer that defines the size of the ticket.                                                                                                      |
| <i>secret_key</i>                            | Refers to the first secret specified in the license code. It is used by the license server for generating the encrypted license revocation ticket. |
| <i>secret_key_length</i>                     | The length of the secret key.                                                                                                                      |
| <i>revocation_ticket</i>                     | The VLSrevocationTicketInfoT structure described below.                                                                                            |

**Description** Decodes the license revocation ticket generated.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Refer to the error codes given in Appendix D, “Error Codes for License Generation Functions” on page 453.

## The VLSrevocationTicketInfoT Structure

**Description**     The VLSrevocationTicketInfoT structure used by the VLScgDecodeLicenseRevocationTicket function.

**Syntax**

```
typedef struct
{
 unsigned long struct_size;
 unsigned long time_stamp;
 unsigned char feature_name[VLS_MAXFEALEN];
 unsigned char feature_version[VLS_MAXFEALEN];
 unsigned long capacity;
 unsigned int base_license_hard_limit;
 unsigned int number_licenses_revoked;
 unsigned int total_number_licenses_revoked;
 unsigned long capacity_revoked;
 unsigned long server_locking_criteria;
 unsigned char
server_locking_info[VLS_MAXSRVLOCKLEN];
} VLSrevocationTicketInfoT;
```

| Member                         | Description                                                                  |
|--------------------------------|------------------------------------------------------------------------------|
| <i>struct_size</i>             | The size of the structure.                                                   |
| <i>time_stamp</i>              | The time when the license revocation ticket was generated.                   |
| <i>feature_name</i>            | The feature name for which the license revocation ticket was generated.      |
| <i>feature_version</i>         | The feature version for which the license revocation ticket was generated.   |
| <i>capacity</i>                | The capacity of the feature whose license revocation ticket was generated.   |
| <i>base_license_hard_limit</i> | The hard limit of the feature whose license revocation ticket was generated. |



| Member                               | Description                                                                      |
|--------------------------------------|----------------------------------------------------------------------------------|
| <i>number_licenses_revoked</i>       | The number of licenses revoked.                                                  |
| <i>total_number_licenses_revoked</i> | The total number of licenses revoked for a feature.                              |
| <i>capacity_revoked</i>              | The capacity of the feature revoked.                                             |
| <i>server_locking_criteria</i>       | The locking criteria of the license server on which the revocation was executed. |



# Chapter 5

---

## Redundancy API

License server 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. The heartbeat interval is set to 20 seconds and the heartbeat time out interval is set to 120 seconds, meaning that when a follower server goes down, the b server would mark its follower server status to UNKNOWN/DOWN only if the heartbeat is not returned after 120 seconds (20+100).

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 *Sentinel LM Developer's Guide*.

## Redundancy Functions and API

The following table summarizes the redundancy functions:

**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, but only until the license server is shut down 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.                                                                                                                                                                                                                                                                                                                                                |
| VLSaddServerToPool     | Sends a request to add a new license server into the pool. This API will actually modify the license redundant file 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   | Removes 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 based on some criteria.                                                                                                                                                                          |

**Redundancy Functions (Continued)**

| Function                    | Description                                                                                                                                                                                                                                |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLSgetDistbCrit             | Returns the current token distribution status for the given license feature and version.                                                                                                                                                   |
| VLSgetDistbCritToFile       | Requests the license server provide current token distribution status for the given license feature and version or for all features or versions (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 provide information for the given license feature and version.                                                                                                                                                 |
| VLSgetLeaderServer Name     | 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. |
| VLSgetLicSharingServer List | Returns the names of the license servers 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).                                |
| VLSgetPoolServerList        | Returns a list of redundant license servers and their status.                                                                                                                                                                              |
| VLSsetBorrowingStatus       | Turns borrowing on/off for given feature and version.                                                                                                                                                                                      |
| VLSsetServerLogState        | Turns logging on/off for the given event.                                                                                                                                                                                                  |

## VLSaddFeature

**Syntax**

```
int VLSaddFeature (
 unsigned char *licenseStr,
 unsigned char *unused1,
 LS_CHALLENGE *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*.

---

**Note:** A feature is not permanently added to the license server, but is cleared when the license server is shut down and restarted.

---

**Returns**

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

**VLSaddFeature Error Codes**

| Error Code         | Description                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------|
| 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.                                                                         |

**VLSaddFeature Error Codes (Continued)**

| Error Code             | Description                                                                                                                                                                             |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSaddFeatureExt

### Syntax

```
int VLSaddFeatureExt (
 unsigned char *licenseString,
 unsigned char *DistCritString,
 unsigned char *unused1,
 LS_CHALLENGE *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:

#### VLSaddFeatureExt Error Codes

| Error Code            | Description                                                                                                                                                                             |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.                                                                                        |



VLSaddFeatureExt Error Codes (Continued)

| Error Code             | Description                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------|
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSaddFeatureToFile

Syntax

```
int VLSaddFeatureToFile(
 unsigned char *licenseString,
 unsigned char *unused1 ,
 unsigned char *unused2 ,
 unsigned char *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.

---

**Note:** The 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:

**VLSaddFeatureToFile Error Codes**

| Error Code             | Description                                                                                                                                                                             |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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     | 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, “Sentinel LM Error and Result Codes,” on page 431.

## VLSaddServerToPool

### Syntax

```
int VLSaddServerToPool(
 char *server_name,
 char *server_addr);
```

| 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:

#### VLSaddServerToPool Error Codes

| Error Code                 | Description                                                                                                                                                                                                    |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR          | <ul style="list-style-type: none"> <li><i>server_name</i> is NULL</li> <li><i>server_address</i> is NULL</li> <li>Using stand-alone library. This function cannot be used with stand-alone library.</li> </ul> |
| LS_NO_SUCCESS              | Generic error indicating that the license server could not be added to the pool.                                                                                                                               |
| 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.                                                                                                                                                                                  |

**VLSaddServerToPool Error Codes (Continued)**

| Error Code                  | Description                                                                                      |
|-----------------------------|--------------------------------------------------------------------------------------------------|
| 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_MESS AGE     | 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**VLChangeDistbCrit**

**Syntax**

```
int VLChangeDistbCrit(
 char *feature_name,
 char *version,
 char *dist_crit);
```

| Argument            | Description                                                                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i> | Name of the feature. Must be unique.                                                                                                                                                           |
| <i>version</i>      | <i>Version</i> of the feature.                                                                                                                                                                 |
| <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.

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

#### VLSchangeDistbCrit Error Codes

| Error Code                                     | Description                                                                   |
|------------------------------------------------|-------------------------------------------------------------------------------|
| <code>LS_BAD_DIST_CRIT</code>                  | Change <i>dist_crit</i> and allocate some keys to the deleted license server. |
| <code>LS_NON_REDUNDANT_SERVER_CONTACTED</code> | LSHOST is set to a non-redundant license server.                              |
| <code>LS_BAD_PARAMETER</code>                  | License server's name is NULL or an empty string.                             |
| <code>LS_NO_AUTHORIZATION</code>               | License server does not recognize this feature name.                          |
| <code>LS_NO_SUCH_FEATURE</code>                | <i>Feature_version</i> is non-existent.                                       |
| <code>LS_UNRESOLVED_SERVER_NAME</code>         | License server's name cannot be resolved.                                     |
| <code>LS_MSG_TO_LEADER</code>                  | The request has been sent to the leader license server.                       |

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 431.

#### VLSdelServerFromPool

**Syntax**

```
int VLSdelServerFromPool(
 char *server_name,
 char *server_addr);
```

| 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** Removes a license server's name from the pool of redundant license servers. This API modifies 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:

**VLSdelServerFromPool Error Codes**

| Error Code                  | Description                                                                                                                                                                                                      |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR           | <ul style="list-style-type: none"><li>• <i>server_name</i> is NULL</li><li>• <i>server_address</i> is NULL</li><li>• Using stand-alone library. This function cannot be used with stand-alone library.</li></ul> |
| 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_CONF_FILE_ERROR         | Error in configuration file.                                                                                                                                                                                     |
| VLS_SERVER_SYNC_IN_PROGRESS | License server synchronization in process.                                                                                                                                                                       |
| 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.                                                                                                                                      |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSdiscoverExt

Syntax

```
int VLSdiscoverExt(
 unsigned char *feature_name,
 unsigned char *version,
 unsigned char *unused1,
 int *num_servers,
 VLSdiscoverInfo *discoverInfo,
 int *option_Flag,
 int *sharing_crit,
 char *vendor_list);
```

| 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. Valid flags are:</p> <ul style="list-style-type: none"> <li>• VLS_DISC_NO_USERLIST – Do not check the host list specified by the user. By default, it first consults the LSFORCEHOST environment variable. If LSFORCEHOST does not exist, it reads the file <i>LSHOST/lshost</i>.</li> <li>• VLS_DISC_RET_ON_FIRST – If the combined query list is NULL, it returns the name of the first contacted license server in the <i>server_list</i>, as soon as it is contacted by any of the license servers. Otherwise, it returns the name of the first contacted license server specified in the combined query list. If this option is not specified. VLSdiscover returns all the license servers that responded.</li> <li>• VLS_DISC_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. If this option is not specified, the combined query list is treated as random.</li> <li>• VLS_DISC_REDUNDANT_ONLY – Expecting reply only from redundant license servers. All non-redundant license servers will ignore the message.</li> <li>• VLS_DISC_DEFAULT_OPTIONS – This flag is a combination of the aforementioned flags. Use it if you are not sure which flag you want to specify.</li> </ul> |
| <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"> <li>• VLScg_NO_SHARING</li> <li>• VLScg_USER_SHARING</li> <li>• VLScg_HOSTNAME_SHARING</li> <li>• VLScg_XDISPLAY_SHARING</li> <li>• VLScg_VENDOR_SHARING</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>vendor_list</i>  | <p>Consists of server names. These license serves 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>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

**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:

#### VLSdiscoverExt Error Codes

| Error Code                        | Description                                                                                      |
|-----------------------------------|--------------------------------------------------------------------------------------------------|
| 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.                      |
| 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 | The license server contacted is non-redundant, and does not support this function.               |

**VLsdiscoverExt Error Codes (Continued)**

| Error Code                | Description                                                                                  |
|---------------------------|----------------------------------------------------------------------------------------------|
| LS_UNRESOLVED_SERVER_NAME | License server's name is not resolvable.                                                     |
| VLS_LEADER_NOT_PRESENT    | Leader name is not known.                                                                    |
| VLS_NON_REDUNDANT_SERVER  | The license server contacted is non-redundant, and therefore does not support this function. |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**VLsgetDistbCrit**

**Syntax**

```
int VLsgetDistbCrit (
char *feature_name ,
char *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.                                                                                                                                                        |
| <i>dist_crit (OUT)</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. |
| <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:

**VLSgetDistbCrit Error Codes**

| Codes                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR           | <ul style="list-style-type: none"><li>• <i>feature_name</i> is NULL</li><li>• <i>version</i> is NULL</li><li>• <i>dist_crit</i> is NULL</li><li>• <i>dist_crit_len</i> is zero or negative</li><li>• <i>challenge</i> argument is non-NULL, but cannot be understood.</li></ul> Using stand-alone library. This function cannot be used with stand-alone library. Also, both feature name and version cannot be NULL at the same time. |
| VLS_NO_SUCH_FEATURE         | License server does not have a license that matches requested feature, version and capacity.                                                                                                                                                                                                                                                                                                                                           |
| 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.                                                                                                                                                                                                                                                                                                                                       |

**VLSgetDistbCrit Error Codes (Continued)**

| Codes                  | Description                                                                              |
|------------------------|------------------------------------------------------------------------------------------|
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**VLSgetDistbCritToFile****Syntax**

```
int VLSgetDistbCritToFile(
char *feature_name,
char *feature_version,
char *file_name);
```

| 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 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:

**VLSgetDistbCritToFile Error Codes**

| Error Code                  | Description                                                                                                                                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR           | <ul style="list-style-type: none"><li>• <i>feature_name</i> is NULL</li><li>• <i>file_name</i> is NULL.</li></ul> 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, version and capacity.                                                                                                        |
| 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.                                                                                                                                                                       |

VLSgetDistbCritToFile Error Codes (Continued)

| Error Code                        | Description                                       |
|-----------------------------------|---------------------------------------------------|
| 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_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 C, “Sentinel LM Error and Result Codes,” on page 431.

### VLSgetFeatureInfoToFile

**Syntax**

```
int VLSgetFeatureInfoToFile (
 unsigned char *feature_name,
 unsigned char *version,
 char *file_name);
```

| Argument            | Description                                                                           |
|---------------------|---------------------------------------------------------------------------------------|
| <i>feature_name</i> | Name of the feature.                                                                  |
| <i>version</i>      | <i>Version</i> of the feature. Must be unique.                                        |
| <i>file_name</i>    | License server will provide information for the specified license feature or version. |

**Description**

Requests the license server to provide all feature information for the given license to *file\_name*.

**Returns**

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

**VLSgetFeatureInfoToFile Error Codes**

| Error Code             | Description                                                                                                       |
|------------------------|-------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR      | <ul style="list-style-type: none"><li>• <i>file_name</i> is NULL</li><li>• <i>feature_name</i> is NULL.</li></ul> |
| VLS_NO_SUCH_FEATURE    | License server does not have a license that matches requested feature, version and capacity.                      |
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.



## VLSgetHostName

**Syntax**

```
int VLSgetHostName(
char *IP_address ,
char *hostname ,
int HostNameBufLen);
```

| Argument              | Description                                             |
|-----------------------|---------------------------------------------------------|
| <i>IP_address</i>     | IP addresses to be converted to <i>hostname</i> .       |
| <i>hostname (OUT)</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:

**VLSgetHostName Error Codes**

| Error Code                | Description                                                                                                                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR         | <ul style="list-style-type: none"><li>• <i>IP_address</i> is NULL</li><li>• <i>hostName</i> is NULL</li><li>• <i>hostNameBufLen</i> is NULL</li></ul> 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetLeaderServerName

**Syntax**

```
int VLSgetLeaderServerName(
 char *leader_name);
```

| Argument           | Description                                                                                                                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>leader_name</i> | Current lead license server's name.<br>Return types: <ul style="list-style-type: none"><li>• 0 = Success. Found leader license server name.</li><li>• 1 = Contact license server is not a redundant license server.</li><li>• 2 = Other error.</li></ul> |

**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.

**Returns** The status code LS\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

### VLSgetLeaderServerName Error Codes

| Error Code                  | Description                                                                                                           |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR           | <ul style="list-style-type: none"><li>• <i>leader_name</i> is NULL</li><li>• <i>leadername_len</i> is NULL.</li></ul> |
| 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.                                |

**VLSgetLeaderServerName Error Codes (Continued)**

| Error Code               | Description                                                                                      |
|--------------------------|--------------------------------------------------------------------------------------------------|
| 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_LEADER_NOT_PRESENT   | Leader name is not known.                                                                        |
| VLS_NON_REDUNDANT_SRVR   | The license server is non-redundant, and therefore does not support this operation.              |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetHostAddress

**Syntax**

```
int VLSgetHostAddress(
 char *hostname ,
 char *IP_AddressBuf ,
 int IPAddrBufLen);
```

| Argument            | Description                                                                             |
|---------------------|-----------------------------------------------------------------------------------------|
| hostname            | The host name of the computer containing the license server that is using the log file. |
| IP_AddressBuf (OUT) | Pointer to the IP address buffer.                                                       |
| IPAddrBufLen        | 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:

### VLSgetHostAddress Error Codes

| Error Code              | Description                                                                                                                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR       | <ul style="list-style-type: none"><li>• <i>IPaddressBuf</i> is NULL</li><li>• <i>IPAddrBufLen</i> is NULL.</li><li>• Using stand-alone library. This function cannot be used with stand-alone library.</li></ul> |
| VLS_UNRESOLVED_HOSTNAME | <ul style="list-style-type: none"><li>• <i>hostname</i> is valid, but could not be resolved.</li><li>• IPX protocol is current.</li></ul>                                                                        |

---

**Note:** If you wish to use IPX protocol for client-server communication then you must set the LSPROTOCOL environment variable to IPX on the client end.

---

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetLicSharingServerList

Syntax

```
int VLSgetLicSharingServerList(
char *feature_name,
char *feature_version,
char *server_list_len,
int *server_list,
int *num_servers);
```

| Argument               | Description                                                                                                                       |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i>    | Name of the feature.                                                                                                              |
| <i>feature_version</i> | Version of the feature.                                                                                                           |
| <i>server_list</i>     | A list that contains the license server's names ( <i>hostNames</i> or IPX addresses).                                             |
| <i>server_list_len</i> | License server will retrieve all the license servers names. If the list is larger than the specified limit, it will be truncated. |
| <i>num_servers</i>     | 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:

### VLSgetLicSharingServerList Error Codes

| Error Code        | Description                                                                                                                                                                                                                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR | <ul style="list-style-type: none"> <li><i>feature_name</i> is NULL</li> <li><i>feature_version</i> is NULL</li> <li><i>server_list</i> is NULL</li> <li><i>server_list_len</i> is zero.</li> <li>License server's name is NULL or an empty string.</li> </ul> <p>Both feature name and version cannot be NULL at the same time.</p> |

**VLSgetLicSharingServerList Error Codes (Continued)**

| Error Code                  | Description                                                                                      |
|-----------------------------|--------------------------------------------------------------------------------------------------|
| LS_BUFFER_TOO_SMALL         | <i>server_list_len</i> 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, version and capacity. |
| 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.                                                |
| VLS_NON_REDUNDANT_SRVR      | The license server is non-redundant, and therefore does not support this operation.              |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetPoolServerList

### Syntax

```
int VLSgetPoolServerList(
 char *outBuf,
 int outBufSz);
```

| Argument             | Description         |
|----------------------|---------------------|
| <i>*outBuf (OUT)</i> | Output buffer.      |
| <i>outBufSz</i>      | Output buffer size. |

### Description

Returns a list of license servers and their status where the servers are in the same pool as the contacted redundant license server. The status for each license server in the list indicates whether that server is active (running) or not. If a non-redundant license server is contacted, the VLS\_NON\_REDUNDANT\_SRVR error code is returned. The license server to be contacted is selected by VLSgetServerName, so you must set the license server's name before calling VLSgetPoolServerList.

### Returns

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

#### VLSgetPoolServerList Error Codes

| Error Code                  | Description                                                                         |
|-----------------------------|-------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR           | License server's name is NULL or an empty string.                                   |
| LS_BUFFER_TOO_SMALL         | The output buffer is too small.                                                     |
| VLS_SERVER_SYNC_IN_PROGRESS | License server synchronization in process.                                          |
| VLS_DIFF_LIB_VER            | Version mismatch between license server API and client API.                         |
| LS_NORESOURCES              | Insufficient resources (such as memory) as available to complete the request.       |
| VLS_NON_REDUNDANT_SRVR      | The license server is non-redundant, and therefore does not support this operation. |
| VLS_LEADER_NOT_PRESENT      | Unknown leader.                                                                     |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSsetBorrowingStatus

### Syntax

```
int VLSsetBorrowingStatus(
 char *feature_name,
 char *feature_version,
 int state);
```

| Argument               | Description                                 |
|------------------------|---------------------------------------------|
| <i>feature_name</i>    | Name of the feature.                        |
| <i>feature_version</i> | Version of the application. Must be unique. |
| <i>state</i>           | Borrowing state: VLS_ON or VLS_OFF.         |

### Description

Turns token borrowing on or off for the given feature and version. The license server to be contacted is selected by VLSgetServerName, so you must set the license server’s name before calling VLSgetPoolServerList.

### Returns

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

#### VLSsetBorrowingStatus Error Codes

| Error Code                  | Description                                                    |
|-----------------------------|----------------------------------------------------------------|
| VLS_CALLING_ERROR           | License server’s name is NULL or an empty string.              |
| VLS_SERVER_SYNC_IN_PROGRESS | License server synchronization in process.                     |
| VLS_DIFF_LIB_VER            | Version mismatch between license server API and client API.    |
| VLS_LEADER_NOT_PRESENT      | Leader name is not known.                                      |
| VLS_FEATURE_INACTIVE        | The given feature is inactive on the specified license server. |
| VLS_MSG_TO_LEADER           | The request has been sent to the leader license server.        |



**VLSsetBorrowingStatus Error Codes (Continued)**

| Error Code             | Description                                                                                                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| VLS_NO_SUCH_FEATURE    | The license server does not have a license that matches the specified feature, version and capacity.                                 |
| VLS_NOT_AUTHORIZED     | The user making the request is invalid.                                                                                              |
| VLS_VENDORIDMISMATCH   | The vendor ID of the requesting application does not match the vendor ID for the feature for which the license server has a license. |
| VLS_NON_REDUNDANT_SRVR | The license server is non-redundant, and therefore does not support this operation.                                                  |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSsetServerLogState

Syntax

```
int VLSsetServerLogState(
 int event,
 int state);
```

| Argument     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>event</i> | The event you want to log. Event may be: <ul style="list-style-type: none"><li>• LOG_SRVR_UP (the license server is running)</li><li>• LOG_LDR_ELECT (a pool leader has been elected)</li><li>• LOG_HRT_BT (license server heartbeat)</li><li>• LOG_BORROW_REQ_RESP (token borrowing request and response)</li><li>• LOG_USG_NOTIFY (follower notifies pool leader of token use)</li><li>• LOG_CHNG_DIST_CRT (token distribution criteria has changed)</li><li>• LOG_DIST_CRT_SYNC (the pool servers are synchronizing distribution criteria)</li><li>• LOG_CFG_FILE (the <i>LSERVRLF</i> file changed)</li><li>• LOG_SRVR_DOWN (license server is down)</li><li>• LOG_MOD_SERVER (addition or deletion of a license server to or from the pool)</li><li>• LOG_ADD_DEL_LIC (redundant license has been added or deleted to or from the pool)</li></ul> |
| <i>state</i> | Logging state: VLS_ON or VLS_OFF.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

Description

Turns logging on or off for the given event. The license server to be contacted is selected by VLSgetServerName, so you must set the license server's name before calling VLSgetPoolServerList.

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

**VLSsetServerLogState Error Codes**

| Error Code                          | Description                                                         |
|-------------------------------------|---------------------------------------------------------------------|
| <code>VLS_CALLING_ERROR</code>      | License server's name is NULL or an empty string.                   |
| <code>VLS_DIFF_LIB_VER</code>       | Version mismatch between license server API and client API.         |
| <code>VLS_FEATURE_INACTIVE</code>   | The given feature is inactive on the specified license server.      |
| <code>VLS_MSG_TO_LEADER</code>      | The request has been sent to the leader license server.             |
| <code>VLS_NOT_AUTHORIZED</code>     | The user making the request is invalid.                             |
| <code>VLS_BAD_SERVER_MESSAGE</code> | A message returned from the license server could not be understood. |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.



# 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) 2004 SafeNet, Inc.
/* All Rights Reserved
/*
/*This Module contains Proprietary Information of
/*SafeNet, 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 Sentinel LM license manager */
returnCode =
VLSqueuedRequest
(LS_ANY,
(unsigned char LSFAR *)"Sentinel LM 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:

License Queuing Functions

| Function                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLSqueuePreference Struct               | Specifies the clients preference for how it wishes to be placed in the queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| VLSserverInfo Struct                    | Stores information about the server.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| VLSgetQueuedClientInfo Struct           | Returns client information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| VLSqueuedRequest<br>VLSqueuedRequestExt | An integrated request for an authorized license code from the license server. Use this API to: <ul style="list-style-type: none"><li>Request a license, with option to queue (<i>requestFlag</i> = VLS_REQ_GET   VLS_REQ_QUEUE).</li><li>Request a license without queuing (<i>requestFlag</i> = VLS_REQ_GET). This option has the same effect as calling a non-queuing API request (LSRequest, VLSrequestExt, etc.).</li><li>Request to be placed in the queue, even if the license server has available licenses (<i>requestFlag</i> = VLS_REQ_QUEUE).</li></ul> |
| VLSgetQueuedClientInfo                  | Retrieves the current information of a queued client, such as the number of requested <i>licenses</i> , <i>feature_name</i> , <i>version</i> , and <i>index</i> .                                                                                                                                                                                                                                                                                                                                                                                                  |
| VLSremoveQueuedClient                   | Removes a queued client from the queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



## License Queuing Functions (Continued)

| Function                  | Description                                                                                                                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.                                                                                                      |
| VLSqueuePreference Struct | Specifies the client preference for getting into the queue.                                                                                                                                                                                                                    |
| VLSinitQueuePreference    | Initializes provided queue preference structure to default values.                                                                                                                                                                                                             |

## VLSqueuePreference Struct

```
typedef struct {
 long wait_time;
 long hold_time;
 int priority_num;
 long absPosition;
 long grpPosition;
} VLSqueuePreference;
```

## VLSqueuePreference Struct

| Member           | Description                                                                                                             |
|------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>wait_time</i> | Maximum time, the client can be in queue.                                                                               |
| <i>hold_time</i> | After allotment, the maximum time interval for which the server will keep the requested units reserved for this client. |

**VLSqueuePreference Struct (Continued)**

| Member              | Description                                                                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>priority_num</i> | Priority vis-a-vis other clients, as decided by the client application. For use in future. Not implemented in SLM7.0.                                                                           |
| <i>absPosition</i>  | The maximum position within the queue, before which the client can be queued.                                                                                                                   |
| <i>grpPosition</i>  | The maximum position within the queue, considering only those queued clients that belong to the same group as this client, before which the client can be queued -1 if the client doesn't care. |

**VLSserverInfo Struct**

```
typedef struct {
 char identifier[VLS_MAX_NAME_LEN];
 char inBuf[VLS_MAX_BUF_LEN];
 char outBuf[VLS_MAX_BUF_LEN];
} VLSserverInfo;
```

**VLSserverInfo Struct**

| Member            | Description                                               |
|-------------------|-----------------------------------------------------------|
| <i>identifier</i> | The name of the server hook which the user wants to call. |
| <i>inBuf</i>      | String passed to the server.                              |
| <i>outBuf</i>     | String returned by the server.                            |

**VLSgetQueuedClientInfo Struct**

```
typedef struct queued_client_info_struct {
 char user_name[VLS_MAXLEN];
 char host_name[VLS_MAXLEN];
 char x_display_name[VLS_MAXLEN];
 char shared_id_name[VLS_MAXLEN];
 char group_name[VLS_MAXLEN];
}
```

```

unsigned long host_id;
long server_start_time;
long server_end_time;
unsigned long qkey_id;
int num_units;
int num_resvd_default;
int num_resvd_native;
long wait_time; /*in secs*/
long hold_time; /*in secs*/
int priority_num;
long absPosition; /
long grpPosition;
long availabilityTime;
} VLSqueuedClientInfo;

```

### VLSqueuedClientInfo Struct

| Member                | Description                                                                                                                                                                                                                                                                       |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>user_name</i>      | The login name of the user using the application, where MAXLEN is set to 64 characters.                                                                                                                                                                                           |
| <i>host_name</i>      | Name of the host/computer where the user is running the application, where MAXLEN is set to 64 characters.                                                                                                                                                                        |
| <i>x_display_name</i> | Name of the X display where the user is displaying the application, where MAXLEN is set to 64 characters.                                                                                                                                                                         |
| <i>shared_id_name</i> | A special vendor-defined ID that can be used for license sharing decisions. It always has the fixed value, defaultsharing- ID, unless it is changed by registering a custom function using the VLSsetSharedId API call. The maximum length of the string is set to 64 characters. |
| <i>group_name</i>     | Name of the reserved group to which the user belongs, where MAXLEN is set to 64 characters. If the user does not belong to an explicitly named group, <i>DefaultGrp</i> is returned.                                                                                              |
| <i>host_id</i>        | The host ID of the computer on which the user is working.                                                                                                                                                                                                                         |

**VLSqueuedClientInfo Struct (Continued)**

| Member            | Description                                                                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| server_start_time | <i>server_start_time</i> is the start time of the license token.                                                                                                     |
| server_end_time   | <i>server_end_time</i> is the end time of the license token. <i>server_end_time</i> should be interpreted as <i>start_time</i> + heart beat interval of the license. |
| qkey_id           | Identifier of the client queue.                                                                                                                                      |
| num_units         | Number of units consumed by the client so far.                                                                                                                       |
| num_resvd_default | The number of tokens given to this queued token from default pool, that is from the unreserved tokens.                                                               |
| num_resvd_native  | The number of tokens given to this queued token from its reservation group.                                                                                          |
| wait_time         | Maximum time (in seconds), the client can be in queue.                                                                                                               |
| hold_time         | After allotment, the maximum time interval (in seconds) for which the server will keep the requested units reserved for this client.                                 |
| priority_num      | Priority vis-a-vis other clients, as decided by the client application. For use in future. Not implemented in SLM7.0.                                                |
| absPosition       | The maximum position within the queue, before which the client can be queued.                                                                                        |
| grpPosition       | Current position within the queue, considering only those queued clients that belong to the same group to which this client belongs to.                              |

**VLSqueuedRequest and VLSqueuedRequestExt****Syntax**

```
int VLSqueuedRequest(
 unsigned char *license_system
 unsigned char *publisher_name,
 unsigned char *product_name,
 unsigned char) *version,
```

```

unsigned long *units_reqd,
unsigned char *log_comment,
LS_CHALLENGE *challenge,
LS_HANDLE *lshandle,
VLSQueuePreference *qPreference,
int requestFlag);

```

```

int VLSQueuedRequestExt(
 unsigned char *license_system,
 unsigned char *publisher_name,
 unsigned char *product_name,
 unsigned char) *version,
 unsigned long *units_reqd,
 unsigned char *log_comment,
 LS_CHALLENGE *challenge,
 LS_HANDLE *lshandle,
 VLSQueuePreference *qPreference,
 int requestFlag,
 VLSserverInfo 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 exists and it is possible to reserve those units, but no units are actually consumed at this time. The default is 1, and this value is used if a NULL value is passed. |

| 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 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"><li>• VLS_REQ_GET - specifies a non-queuing request (without queuing the client). If license is not available, client will not be queued.</li><li>• VLS_REQ_QUEUE - specifies to queue the client (without returning with the license). Even if license is available, client will be queued.</li></ul> <p>If both are specified, the client requests the license server to give the license, if available, otherwise to 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 call 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 license token when (and if) the license is available, put the client in the queue when the license is not

available, or issue 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:

**VLSQueuedRequest and VLSQueuedRequestExt Error Codes**

| Error Code                 | Description                                                                                                                                                                                                                                                                  |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR          | <ul style="list-style-type: none"><li>• <i>request_flag</i> specifies queuing but <i>qPreference</i> is NULL.</li><li>• <i>lshandle</i> is NULL.</li><li>• <i>challenge</i> argument is non-NULL, but cannot be understood.</li><li>• <i>publisherName</i> is NULL</li></ul> |
| VLS_APP_UNNAMED            | <ul style="list-style-type: none"><li>• <i>product_name</i> is NULL</li><li>• <i>version</i> is NULL</li></ul>                                                                                                                                                               |
| VLS_NO_LICENSE_GIVEN       | <ul style="list-style-type: none"><li>• <i>units_reqd</i> is zero.</li><li>• Invalid handle specified.</li><li>• Generic error indicating that the license is not granted.</li></ul>                                                                                         |
| 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, version and capacity.                                                                                                                                                                                 |
| 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.                                                                                      |



**VLSqueuedRequest and VLSqueuedRequestExt Error Codes (Continued)**

| Error Code                  | Description                                                                                                                                                   |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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.                                                                                   |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetQueuedClientInfo

Syntax

```
int VLSgetQueuedClientInfo(
 unsigned char *feature_name,
 unsigned char *version,
 int index,
 VLSqueuedClientInfo *client_info);
```

| Argument            | Description                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i> | Feature name of the client for which we are requesting information.                                                                        |
| <i>version</i>      | <i>Version</i> for which licenses are requested. Must be unique, for the associated feature.                                               |
| <i>index</i>        | <i>Index</i> 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:

### VLSgetQueuedClientInfo Error Codes

| Error Code        | Description                                                                                                                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR | <ul style="list-style-type: none"><li><i>client_info</i> parameter is NULL.</li><li><i>index</i> is negative.</li><li>Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.</li></ul> |
| VLS_APP_UNNAMED   | <ul style="list-style-type: none"><li><i>feature_name</i> is NULL</li><li><i>version</i> is NULL</li></ul> Both feature and version cannot be NULL                                                                                     |

**VLSgetQueuedClientInfo Error Codes**

| Error Code                  | Description                                                                                                                         |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| VLS_NO_LICENSE_GIVEN        | Finished retrieving client information for all the clients.                                                                         |
| VLS_NO_SUCH_FEATURE         | License server does not have a license that matches requested feature, version and capacity.                                        |
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSremoveQueuedClient

**Syntax**

```
int VLSremoveQueuedClient(
 unsigned char *feature_name
 unsigned char *version
 int qkey_id);
```

| Argument            | Description                                                         |
|---------------------|---------------------------------------------------------------------|
| <i>feature_name</i> | Feature name of the client for which we are requesting information. |
| <i>version</i>      | <i>Version</i> for which licenses are requested.                    |
| <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:

**VLSremoveQueuedClient Error Codes**

| Error Code        | Description                                                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR | <ul style="list-style-type: none"><li>• <i>qkey_id</i> parameter cannot be negative.</li><li>• Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.</li></ul> |

**VLSremoveQueuedClient Error Codes (Continued)**

| Error Code                | Description                                                                                                                                                 |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_APP_UNNAMED           | <ul style="list-style-type: none"> <li><i>feature_name</i> is NULL</li> <li><i>version</i> is NULL</li> </ul> Both feature name and version cannot be 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.                                                                                                                      |
| 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.                                                                                 |
| 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, “Sentinel LM Error and Result Codes,” on page 431.

## VLSremoveQueue

### Syntax

```
int VLSremoveQueue(
 unsigned char *feature_name
 unsigned char *version);
```

| Argument            | Description                                                      |
|---------------------|------------------------------------------------------------------|
| <i>feature_name</i> | Identifies the license whose queue needs to be removed.          |
| <i>version</i>      | <i>Version</i> 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:

#### VLSremoveQueue Error Codes

| Error Code                | Description                                                                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR         | Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.                                                       |
| VLS_APP_UNNAMED           | <ul style="list-style-type: none"><li>• <i>feature_name</i> is NULL</li><li>• <i>version</i> is NULL</li></ul> Both feature name and version cannot be 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.                                                                  |

**VLSremoveQueue Error Codes (Continued)**

| Error Code             | Description                                                                                   |
|------------------------|-----------------------------------------------------------------------------------------------|
| 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, “Sentinel LM Error and Result Codes,” on page 431.

**VLSgetHandleStatus****Syntax**

```
int VLSgetHandleStatus(
 LS_Handle lshandle);
```

| Argument        | Description                                                    |
|-----------------|----------------------------------------------------------------|
| <i>lshandle</i> | Identifies the handle previously returned by VLSQueuedRequest. |

**Description**

Reports the current status of the handle.

**Returns**

Returns the following error codes:

**VLSgetHandleStatus Error Codes**

| Error Code           | Description                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------|
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSupdateQueuedClient

**Syntax**

```
int VLSupdateQueuedClient(
 LS_HANDLE lshandle ,
 long *absExpiryTime ,
 unsigned char *unused1 ,
 LS_CHALLENGE *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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <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 VLSgetQueuedLicense. 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 VLSgetQueuedLicense.<br>Possible values for <i>absExpiryTime</i> are: <ul style="list-style-type: none"><li>• Zero = license is not available.</li><li>• Non-zero = license is available and will stop calling the API.</li></ul> |
| <i>unused1</i><br><i>unused2</i> | Uses NULL as the value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**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 clients need 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:

**VLSupdateQueuedClient Error Codes**

| Error Code               | Description                                                                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR        | <ul style="list-style-type: none"><li>• <i>absExpiryTime</i> is NULL.</li><li>• Handle cannot be active.</li><li>• <i>challenge</i> argument is non-NULL, but cannot be understood.</li></ul> |
| LS_BADHANDLE             | Invalid handle.                                                                                                                                                                               |
| 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, version and capacity.                                                                                                  |
| 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.       |

VLSupdateQueuedClient Error Codes (Continued)

| Error Code             | Description                                                                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetQueuedLicense

Syntax

```
int VLSgetQueuedLicense(
LS_HANDLE lshandle,
unsigned char *log_comment,
LS_CHALLENGE *challenge);
```

| Argument           | Description                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lshandle</i>    | The handle previously returned by VLSqueuedRequest. 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.                                                                                                                                              |

| Argument         | Description                                                                                                                       |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>challenge</i> | The challenge-response for this operation. Pointer to a <i>challenge</i> structure. The challenge-response will also be returned. |

**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:

#### VLSgetQueuedLicense Error Codes

| Error Code               | Description                                                                                  |
|--------------------------|----------------------------------------------------------------------------------------------|
| 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, version and capacity. |
| 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.                                              |

**VLSgetQueuedLicense Error Codes (Continued)**

| Error Code             | Description                                                                                                                                                                             |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 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.                                                                                                             |
| 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, “Sentinel LM Error and Result Codes,” on page 431.

## VLSinitQueuePreference

**Syntax**

```
int VLSinitQueuePreference(
VLSqueuePreference *qPreference);
```

| Argument           | Description                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>qPreference</i> | Pointer to the VLSqueuePreference 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. For more details read through “VLSqueuePreference Struct” on page 297.

**Returns**

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

| Error Code        | Description                 |
|-------------------|-----------------------------|
| VLS_CALLING_ERROR | <i>qPreference</i> is NULL. |

For a complete list of the error codes, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.



# Chapter 7

---

## Commuter License API

Commuter licensing is the capability to temporarily check out an authorization to use a protected application from a Sentinel LM 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:

**Commuter License Related Functions**

| Function                      | Description                                                                                                                     |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| VLSCommuterInfo               | Commuter information structure                                                                                                  |
| VLSSetCommuterInfo            | Returns the commuter license information.                                                                                       |
| VLSSetAndInstallCommuter-Code | Obtains the commuter code from the license server and issues the commuter authorization to the client side persistence database |
| VLSSetAndReturnCommuterCode   | Removes the commuter authorization from the client side persistence database and returns the token to the license server.       |

**Commuter License Related Functions (Continued)**

| Function               | Description                                                |
|------------------------|------------------------------------------------------------|
| VLSgetMachineIDString  | Obtains commuter locking code from a remote computer.      |
| VLSgetCommuterCode     | Checks out a commuter authorization for a remote computer. |
| VLSinstallCommuterCode | Install a commuter authorization on a remote computer.     |

**VLSCommuterInfo**

**Syntax**

```
{
 int commuter_code_version;
 int codegen_version;
 char feature_name[VLS_MAXFEALEN];
 char feature_version[VLS_MAXVERLEN];
 int birth_day;
 int birth_month;
 int birth_year;
 int death_day;
 int death_month;
 int death_year;
 int num_of_licenses;
 int locking_crit;
 char lock_info[VLS_MAXCLLOCKLEN];
 char vendor_info[VLS_VENINFOLEN + 1];
 char issuing_server[MAX_NAME_LEN];
 long key_life_time;
 int protocol_type;
 int status;
}VLSCommuterInfo;
```



| Argument                     | Description                                                                                 |
|------------------------------|---------------------------------------------------------------------------------------------|
| <i>commuter_code_version</i> | Version of commuter code                                                                    |
| <i>codegen_version</i>       | Version of the code generator used                                                          |
| <i>feature_name</i>          | Name of the feature                                                                         |
| <i>feature_version</i>       | Version of the feature                                                                      |
| <i>birth_day</i>             | Start day (1-31)                                                                            |
| <i>birth_month</i>           | Start month (1-12)                                                                          |
| <i>birth_year</i>            | Start year                                                                                  |
| <i>death_day</i>             | End day (1-31)                                                                              |
| <i>death_month</i>           | End month (1-12)                                                                            |
| <i>death_year</i>            | End year                                                                                    |
| <i>num_of_licenses</i>       | Number of licenses                                                                          |
| <i>locking_crit</i>          | Locking criteria of the client                                                              |
| <i>lock_info</i>             | Locking information of the client                                                           |
| <i>vendor_info</i>           | The vendor-defined information string. Maximum length of this string can be 395 characters. |
| <i>issuing_server</i>        | License checked out from <servername>                                                       |
| <i>key_life_time</i>         | The license lifetime for this feature (in seconds).                                         |
| <i>protocol_type</i>         | Type of protocol used                                                                       |
| <i>status</i>                | <ul style="list-style-type: none"><li>• 1 - Active</li><li>• 0 - Inactive</li></ul>         |

## VLSgetCommuterInfo

**Syntax**

```
int VLSgetCommuterInfo(
 unsigned char *feature_name,
 unsigned char *version,
 int index,
 VLScommuterInfo *commuter_info);
```

| 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 the call will return information about this feature. The call will ignore the *index* argument.
2. If *feature\_name* is NULL, then the call will return information about the *index* feature in the persistence database. The call will ignore the *version* argument.

VLSgetCommuterInfo should be called until it returns 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, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetAndInstallCommuterCode

### Syntax

```
int VLSgetAndInstallCommuterCode(
 unsigned char *feature_name,
 unsigned char *feature_version,
 long *units_reqd,
 int *duration,
 int *lock_mask,
 insigned char *log_comment,
 LS_CHALLENGE *challenge);
```

| Argument               | Description                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i>    | Name of the feature.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>feature_version</i> | Version of the feature.                                                                                                                                                                                                                                                                                                                                                                                                        |
| <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. 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. |

| Argument           | Description                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>duration</i>    | Displays the number of days for which the license has to be checked out. Specify zero (0) for unlimited check-out.                                                                                                                                                                                                                                                                                                  |
| <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.

Before calling `VLSgetAndInstallCommuterCode`, the `VLSgetMachineID` function must be called in order to obtain the supported lock masks, which are passed in the *lock\_mask* parameter.

---

**Note:** `VLSgetAndInstallCommuterCode` can also be used for the explicit check-out of a perpetual license.

---

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

#### **VLSgetAndInstallCommuterCode Error Codes**

| Error Code                     | Description                                                                                                   |
|--------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>VLS_CALLING_ERROR</code> | <ul style="list-style-type: none"><li>• <i>duration</i> is NULL</li><li>• <i>lock_mask</i> is NULL.</li></ul> |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSuninstallAndReturnCommuterCode

### Syntax

```
int VLSuninstallAndReturnCommuterCode(
 unsigned char *feature_name,
 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 LS\_SUCCESS is returned if successful. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

---

**Note:** Note that VLSuninstallAndReturnCommuterCode cannot be used to check in an authorization for a remote user to prevent the remote user from checking in the authorization while continuing to use it remotely.

---

## VLScleanExpiredCommuterCode

### Syntax

```
int VLScleanExpiredCommuterCode(
 unsigned char *feature_name,
```

```
unsigned char *feature_version,
unsigned char *log_comment
unsigned long *unused);
```

| 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. |
| <i>unused</i>          | This parameter is reserved for future use.                                                                                                                 |

**Description** Cleans the expired commuter authorization on a client system (both remote and normal).

**Returns** The status code VLScg\_SUCCESS is returned if successful. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

### Get Commuter Locking Code from Remote Computer (VLSgetMachineIDString)

Returns an encrypted string that contains the fingerprint information based on the locking criteria specified in the call.

Use this call when you are trying to check out a commuter authorization for a remote computer that does not have access to the license server. The computer that will actually use the commuter authorization runs this call and then passes on the string (via e-mail, disk, etc.) to a computer that has access to the license server. The commuter authorization is then checked out and transmitted to a remote user, and locked to the information given by this string.

If the machine that requires the commuter authorization has network access to the license server, then you do not need to use this method. Instead, check out the license using VLSgetAndInstallCommuterCode.

**Syntax**

```
LS_STATUS_CODE VLSgetMachineIDString(
 unsigned long *lock_selector,
```

```
unsigned char *machineIDString,
unsigned long *bufSz);
```

| Argument               | Description                                                                                                                                                                                                                                                                                                                  |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lock_selector</i>   | Bitmask identifying what criteria you would like to be contained in the Machine ID string. See “lock_selector Values” on page 327 for information on the values for this bitmask. If you set this argument to NULL, this API call will use the locking selector information it finds on the computer on which it is running. |
| <i>machineIDString</i> | A string that represents the machine’s locking information. This will be passed on to VLSgetCommuterCode on a computer that can actually check out a commuter authorization from the license server.                                                                                                                         |
| <i>bufSz</i>           | Returns the buffer size of the machineIDString parameter if machineIDString is NULL. Otherwise, specifies the size of the machineIDString parameter.                                                                                                                                                                         |

**Returns**

The status code LS\_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for the failure. Possible errors returned include VLS\_UNABLE\_TO\_GET\_MACHINE\_ID\_STRING.

For a complete list of error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

---

**Note:** If NULL is passed as the locking criteria in the VLSgetMachineIDString call, VLS\_CALLING\_ERROR is returned.

---

***lock\_selector Values***

The value of lock\_selector is a bitmask in which each bit selects a fingerprinting element. It does not describe the fingerprint, but only designates the locking criteria that will be used to compute the fingerprint. The masks which define each locking criterion are listed below:

|                  |     |
|------------------|-----|
| VLS_LOCK_ID_PROM | 0x1 |
| VLS_LOCK_IP_ADDR | 0x2 |
| VLS_LOCK_DISK_ID | 0x4 |

|                        |       |
|------------------------|-------|
| VLS_LOCK_HOSTNAME      | 0x8   |
| VLS_LOCK_ETHERNET      | 0x10  |
| VLS_LOCK_NW_IPX        | 0x20  |
| VLS_LOCK_NW_SERIAL     | 0x40  |
| VLS_LOCK_PORTABLE_SERV | 0x80  |
| VLS_LOCK_CUSTOM        | 0x100 |
| VLS_LOCK_PROCESSOR_ID  | 0x200 |
| VLS_LOCK_ALL           | 0x3FF |

---

**Note:** VLS\_LOCK\_PORTABLE\_SERV refers to the Computer ID key, and that VLS\_LOCK\_ALL selects all locking criteria.

---

## Checking Out a Remote Authorization (VLSgetCommuterCode)

Obtains a commuter authorization from the license server to be passed on to a remote client that does not have network access to the license server. This call checks out a commuter authorization for another machine. It requires a commuter locking code string from the VLSgetMachineIDString call used on the remote computer. After successful completion of the call, the authorization code string should be passed on to the remote computer which will use VLSinstallCommuterCode to install the authorization.

If the machine that requires the commuter authorization has network access to the license server, then you should not use this call. Instead, check out the license using VLSgetAndInstallCommuterCode. Once a commuter authorization is checked out for a remote computer, it cannot be checked back in until the commuter authorization expires.

### Syntax

```
LS_STATUS_CODE VLSgetCommuterCode(
 unsigned char *feature_name,
 unsigned char *feature_version,
 unsigned long *units_rqd,
 unsigned long *duration,
 unsigned long *lock_mask,
 unsigned char *log_comment,
 unsigned char *machineIDString,
 unsigned char *commuter_code,
```



```

LS_CHALLENGE *challenge ,
VLSserverInfo *requestInfo ,
VLSserverInfo *commuterInfo ,
unsigned long *reserved1);

```

| Argument               | Description                                                                                                                                                                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i>    | The feature name of the license to check out from the license server.                                                                                                                                                                               |
| <i>feature_version</i> | The feature version of the license to check out from the license server.                                                                                                                                                                            |
| <i>units_reqd</i>      | Number of units required for the license.                                                                                                                                                                                                           |
| <i>duration</i>        | Number of days the commuter authorization will last. This value may be superseded by the maximum limit allowed by the license.                                                                                                                      |
| <i>lock_mask</i>       | The desired locking criteria for the client machine. The value here should be equal to or a subset of the value used by the VLSgetMachineIDString call. This value will return the actual locking criteria used to lock the commuter authorization. |
| <i>log_comment</i>     | A comment which will be placed inside the log file on the license server.                                                                                                                                                                           |
| <i>machineIDString</i> | Machine ID string generated by the remote computer desiring the commuter authorization.                                                                                                                                                             |
| <i>commuter_code</i>   | The actual commuter authorization code. This string should be passed on to the remote computer desiring the commuter authorization for installation.                                                                                                |
| <i>challenge</i>       | A challenge and response for the given license on the license server. Set to NULL if you are not using this feature.                                                                                                                                |
| <i>requestInfo</i>     | Reserved. Use NULL for this value.                                                                                                                                                                                                                  |
| <i>commuterInfo</i>    | To be used in future for hooks.                                                                                                                                                                                                                     |
| <i>reserved1</i>       | Reserved. Use NULL for this value.                                                                                                                                                                                                                  |

**Returns** The status code `LS_SUCCESS` is returned if successful. Otherwise, a specific error code is returned indicating the reason for the failure. Possible error codes that can be returned by this call include `VLS_INVALID_MACHINEID_STRING`.

For a complete list of error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**Installing a Remote Commuter Authorization  
(VLSinstallCommuterCode)**

Installs a commuter authorization onto a remote computer. A computer that has network access to the license server should generate the commuter authorization using `VLSgetCommuterCode` (see “Get Commuter Locking Code from Remote Computer (`VLSgetMachineIDString`)” on page 326). The commuter authorization is then passed on to the computer requiring the authorization and installed using `VLSinstallCommuterCode`. After successful completion of this call, the remote computer should be able to use the commuter authorization.

If the machine that requires the commuter authorization has network access to the license server, then you do not need to use this call. Instead, check out the commuter authorization using `VLSgetAndInstallCommuterCode`. Once a commuter authorization is checked out for a remote computer, it cannot be checked back in—it simply expires.

**Syntax**

```
LS_STATUS_CODE VLSinstallCommuterCode (
 unsigned char *commuter_code,
 unsigned char *reserved1,
 unsigned long *reserved2);
```

| Argument             | Description                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------|
| <i>commuter_code</i> | The commuter authorization that was generated by a computer with network access to the license server. |
| <i>reserved1</i>     | Reserved. Use NULL for this value.                                                                     |
| <i>reserved2</i>     | Reserved. Use NULL for this value.                                                                     |

**Returns** The status code `LS_SUCCESS` is returned if successful. Otherwise, a specific error code is returned indicating the reason for the failure. Possible errors that can be returned by this call include `VLS_UNABLE_TO_INSTALL_COMMUTER_CODE`.

For a complete list of error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.



# Chapter 8

---

## Capacity License API

As the name suggests, the capacity license feature defines the capacity of a license. A capacity license is identified by feature name, version and capacity. The license request is granted on the basis of feature name, version and capacity. Capacity licensing in Sentinel LM allows multiple license of same feature, version and different capacity to exist on the same Sentinel LM license server. For examples of capacity licensing and more information on this feature, see the *Sentinel LM Developer's Guide*.

---

**Note:** Capacity Licensing is available through APIs only and is not supported by Sentinel LM-Shell.

---

### Capacity License Related Functions

The following table summarizes the capacity license related functions:

**Capacity License Related Functions**

| Function             | Description                                                              |
|----------------------|--------------------------------------------------------------------------|
| VLSrequestExt2       | Supports capacity and non-capacity requests                              |
| VLSgetFeatureInfoExt | Tracks the features available on the server                              |
| VLSgetCapacityList   | Returns the list of all the capacity for particular feature and version. |

Capacity License Related Functions (Continued)

| Function                 | Description                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------|
| VLSgetClientInfoExt      | Returns the list of all clients running for a particular feature, version, and capacity |
| VLSdeleteFeatureExt      | Deletes a license from the server based on feature, version and capacity                |
| VLSgetCapacityFromHandle | Returns the team capacity and user capacity allocated to a handle                       |
| VLSsetTeamId             | Redefines team ID functions                                                             |
| VLSsetTeamIdValue        | Registers a customized team ID value                                                    |

VLSrequestExt2

Syntax

```
VLSrequestExt2 (
 unsigned char *license_system,
 unsigned char *publisher_name,
 unsigned char *product_name,
 unsigned char *version,
 unsigned long *units_reqd,
 unsigned char *log_comment,
 LS_CHALLENGE *challenge,
 LS_HANDLE *lshandle,
 VLSserverInfo *serverInfo,
 unsigned long *team_capacity_reqd,
 unsigned long *capacity_reqd,
 unsigned char *unused1,
 unsigned long *special_flag);
```

| Argument              | Description                                                                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>license_system</i> | Unused. <ul style="list-style-type: none"><li>• Use LS_ANY as the value of this variable.</li><li>• LS_ANY is specified to indicate a match against the installed license system.</li></ul>         |
| <i>publisher_name</i> | <ul style="list-style-type: none"><li>• A string identifying the publisher of the product. Limited to 32 characters and cannot be NULL.</li><li>• Company name and trademark may be used.</li></ul> |

| Argument            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>product_name</i> | <ul style="list-style-type: none"><li>• Name of the feature for which a license code is requested.</li><li>• May consist of any printable characters and cannot be NULL.</li><li>• Limited to 24 characters.</li></ul>                                                                                                                                                                                                                                                                                                                                         |
| <i>version</i>      | <ul style="list-style-type: none"><li>• Version of the feature for which a license code is requested.</li><li>• May consist of any printable characters. Limited to 11 characters.</li><li>• Version can be NULL.</li></ul>                                                                                                                                                                                                                                                                                                                                    |
| <i>units_reqd</i>   | <ul style="list-style-type: none"><li>• The number of licenses required. The license server verifies that the number of units exist and may reserve those units. The number of available units is returned.</li><li>• 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>units_reqd</i>. If <i>units_reqd</i> is NULL, a value of 1 unit is assumed.</li><li>• To use the capacity licensing it is necessary that units required be always 1.</li></ul> |
| <i>log_comment</i>  | <ul style="list-style-type: none"><li>• A string to be written by the license server to the comment field of the usage log file.</li><li>• Pass a NULL value for this argument if no log comment is desired.</li></ul>                                                                                                                                                                                                                                                                                                                                         |
| <i>challenge</i>    | <ul style="list-style-type: none"><li>• The challenge structure. If challenge-response mechanism is not being used, this pointer must be NULL.</li><li>• The response to the challenge is provided in the same structure, provided a license was issued, i.e., provided the function <code>VLSrequestExt2</code> returns <code>LS_SUCCESS</code>.</li></ul>                                                                                                                                                                                                    |

| Argument                  | Description                                                                                                                                                                                                                                                                                                                           |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lshandle</i>           | <ul style="list-style-type: none"><li>• The handle for this request is returned in <i>lshandle</i>. This handle must be used to later update and release this license code.</li><li>• A client can have more than one handle active at a time.</li><li>• Space for <i>lshandle</i> must be allocated by the caller.</li></ul>         |
| <i>serverInfo</i>         | <ul style="list-style-type: none"><li>• This information is passed to the license server for use in server hook functions.</li><li>• <code>VLSinitServerInfo</code> must be called to initialize <i>serverInfo</i>.</li></ul>                                                                                                         |
| <i>team_capacity_reqd</i> | <ul style="list-style-type: none"><li>• Required team capacity</li><li>• If the server does not have the requested capacity this field will return the team capacity available with the server for this feature and version.</li><li>• If the request is made for a non-capacity license, this must be passed as NULL.</li></ul>      |
| <i>capacity_reqd</i>      | <ul style="list-style-type: none"><li>• Required user capacity</li><li>• If the server does not have the requested user capacity, this field will return the user capacity available with the server for this feature and version.</li><li>• If the request is made for a non-capacity license this must be passed as NULL.</li></ul> |
| <i>unused1</i>            | Reserved for future use.                                                                                                                                                                                                                                                                                                              |



| Argument            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>special_flag</i> | <p>An IN/OUT parameter with the following possible values:</p> <ul style="list-style-type: none"><li>• The IN value can be VLS_IGNORE_GRACE_ERROR (default) or VLS_NOTIFY_GRACE_ERROR.</li></ul> <p>When the flag is set to VLS_IGNORE_GRACE_ERROR, any error encountered during the installation of a grace license will be ignored and the VLSrequestExt2 function will succeed.</p> <p>When the flag is set to VLS_NOTIFY_GRACE_ERROR, any error encountered during the installation of a grace license will make the VLSrequestExt2 function fail (and the application will not run).</p> <ul style="list-style-type: none"><li>• The OUT value will be the grace period-related error code.</li></ul> <p>VLS_IGNORE_GRACE_ERROR will be applicable to the other variants of the request API functions (like LSrequest, VLSrequestExt, and so on).</p> |

**Description** Supports capacity as well as non-capacity requests.

If the request is denied due to either insufficient team capacity or user capacity then accordingly the *capacity\_reqd* or *team\_capacity\_reqd* field should contain the available capacity.

VLSrequestExt2 must be used whenever the user wishes to use the capacity feature in a license. The call can also be used to obtain a token from normal license.

If the developer wishes to override any of the default user information passed to the license server, he would be using the VLSsetTeamId/ VLSsetTeamIdValue APIs.

The following information is sent by the client library as user identification information:

- User Name
- Host Name
- X-Display name
- Vendor defined string.

This information is used by the server when it manages or creates teams. VLSsetTeamId/ VLSsetTeamIdValue needs to be called before calling the request API so that it can pass the correct information about the user name etc. to the license server.

Lets consider a possible scenario to interpret the above:

Say we initialize the license system as:

```
int team_id = 1; /* Override username information */
int units_reqd = 1; /* Should always be 1 if using
capacity request*/
unsigned long team_capacity = 1000; /* Say*/
unsigned long user_capacity = 800; /* Cannot be greater
than team
LS_STATUS_CODE ret_val;
 if(VLSinitialize()){
 // Error in initializing SLM library.
 // Do error condition
 }
 VLSsetTeamId(1, "SENTINEL");
```

Here we pass 'SENTINEL' as the user name. So even if the user has logged into the client machine with say "XYZ" user name, the license server would see the request as if it is coming from user "SENTINEL". Now

```
ret_val = VLSrequestExt2(featureName,
version,&units_reqd, &team_capacity, &user_capacity);
 if(ret_val == LS_SUCCESS){
```

```
// Succesfully got the requested token as well as team
and user capacity. Now do further actions based on
these values
}
```

In case you are unable to get a license token.The possible reasons could be:

- Team limit has been exhausted
- User capacity has been exhausted
- Team capacity has been exhausted in case of pooled licenses only.

**Returns**

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

**VLSrequestExt2 Error Codes**

| Error Code              | Description                                                                                                                                                                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_APP_UNNAMED         | <ul style="list-style-type: none"><li>• <i>featureName</i> is NULL</li><li>• <i>version</i> is NULL</li></ul> Both feature name and version cannot be NULL at the same time.                                                                     |
| VLS_CALLING_ERROR       | <ul style="list-style-type: none"><li>• <i>lshandle</i> is NULL</li><li>• <i>challenge argument</i> is non NULL</li><li>• Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.</li></ul>       |
| VLS_NO_LICENSE_GIVEN    | <ul style="list-style-type: none"><li>• <i>unitsReqd</i> is zero</li><li>• <i>lshandle</i> is not a valid handle</li></ul>                                                                                                                       |
| VLS_NO_SUCH_FEATURE     | License server does not have license that matches requested feature, version and capacity.                                                                                                                                                       |
| LS_NOLICENSESAVAILABLE  | All licenses are in use.                                                                                                                                                                                                                         |
| LS_INSUFFICIENTUNITS    | License server does not 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      | <ul style="list-style-type: none"><li>• License server has determined that the client system lock has been modified.</li><li>• The license for this feature has time tampering protection enabled, so the license operation is denied.</li></ul> |

**VLSrequestExt2 Error Codes (Continued)**

| Error Code                     | Description                                                                                                                                               |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 request.                                                               |
| VLS_NO_SERVER_RESPONSE         | Communication with license server has timed out.                                                                                                          |
| VLS_HOST_UNKNOWN               | Invalid <i>hostName</i> was specified.                                                                                                                    |
| VLS_NO_SERVER_FILE             | <ul style="list-style-type: none"> <li>• No license server has been set</li> <li>• Unable to determine which license server to use.</li> </ul>            |
| VLS_BAD_SERVER_MESSAGE         | Message from 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 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.                                  |
| VLS_INSUFFICIENT_TEAM_CAPACITY | License server does not currently have sufficient team capacity available.                                                                                |

VLSrequestExt2 Error Codes (Continued)

| Error Code                     | Description                                                                                     |
|--------------------------------|-------------------------------------------------------------------------------------------------|
| VLS_INSUFFICIENT_USER_CAPACITY | License server does not currently have sufficient user capacity available for this team member. |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSgetFeatureInfoExt

Syntax

```
LS_STATUS_CODE VLSgetFeatureInfoExt(
 unsigned char *feature_name,
 unsigned char *version,
 unsigned long *capacity,
 int index,
 char *unused1,
 long *unused2,
 VLSfeatureInfo feature_info);
```

| Argument            | Description                                                                             |
|---------------------|-----------------------------------------------------------------------------------------|
| <i>feature_name</i> | Name of the feature.                                                                    |
| <i>version</i>      | Version of the feature.                                                                 |
| <i>capacity</i>     | Capacity of the feature.                                                                |
| <i>index</i>        | Used to specify a particular feature.                                                   |
| <i>unused1</i>      | Use NULL as value.                                                                      |
| <i>unused2</i>      | Use NULL as value.                                                                      |
| <i>feature_info</i> | The structure in which information will be returned. Space must be allocated by caller. |

Description

Returns the information of features available on the server.

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

- If information about a non-capacity license is desired, capacity should be passed as NULL and feature must be non-NULL.
- If information about all licensed features (capacity as well as non-capacity) is desired, feature name should be NULL, and index should be used in a loop.

**Returns**

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

**VLSgetFeatureInfoExt Error Codes**

| Error Code             | Description                                                                                                                                                                                                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_CALLING_ERROR      | <ul style="list-style-type: none"> <li>• <i>featureinfo</i> is NULL</li> <li>• <i>index</i> is negative</li> <li>• Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.</li> </ul> |
| VLS_APP_UNNAMED        | Version is NULL when name 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.                                                                                                                                                               |
| VLS_NO_SUCH_FEATURE    | License server does not have license that matches requested feature, version and capacity.                                                                                                                                           |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetCapacityList

**Syntax**

```
VLSgetCapacityList(
 ifndef LSNOPRONT0
 unsigned char LSFAR *feature_name,
 unsigned char LSFAR *feature_version,
 int LSFAR *index,
 unsigned long LSFAR *bufferSize,
 char LSFAR *capacityList,
 char LSFAR *log_comment,
 unsigned long LSFAR *unused2
#endif);
```

| Argument               | Description                                                                                                                                        |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>feature_name</i>    | Name of the feature.                                                                                                                               |
| <i>feature_version</i> | Version of the feature.                                                                                                                            |
| <i>index</i>           | Returns the index of the license up to which the capacity has been retrieved based on the bufferSize                                               |
| <i>bufferSize</i>      | Specifies the size of <i>capacityList</i> .                                                                                                        |
| <i>capacityList</i>    | An array containing a list of all the capacities available for this feature and version, separated by space. The caller should allocate the space. |
| <i>log_comment</i>     | Use NULL as value.                                                                                                                                 |
| <i>unused1</i>         | Use NULL as value.                                                                                                                                 |

**Description**

Returns the list of all the capacities of all the licenses having specified feature and version but different capacity. This function returns list of capacities as one string, each capacity separated by a space character.

If capacityList is passed as NULL, the API returns the buffersize required. VLSgetCapacityList returns an error if the license is a non-capacity license. For example if Sentinel LM license server has following licenses:

- Feature F1, version V1, capacity 500



- Feature F1, version V1, capacity 1000
- Feature F1, version V1, capacity 1500

Then this API would return "500 1000 5000" as the output string in "capacity\_list".

For a discussion of pooled versus non-pooled capacity licenses, refer to the *Sentinel LM Developer's Guide*.

## Returns

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

### VLSgetCapacityList Error Codes

| Error Code             | Description                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------------------|
| VLS_NO_SUCH_FEATURE    | License server does not have a license that matches the request feature, version and capacity.         |
| 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. |
| 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, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetClientInfoExt

**Syntax**

```
VLSgetClientInfoExt (
 unsigned char *feature_name ,
 unsigned char *version ,
 unsigned long *capacity ,
 int index ,
 char *log_comment ,
 VLSclientInfo *client_info);
```

| Argument            | Description                                                                         |
|---------------------|-------------------------------------------------------------------------------------|
| <i>feature_name</i> | Name of the feature.                                                                |
| <i>version</i>      | Version of the feature.                                                             |
| <i>capacity</i>     | Capacity of the feature.                                                            |
| <i>index</i>        | Used to specify a particular client.                                                |
| <i>log_comment</i>  | Comment.                                                                            |
| <i>client_info</i>  | The structure in which information will be returned. Space allocated by the caller. |

**Description**

Returns the list of all the clients running for a particular feature, version and capacity. If the capacity is specified as NULL, this API shall return the list of all the clients for a particular feature and version.

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.

---

**Note:** Memory for *client\_info* 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:

**VLSgetClientInfoExt Error Codes**

| Error Code                   | Description                                                                                                                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_APP_UNNAMED              | <ul style="list-style-type: none"> <li><i>featureName</i> is NULL.</li> <li><i>version</i> is NULL</li> </ul> Both feature name and version cannot be NULL at the same time.                                                            |
| VLS_CALLING_ERROR            | <ul style="list-style-type: none"> <li><i>clientInfo</i> parameter is NULL</li> <li><i>index</i> is negative</li> <li>Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.</li> </ul> |
| 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, version and capacity.                                                                                                                                            |
| VLS_MULTIPLE_VENDOR_ID_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, “Sentinel LM Error and Result Codes,” on page 431.

VLSdeleteFeatureExt

Syntax

```
VLSdeleteFeatureExt(
 unsigned char *feature_name ,
 unsigned char *version ,
 unsigned long *capacity ,
 unsigned char *log_comment ,
 LS_CHALLENGE *challenge);
```

| Argument            | Description                     |
|---------------------|---------------------------------|
| <i>feature_name</i> | Name of the feature.            |
| <i>version</i>      | <i>Version</i> of the feature.  |
| <i>capacity</i>     | <i>Capacity</i> of the feature. |
| <i>log_comment</i>  | Unused                          |
| <i>challenge</i>    | Unused                          |

Description

Deletes a license from the server based on feature, version and capacity. If the capacity is NULL, this API will delete a non-capacity license for the feature, version specified.

The license is deleted from the server only and not from the license file.

Returns

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

VLSdeleteFeatureExt Error Codes

| Error Code        | Description                                                                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_APP_UNNAMED   | <ul style="list-style-type: none"><li><i>featureName</i> is NULL.</li><li><i>version</i> is NULL</li></ul> Both feature name and version cannot be NULL at the same time. |
| VLS_CALLING_ERROR | Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.                                                                    |

**VLSdeleteFeatureExt Error Codes (Continued)**

| Error Code                  | Description                                                                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLS_NO_SUCH_FEATURE         | License server does not have a license that matches requested feature, version and capacity.                                                                |
| 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 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, “Sentinel LM Error and Result Codes,” on page 431.

## VLSgetCapacityFromHandle

Syntax

```
VLSgetCapacityFromHandle(
 LS_HANDLE lshandle,
 unsigned long LSFAR *team_capacity,
 unsigned long LSFAR *user_capacity
 unsigned long LSFAR *license_capacity);
```

| Argument               | Description                                                    |
|------------------------|----------------------------------------------------------------|
| <i>handle</i>          | Handle                                                         |
| <i>team_capacity</i>   | Team capacity allocated to the handle and issued by the server |
| <i>user_capacity</i>   | User capacity allocated to the handle and issued by the server |
| <i>license_capcity</i> | License capacity allocated to the handle                       |

Description

VLSgetCapacityFromHandle returns the team capacity and user capacity allocated to a handle.

Returns

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

VLSgetCapacityFromHandle Error Codes

| Error Code   | Description            |
|--------------|------------------------|
| LS_BADHANDLE | The handle is invalid. |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

## VLSsetTeamId

See “VLSsetSharedId/ VLSsetTeamId” on page 71.

## VLSsetTeamIdValue

See “VLSsetSharedIdValue/ VLSsetTeamIdValue” on page 73.







# Chapter 9

---

## Upgrade License API

The Sentinel LM upgrade license feature enables you to update your customer's existing license to change the version or/and increase the capacity. A special upgrade license must be created to update the existing license.

### Upgrade License Code Generator API

The following table summarizes the upgrade license code generator related functions:

**Upgrade License Code Generator Related Functions**

| Function              | Description                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------|
| ucodeT Struct         | Contains the values for the upgrade license.                                               |
| VLSucgInitialize      | Initializes the upgrade codegen library                                                    |
| VLSucgCleanup         | Destroys the handle created using VLSucgInitialize                                         |
| VLSucgReset           | Sets all the fields of <i>ucodeT</i> to their default values                               |
| VLSucgGetNumErrors    | Identifies the total number of messages recorded in the handle                             |
| VLSucgGetError Length | Returns the length of error message identified by <i>msgNum</i> and recorded in the handle |

**Upgrade License Code Generator Related Functions (Continued)**

| Function                       | Description                                                                                                             |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| VLSucgGetError Message         | Returns the earliest error from the handle up to bufLen characters                                                      |
| VLSucgPrintError               | Prints the complete info of all the error messages stored in the handle to a file.                                      |
| VLSucgAllowBase FeatureName    | Identifies whether the corresponding VLSucgSetBaseFeatureName should be called or not                                   |
| VLSucgSetBaseFeature Name      | Sets the value of <i>base_feature_name</i> in the <i>ucodeT</i> struct.                                                 |
| VLSucgAllowBase FeatureVersion | Identifies whether the corresponding VLSucgSetBaseFeatureVersion should be called or not.                               |
| VLSucgSetBaseFeature Version   | Sets the value of <i>base_feature_version</i> in the <i>ucodeT</i> struct.                                              |
| VLSucgAllowUpgrade Code        | Identifies whether the corresponding VLSucgSetUpgradeCode API should be called or not                                   |
| VLSucgSetUpgrade Code          | Sets the value of <i>base_lock_code</i> in the <i>ucodeT</i> struct to the value in the <i>upgrade_code</i>             |
| VLSucgAllowUpgrade Flag        | Identifies whether the corresponding VLSucgSetUpgradeFlag should be called or not                                       |
| VLSucgSetUpgrade Flag          | Sets the value of <i>upd_flags</i> in the <i>ucodeT</i> struct.                                                         |
| VLSucgAllowUpgrade Version     | Identifies whether the corresponding VLSucgSetUpgradeVersion should be called or not                                    |
| VLSucgSetUpgrade Version       | Sets the value of <i>upd_version</i> in the <i>ucodeT</i> struct.                                                       |
| VLSucgAllowUpgrade Capacity    | Identifies whether the corresponding VLSucgSetUpgradeCapacityUnits and VLSucgSetUpgradeCapacity should be called or not |
| VLSucgSetUpgrade CapacityUnits | Sets the value of <i>capacity_units</i> in the <i>ucodeT</i> struct.                                                    |
| VLSucgSetUpgrade Capacity      | Sets the value of <i>capacity_increment</i> in the <i>ucodeT</i> struct.                                                |

## Upgrade License Code Generator Related Functions (Continued)

| Function                   | Description                                                                     |
|----------------------------|---------------------------------------------------------------------------------|
| VLSucgGenerateLicense      | Generates the upgrade license string for the given <i>ucodeT</i> struct         |
| VLSucgGetLicenseMeterUnits | Returns the number of license generation units available in the attached dongle |
| VLSGenerateUpgradeLockCode | Allows the user to generate a unique upgrade code for the base license.         |

**ucodeT Struct****Syntax**

```
typedef struct {
 long structSz;
 unsigned int vendor_code;
 unsigned int version_num;
 /* Feature/Version of the base license that needs to be
 upgraded */
 char base_feature_name[VLSucg_MAX_CODE_COMP_LEN+1];
 char base_feature_version[VLSucg_MAX_CODE_COMP_LEN+1];
 char base_lock_code[VLSucg_MAX_CODE_COMP_LEN+1];
 unsigned long generation_time;
 unsigned long generation_sequence;
 unsigned long upd_flags;
 char upd_version[VLSucg_MAX_CODE_COMP_LEN+1];

 /* New version for this feature*/
 int capacity_units;
 unsigned long capacity_increment ;
 unsigned long unused1;
 unsigned long unused2;
} ucodeT;
```

**ucodeT Struct**

| Member             | Description                                     |
|--------------------|-------------------------------------------------|
| <i>structSz</i>    | Size of the structure.                          |
| <i>Vendor_code</i> | Internal use                                    |
| <i>version_num</i> | Upgrade license code generation library version |

**ucodeT Struct (Continued)**

| Member                      | Description                                                                                                                                                                                                |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base_feature_name</i>    | Feature Name of the base license that needs to be upgraded                                                                                                                                                 |
| <i>base_feature_version</i> | Feature Version of the base license that needs to be upgraded                                                                                                                                              |
| <i>lock_code</i>            | A unique code to identify base licenses which needs to be upgraded.                                                                                                                                        |
| <i>generation_time</i>      | This value shall be set automatically during the license generation time in GMT. It details about the time of license generation.                                                                          |
| <i>generation_sequence</i>  | This value shall be set at license generation time along with <i>generation_time</i> to ensure that on a fast system, even if two licenses are generated at the same time, this value should be different. |
| <i>upd_flags</i>            | Bit-wise flag. Will control what will be updated <ul style="list-style-type: none"><li>• VLSucg_UPGRADE_VERSION</li><li>• VLSucg_UPGRADE_CAPACITY</li><li>• VLSucg_UPGRADE_ALL</li></ul>                   |
| <i>upd_version</i>          | New version for this feature.                                                                                                                                                                              |
| <i>capacity_units</i>       | Flag which determines capacity least count                                                                                                                                                                 |
| <i>capacity_increment</i>   | Capacity increment for this feature.                                                                                                                                                                       |
| <i>Unused</i>               | For future use.                                                                                                                                                                                            |
| <i>Unused</i>               | For future use.                                                                                                                                                                                            |

## VLSucgInitialize

**Syntax**

```
int VLSucgInitialize(
VLSucg_HANDLE *iHandle);
```

| Argument       | Description                                                                                      |
|----------------|--------------------------------------------------------------------------------------------------|
| <i>iHandle</i> | The pointer to instance handle for this library, provides access to the internal data structure. |

**Description**

Initializes the upgrade codegen library.

VLSucgInitialize should be called before any other API. VLSucgInitialize returns a unique handle, which is used in all the other API of this library.

**Returns**

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

### VLSucgInitialize Error Codes

| Error Code                    | Description                                                                  |
|-------------------------------|------------------------------------------------------------------------------|
| VLSucg_BAD_HANDLE             | Call VLSucgCleanup to free the resources associated with the invalid handle. |
| VLSucg_MAX_LIMIT_CROSSED      | Library has crossed the limit of maximum handles it can allocate.            |
| VLSucg_LICMETER_NOT_SUPPORTED | Your Sentinel LM License Meter is not supported.                             |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgCleanup

**Syntax**

```
int VLSucgCleanup(
VLSucg_HANDLE *iHandle);
```

| Argument       | Description                      |
|----------------|----------------------------------|
| <i>iHandle</i> | Instance handle for this library |

**Description**

Destroys the handle created using VLSucgInitialize.

VLSucgCleanup cleans up the resources associated with the handle.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLSucgCleanup Error Codes**

| Error Code        | Description                                 |
|-------------------|---------------------------------------------|
| VLSucg_BAD_HANDLE | If the handle passed is not a valid handle. |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLSucgReset**

**Syntax**

```
int VLSucgReset(
 VLSucg_HANDLE iHandle,
 ucodeT *ucodeP);
```

| Argument       | Description                         |
|----------------|-------------------------------------|
| <i>iHandle</i> | Instance handle for this library    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct |

**Description** Sets all the fields of *ucodeT* to their default values. VLSucgReset is used after the Initialize and before the Set and Allow APIs.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLSucgReset Error Codes**

| Error Code           | Description                            |
|----------------------|----------------------------------------|
| VLSucg_INVALID_INPUT | If the <i>ucodeP</i> is passed as NULL |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgGetNumErrors

**Syntax**

```
int VLSucgGetNumErrors(
VLSucg_HANDLE iHandle,
int *numMsgsP);
```

| Argument        | Description                                 |
|-----------------|---------------------------------------------|
| <i>iHandle</i>  | Instance handle for this library            |
| <i>numMsgsP</i> | The number of messages queued to the handle |

**Description**

Identifies the total number of messages recorded in the handle.

**Returns**

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

### VLSucgGetNumErrors Error Codes

| Error Code          | Description                                 |
|---------------------|---------------------------------------------|
| VLSucg_BAD_HANDLE   | If the handle passed is not a valid handle. |
| VLSucg_NO_RESOURCES | If no resources are available.              |
| VLSucg_FAIL         | On failure                                  |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgGetErrorLength

**Syntax**

```
int VLSucgGetErrorLength(
VLSucg_HANDLE iHandle,
int msgNum,
int *errLenP);
```

| Argument       | Description                                                             |
|----------------|-------------------------------------------------------------------------|
| <i>iHandle</i> | Instance handle for this library                                        |
| <i>msgNum</i>  | The number of the message whose length is to be queried, starts from 0. |
| <i>errLenP</i> | The length of messages identified by <i>msgNum</i>                      |

**Description** Returns the length of error message identified by *msgNum* and recorded in the handle.

The length returned by VLSucgGetErrorLength include the space required for NULL termination.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

VLSucgGetErrorLength Error Codes

| Error Code          | Description                                 |
|---------------------|---------------------------------------------|
| VLSucg_BAD_HANDLE   | If the handle passed is not a valid handle. |
| VLSucg_NO_RESOURCES | If no resources are available.              |
| VLSucg_FAIL         | On failure                                  |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.



## VLSucgGetErrorMessage

### Syntax

```
int VLSucgGetErrorMessage(
 VLSucg_HANDLE iHandle ,
 char *msgBuf ,
 int bufLen);
```

| Argument       | Description                                                             |
|----------------|-------------------------------------------------------------------------|
| <i>iHandle</i> | 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

Returns the earliest error from the handle up to *bufLen* characters.

- The *bufLen* must be the length of the pre allocated buffer *msgBuf*.
- The message returned should always be NULL terminated.

### Returns

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

#### VLSucgGetErrorMessage Error Codes

| Error Code          | Description                                 |
|---------------------|---------------------------------------------|
| VLSucg_BAD_HANDLE   | If the handle passed is not a valid handle. |
| VLSucg_NO_RESOURCES | If no resources are available.              |
| VLSucg_FAIL         | On Failure                                  |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgPrintError

**Syntax**

```
int VLSucgPrintError(
VLSucg_HANDLE iHandle,
FILE *file);
```

| Argument       | Description                      |
|----------------|----------------------------------|
| <i>iHandle</i> | Instance handle for this library |
| <i>file</i>    | File pointer                     |

**Description** Prints the complete info of all the error messages stored in the handle to a file.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

VLSucgPrintError Error Codes

| Error Code          | Description                                 |
|---------------------|---------------------------------------------|
| VLSucg_BAD_HANDLE   | If the handle passed is not a valid handle. |
| VLSucg_NO_RESOURCES | If no resources are available.              |
| VLSucg_FAIL         | On Failure                                  |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgAllowBaseFeatureName

### Syntax

```
Int VLSucgAllowFeatureName(
VLSucg_HANDLE iHandle ,
ucodeT *ucodeP);
```

| Argument       | Description                         |
|----------------|-------------------------------------|
| <i>iHandle</i> | Instance handle for this library    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct |

### Description

Identifies whether the corresponding VLSucgSetBaseFeatureName should be called or not.

If the VLSucgAllowBaseFeatureName returns 1 only then the corresponding VLSucgSetBaseFeatureName should be called.

### Returns

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

#### VLSucgAllowBaseFeatureName Error Codes

| Error Code | Description                              |
|------------|------------------------------------------|
| 1          | VLSucgSetBaseFeatureName is allowed.     |
| 0          | VLSucgSetBaseFeatureName is not allowed. |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgSetBaseFeatureName

Syntax

```
int VLSucgSetBaseFeatureName(
 VLSucg_HANDLE iHandle,
 ucodeT *ucodeP,
 char *feature_name);
```

| Argument            | Description                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i>      | Instance handle for this library                                                                                     |
| <i>ucodeP</i>       | The pointer to <i>ucodeT</i> struct                                                                                  |
| <i>feature_name</i> | <ul style="list-style-type: none"><li>Any printable ASCII text except #.</li><li>Maximum of 24 characters.</li></ul> |

Description

Sets the value of *base\_feature\_name* in the *ucodeT* struct.

This function also checks the input variables for their validity and boundary conditions.

Returns

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

VLSucgSetBaseFeatureName Error Codes

| Error Code                            | Description                                                                          |
|---------------------------------------|--------------------------------------------------------------------------------------|
| <code>VLSucg_INVALID_CHARS</code>     | Invalid characters in <i>feature_name</i> .                                          |
| <code>VLSucg_NO_FEATURE_NAME</code>   | If <i>feature_name</i> is NULL.                                                      |
| <code>VLSucg_RESERV_STR_ERROR</code>  | If the <i>feature_name</i> is a reserved string.                                     |
| <code>VLSucg_EXCEEDS_MAX_VALUE</code> | If the length of <i>feature_name</i> string exceeds maximum allowed length(24 char). |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgAllowBaseFeatureVersion

**Syntax**

```
int VLSucgAllowBaseFeatureVersion(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP);
```

| Argument       | Description                         |
|----------------|-------------------------------------|
| <i>iHandle</i> | Instance handle for this library    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct |

**Description**

Identifies whether the corresponding VLSucgSetBaseFeatureVersion should be called or not.

If the VLSucgAllowBaseFeatureVersion returns 1 only then the corresponding VLSucgSetBaseFeatureVersion should be called.

**Returns**

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

### VLSucgAllowBaseFeatureVersion Error Codes

| Error Code | Description                                 |
|------------|---------------------------------------------|
| 1          | VLSucgSetBaseFeatureVersion is allowed.     |
| 0          | VLSucgSetBaseFeatureVersion is not allowed. |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgSetBaseFeatureVersion

Syntax

```
int VLSucgSetBaseFeatureVersion(
 VLSucg_HANDLE iHandle,
 ucodeT *ucodeP,
 char *feature_version);
```

| Argument               | Description                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i>         | Instance handle for this library                                                                                     |
| <i>ucodeP</i>          | The pointer to <i>ucodeT</i> struct                                                                                  |
| <i>feature_version</i> | <ul style="list-style-type: none"><li>Any printable ASCII text except #.</li><li>Maximum of 11 characters.</li></ul> |

Description

Sets the value of *base\_feature\_version* in the *ucodeT* struct. This function checks the input variables for their validity and boundary conditions.

Returns

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

### VLSucgSetBaseFeatureVersion Error Codes

| Error Code               | Description                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------|
| VLSucg_INVALID_CHARS     | If <i>feature_version</i> characters are not printable.                                 |
| VLSucg_RESERV_STR_ERROR  | If the <i>feature_version</i> is a reserved string.                                     |
| VLSucg_EXCEEDS_MAX_VALUE | If the length of <i>feature_version</i> string exceeds maximum allowed length(11 char). |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgAllowUpgradeCode

**Syntax**

```
int VLSucgAllowUpgradeCode(
VLSucg_HANDLE iHandle ,
ucodeT *ucodeP) ;
```

| Argument       | Description                         |
|----------------|-------------------------------------|
| <i>iHandle</i> | Instance handle for this library    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct |

**Description**

Identifies whether the corresponding VLSucgSetUpgradeCode should be called or not.

Only if the VLSucgAllowUpgradeCode returns 1 then the corresponding VLSucgSetUpgradeCode should be called.

**Returns**

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

### VLSucgAllowUpgradeCode Error Codes

| Error Code | Description                         |
|------------|-------------------------------------|
| 1          | VLSucgSetUpgradeCode is allowed.    |
| 0          | VLSucgSetUpgradeCode is not allowed |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgSetUpgradeCode

**Syntax**

```
int VLSucgSetUpgradeCode(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP,
char *upgrade_code);
```

| Argument            | Description                          |
|---------------------|--------------------------------------|
| <i>iHandle</i>      | Instance handle for this library.    |
| <i>ucodeP</i>       | The pointer to <i>ucodeT</i> struct. |
| <i>upgrade_code</i> | Upgrade code of base license.        |

**Description**

Sets the value of the *lock\_code* variable in the *ucodeT* struct.

This function checks the input variables for their validity and boundary conditions. However, this function does not checks the validity of upgrade code.

---

**Note:** All the validations and matching of base license information with upgrade license information will be done in VLSucgGenerateLicense.

---

**Returns**

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

**VLSucgSetUpgradeCode Error Codes**

| Error Code               | Description                                                                 |
|--------------------------|-----------------------------------------------------------------------------|
| VLSucg_INVALID_INPUT     | If ucodeP is passed as NULL.                                                |
| VLSucg_NO_UPGRADE_CODE   | If the upgrade_code is passed as NULL or empty string.                      |
| VLSucg_EXCEEDS_MAX_VALUE | If the length of <i>upgrade_code</i> string exceeds maximum allowed length. |
| VLSucg_FAIL              | On Failure.                                                                 |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.



## VLSucgAllowUpgradeFlag

**Syntax**

```
int VLSucgAllowUpgradeFlag(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP);
```

| Argument       | Description                          |
|----------------|--------------------------------------|
| <i>iHandle</i> | Instance handle for this library.    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct. |

**Description**

Indicates whether the corresponding VLSucgSetUpgradeFlag should be called or not.

If the VLSucgAllowUpgradeFlag returns 1 only then the corresponding VLSucgSetUpgradeFlag should be called.

**Returns**

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

### VLSucgAllowUpgradeFlag Error Codes

| Error Code | Description                      |
|------------|----------------------------------|
| 1          | Capacity Upgrade is allowed.     |
| 0          | Capacity Upgrade is not allowed. |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgSetUpgradeFlag

Syntax

```
int VLSucgSetUpgradeFlag(
 VLSucg_HANDLE iHandle,
 ucodeT *ucodeP,
 char *flag);
```

| Argument       | Description                                                                                                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i> | Instance handle for this library.                                                                                                                                                                                                                                  |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct.                                                                                                                                                                                                                               |
| <i>flag</i>    | The value of <i>flag</i> is used to set the <i>upd_flags</i> of <i>ucodeT</i> struct. Legal values are bit combinations of <ul style="list-style-type: none"><li>• VLSucg_UPGRADE_VERSION</li><li>• VLSucg_UPGRADE_CAPACITY</li><li>• VLSucg_UPGRADE_ALL</li></ul> |

Description

Sets the value of *upd\_flags* in the *ucodeT* struct. This function also checks the input variables for their validity and boundary conditions.

- If the flag value is VLSucg\_UPGRADE\_VERSION then only version upgrade license can be generated.
- If the flag value is VLSucg\_UPGRADE\_CAPACITY then only capacity upgrade license can be generated.
- If the flag value is VLSucg\_UPGRADE\_ALL then both version and capacity upgrade license can be generated.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

#### VLSucgSetUpgradeFlag Error Codes

| Error Code                 | Description                                                                                                  |
|----------------------------|--------------------------------------------------------------------------------------------------------------|
| VLSucg_BAD_HANDLE          | If the handle passed is not a valid handle.                                                                  |
| VLSucg_INVALID_INPUT       | If the either the ucodeP or upd_flags is passed as NULL. Also if the upd_flags is passed as an empty string. |
| VLSucg_INVALID_INT_TYPE    | If value of upd_flags is not numeric.                                                                        |
| VLSucg_EXCEEDS_MAX_VALUE   | If value of upd_flags exceeds VLSucg_UPGRADE_ALL                                                             |
| VLSucg_LESS_THAN_MIN_VALUE | If value is lower than VLSucg_UPGRADE_VERSION.                                                               |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

### VLSucgAllowUpgradeVersion

#### Syntax

```
int VLSucgAllowUpgradeVersion(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP);
```

| Argument       | Description                          |
|----------------|--------------------------------------|
| <i>iHandle</i> | Instance handle for this library.    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct. |

**Description** Indicates whether the corresponding VLSucgSetUpgradeVersion should be called or not.

Only if the VLSucgAllowUpgradeVersion returns 1 then the corresponding VLSucgSetUpgradeVersion should be called.

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLSucgAllowUpgradeVersion Error Codes**

| Error Code | Description                            |
|------------|----------------------------------------|
| 1          | VLSucgSetUpgradeVersion is allowed     |
| 0          | VLSucgSetUpgradeVersion is not allowed |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLSucgSetUpgradeVersion**

**Syntax**

```
int VLSucgSetUpgradeVersion(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP,
char *upgrade_version);
```

| Argument               | Description                                                                                                     |
|------------------------|-----------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i>         | Instance handle for this library.                                                                               |
| <i>ucodeP</i>          | The pointer to <i>ucodeT</i> struct.                                                                            |
| <i>upgrade_version</i> | <ul style="list-style-type: none"><li>Any printable ASCII except #.</li><li>Maximum of 11 characters.</li></ul> |

**Description** Sets the value of *upd\_version* in the *ucodeT* struct to the value of *upgrade\_version*. This function also checks the input variables for their validity and boundary conditions.

**Returns** The status code VLSucg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

#### VLSucgSetUpgradeVersion Error Codes

| Error Code               | Description                                                                                                                                                        |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VLSucg_INVALID_INPUT     | If the either the <code>ucodeP</code> or <code>upgrade_version</code> is passed as NULL. Also if the <code>upgrade_version</code> does not contain a valid string. |
| VLSucg_INVALID_CHARS     | If <code>upgrade_version</code> characters are not printable.                                                                                                      |
| VLSucg_RESERV_STR_ERROR  | If the <code>upgrade_version</code> is a reserved string.                                                                                                          |
| VLSucg_EXCEEDS_MAX_VALUE | If the length of <code>upgrade_version</code> string exceeds maximum allowed length.                                                                               |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgAllowUpgradeCapacity

**Syntax**

```
int VLSucgAllowUpgradeCapacity(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP);
```

| Argument       | Description                          |
|----------------|--------------------------------------|
| <i>iHandle</i> | Instance handle for this library.    |
| <i>ucodeP</i>  | The pointer to <i>ucodeT</i> struct. |

**Description** Indicates whether the corresponding VLSucgSetUpgradeCapacityUnits and VLSucgSetUpgradeCapacity should be called or not. If the VLSucgAllowUpgradeCapacity returns 1 only then the corresponding Capacity should be called.

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

**VLScgAllowUpgradeCapacity Error Codes**

| Error Code | Description                            |
|------------|----------------------------------------|
| 1          | VLScgSetUpgradeCapacity is allowed     |
| 0          | VLScgSetUpgradeCapacity is not allowed |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLScgSetUpgradeCapacityUnits**

**Syntax**

```
int VLScgSetUpgradeCapacityUnits(
 VLSucg_HANDLE iHandle,
 ucodeT *ucodeP,
 char *cap_units);
```

| Argument         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iHandle</i>   | Instance handle for this library.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>ucodeP</i>    | The pointer to <i>ucodeT</i> struct.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>cap_units</i> | Capacity specification units: from 0 to 4. The values are: <ul style="list-style-type: none"><li>• If <i>capacity_units</i> is 0, capacity shall be multiple of 1(s), maximum 1022.</li><li>• If <i>capacity_units</i> is 1, capacity shall be multiple of 10(s), maximum 10220.</li><li>• If <i>capacity_units</i> is 2, capacity shall be multiple of 100(s), maximum 102200.</li><li>• If <i>capacity_units</i> is 3, capacity shall be multiple of 1000(s), maximum 1022000.</li><li>• If <i>capacity_units</i> is 4, capacity shall be multiple of 10000(s), maximum 10220000.</li></ul> |

**Description** Sets the value of *capacity\_units* in the *ucodeT* struct. This function should be called either in case of capacity upgrade or in case of both version and capacity upgrade.

VLSucgSetUpgradeCapacityUnits also check the input variables for their validity and boundary conditions.

**Returns**

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

**VLSucgSetUpgradeCapacityUnits Error Codes**

| Error Code                 | Description                                                                                                  |
|----------------------------|--------------------------------------------------------------------------------------------------------------|
| VLSucg_INVALID_INPUT       | If the either the ucodeP or cap_units is passed as NULL. Also if the cap_units is passed as an empty string. |
| VLSucg_INVALID_INT_TYPE    | If value of cap_units is not numeric.                                                                        |
| VLSucg_EXCEEDS_MAX_VALUE   | If the value of cap_units exceeds VLScg_CAPACITY_UNITS_MAX_VALUE                                             |
| VLSucg_LESS_THAN_MIN_VALUE | If the value is lower than VLScg_CAPACITY_UNITS_MIN_VALUE.                                                   |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## VLSucgSetUpgradeCapacity

**Syntax**

```
int VLSucgSetUpgradeCapacity(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP,
char *cap_increment);
```

| Argument      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iHandle       | Instance handle for this library.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| ucodeP        | The pointer to ucodeT struct.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| cap_increment | Controls the capacity <ul style="list-style-type: none"><li>• If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022.</li><li>• If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220.</li><li>• If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200.</li><li>• If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000.</li><li>• If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000.</li></ul> <b>NOLIMITSTR or EMPTY("/0") String can be used to specify infinite capacity.</b> |

**Definition**

Sets the value of *capacity\_increment* in the *ucodeT* struct. This function also check the input variables for their validity and boundary conditions. Infinite capacity shall also be allowed.

**Returns**

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

### VLSucgSetUpgradeCapacity Error Codes

| Error Code           | Description                                                                                                          |
|----------------------|----------------------------------------------------------------------------------------------------------------------|
| VLSucg_BAD_HANDLE    | If the handle passed is not a valid handle.                                                                          |
| VLSucg_INVALID_INPUT | If the either the ucodeP or cap_increment is passed as NULL. Also if the cap_increment is passed as an empty string. |
| VLSucg_NOT_MULTIPLE  | If value is not a correct multiple.                                                                                  |



**VLSucgSetUpgradeCapacity Error Codes (Continued)**

| Error Code                 | Description                                            |
|----------------------------|--------------------------------------------------------|
| VLSucg_INVALID_INT_TYPE    | If value of cap_increment is not numeric.              |
| VLSucg_EXCEEDS_MAX_VALUE   | If the value of cap_increment exceeds maximum allowed. |
| VLSucg_LESS_THAN_MIN_VALUE | If the value is lower than minimum allowed.            |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLSucgGenerateLicense****Syntax**

```
int VLSucgGenerateLicense(
VLSucg_HANDLE iHandle,
ucodeT *ucodeP,
char *upgrade_code,
char **result);
```

| Argument            | Description                                              |
|---------------------|----------------------------------------------------------|
| <i>iHandle</i>      | Instance handle for this library.                        |
| <i>ucodeP</i>       | The pointer to <i>ucodeT</i> struct.                     |
| <i>upgrade_code</i> | Upgrade code of base license                             |
| <i>result</i>       | Address of pointer pointing to generated license string. |

**Description**

Generates the upgrade license string for the given *ucodeT* struct. VLSucgGenerateLicense should be called after all the VLSucgSet functions are called. Memory allocation and free for *ucodeT* are the responsibilities of the caller of the API. Memory allocation for the license string shall be taken care by the API.

VLSucgGenerateLicense decodes the *upgrade\_code* and extract the information of base license. It performs the following validation before generating an upgrade license:

- Feature Name, Version and vendor code of base license is matched with the base feature name, base version and vendor code of *ucodeT*.
- The capacity upgrade is allowed only if the base license is a Non-pooled capacity license.

**Returns**

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

**VLSucgGenerateLicense Error Codes**

| Error Code                                    | Description                                                                               |
|-----------------------------------------------|-------------------------------------------------------------------------------------------|
| <code>VLSucg_INVALID_INPUT</code>             | If the <code>ucodeP</code> is passed as <code>NULL</code> .                               |
| <code>VLSucg_INVALID_VENDOR_CODE</code>       | If vendor identification is illegal.                                                      |
| <code>VLSucg_VENDOR_ENCRYPTION_FAIL</code>    | If vendor-customized encryption fails.                                                    |
| <code>VLSucg_MALLOC_FAILURE</code>            | If error occur while allocating internal memory for <code>ucodeT</code> struct.           |
| <code>VLSucg_LICMETER_EXCEPTION</code>        | If error occur while accessing the dongle.                                                |
| <code>VLSucg_NO_NETWORK_AUTHORIZATION</code>  | If not authorized to generate network licenses.                                           |
| <code>VLSucg_LICMETER_COUNTER_TOOLOW</code>   | If license meter count is less than the expected decrement count.                         |
| <code>VLSucg_NO_CAPACITY_AUTHORIZATION</code> | If not authorized to generate capacity licenses.                                          |
| <code>VLSucg_NO_UPGRADE_AUTHORIZATION</code>  | If not authorized to generate upgrade licenses.                                           |
| <code>VLSucg_INTERNAL_ERROR</code>            | If any internal error occur while generating the license string.                          |
| <code>VLSucg_INVALID_BASE_LIC_INFO</code>     | The information-feature name, version vendor code provided for base license is incorrect. |
| <code>VLSucg_CAPACITY_UPD_NOT_ALLOWED</code>  | Capacity upgrade is not allowed, as the base lic is a non-capacity license.               |

**VLSucgGenerateLicense Error Codes (Continued)**

| Error Code                    | Description                                      |
|-------------------------------|--------------------------------------------------|
| VLSucg_INVALID_UPGRADE_CODE   | The specified upgrade code is invalid.           |
| VLSucg_LICMETER_NOT_SUPPORTED | Your Sentinel LM License Meter is not supported. |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLSucgGetLicenseMeterUnits****Syntax**

```
int VLSucgGetLicenseMeterUnits(
VLSucg_HANDLE iHandle,
long *initialUnitsP,
long *unitsLeftP,
int ucodegen_version);
```

| Argument                | Description                                              |
|-------------------------|----------------------------------------------------------|
| <i>iHandle</i>          | Instance handle for this library.                        |
| <i>initialUnitsP</i>    | User provided license string to be decoded.              |
| <i>unitsLeftP</i>       | User allocated buffer to receive decoded license string. |
| <i>ucodegen_version</i> | Version of the ucodegen library                          |

**Description** Returns the number of license generation units available in the attached dongle.

**Returns** The status code VLSucg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLSucgGetLicenseMeterUnits Error Codes**

| Error Code                    | Description                           |
|-------------------------------|---------------------------------------|
| VLSucg_INVALID_VENDOR_CODE    | If vendor identification is illegal.  |
| VLSucg_VENDOR_ENCRYPTION_FAIL | If vendor-customized encryption fails |

**VLSucgGetLicenseMeterUnits Error Codes (Continued)**

| Error Code                    | Description                                                              |
|-------------------------------|--------------------------------------------------------------------------|
| VLSucg_MALLOC_FAILURE         | If error occur while allocating internal memory for <i>ucodeT</i> struct |
| VLSucg_FAIL                   | On Failure.                                                              |
| VLSucg_LICMETER_NOT_SUPPORTED | Your Sentinel LM License Meter is not supported.                         |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

**VLSSgenerateUpgradeLockCode**

**Syntax**

```
int VLSgenerateUpgradeLockCode(
 unsigned char *lic_string,
 unsigned char *upgrade_lock_code,
 unsigned long *buffer_size);
```

| Argument                 | Description                                                                                                                                                                                                    |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lic_string</i>        | Base license string.                                                                                                                                                                                           |
| <i>upgrade_lock_code</i> | Buffer containing the generated upgrade lock code. The caller should allocate the memory space.                                                                                                                |
| <i>buffer_size</i>       | <ul style="list-style-type: none"><li>• Size of the allocated buffer.</li><li>• If NULL is passed instead of buffer, then this will return buffer size required for the generated upgrade lock code.</li></ul> |

**Description**

VLSSgenerateUpgradeLockCode allows the user to generate a unique upgrade code for the base license. The upgrade code must be an encrypted string so that it doesn't make any visible sense to the user/developer.

This API is a part of the generic library (*lsutil32.lib*).

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

#### VLSgenerateUpgradeLockCode Error Codes

| Error Code              | Description                                                                                                                                              |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| LS_NORESOURCES          | Unable to allocate memory required to decode the passed license string and to generate upgrade code.                                                     |
| VLS_CALLING_ERROR       | If called with invalid arguments.                                                                                                                        |
| LS_NO_SUCCESS           | If unable to generate upgrade code.                                                                                                                      |
| VLS_VENDORIDMISMATCH    | If license string with invalid vendor code is passed.                                                                                                    |
| VLS_UPGRADE_NOT_ALLOWED | It shall not generate the Upgrade Code if the base license is found to be a Multi Feature Short Numeric, or Trial or Commuter or Redundant License.      |
| LS_BUFFER_TOO_SMALL     | <ul style="list-style-type: none"><li><i>buffer</i> parameter is NULL.</li><li>Size of upgrade lock code exceeds <i>buffer_size</i> parameter.</li></ul> |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.

## Upgrade License Decode API

The following table summarizes the upgrade license decode related functions:

#### Upgrade License Decode Related Functions

| Function             | Description                                              |
|----------------------|----------------------------------------------------------|
| <code>ulcCode</code> | Stores information required to decode upgrade lock code. |

Upgrade License Decode Related Functions (Continued)

| Function                    | Description                                                                         |
|-----------------------------|-------------------------------------------------------------------------------------|
| VLDecodeUpgrade<br>lockCode | Decodes the upgrade lock code.                                                      |
| VLScgDecodeLicense          | Decodes the encrypted license string generated by<br>upgrade code generator library |

ulcCode Struct

```
typedef struct
{
 int version_num;
 char hash_vendor_string[VENDOR_HASH_LENGTH];
 int capacity_flag;
 int standalone_flag;
 unsigned num_keys;
 int birth_day;
 int birth_month;
 int birth_year;
 int death_day;
 int death_month;
 int death_year;
 int client_server_lock_mode;
 unsigned char base_lock_code[BASE_LOCK_CODE_LENGTH +
 1];
 char base_feature_name[VLScg_MAX_CODE_COMP_LEN +
 1];
 char base_feature_version[VLScg_MAX_CODE_COMP_LEN
 + 1];
 unsigned long capacity;
} ulcCode;
```

ulcCode Struct

| Member                    | Description                                                             |
|---------------------------|-------------------------------------------------------------------------|
| <i>version_num</i>        | Number maintaining the version of the structure                         |
| <i>hash_vendor_string</i> | A numeric value representing the feature and version<br>of the license. |

**ulcCode Struct**

| Member                         | Description                          |
|--------------------------------|--------------------------------------|
| <i>capacity_flag</i>           | The value of capacity flag.          |
| <i>standalone_flag</i>         | The value of standalone flag.        |
| <i>num_keys</i>                | The number of keys                   |
| <i>birth_day</i>               | The starting day of the license.     |
| <i>birth_month</i>             | The starting month of the license.   |
| <i>birth_year</i>              | The starting year of the license.    |
| <i>death_day</i>               | The expiration day of the license.   |
| <i>death_month</i>             | The expiration month of the license. |
| <i>death_year</i>              | The expiration year of the license.  |
| <i>client_server_lock_mode</i> | The locking mode                     |
| <i>base_lock_code</i>          | Base lock code                       |
| <i>base_feature_name</i>       | Base feature name                    |
| <i>base_feature_version</i>    | Base feature version                 |
| <i>capacity</i>                | Capacity of the license.             |

**VLSdecodeUpgradelockCode****Syntax**

```
int VLSdecodeUpgradelockCode(
 char *upgrade_lock_code ,
 char *compactd_upd_lock_code ,
 int length ,
 ulcCode **ulcCodePP);
```

| Argument                 | Description                      |
|--------------------------|----------------------------------|
| <i>upgrade_lock_code</i> | Upgrade lock code to be decoded. |

| Argument                       | Description                                                                                           |
|--------------------------------|-------------------------------------------------------------------------------------------------------|
| <i>compacted_upd_lock_code</i> | Upgrade lock code string after removing comment chars and white spaces. This can also be set as null. |
| <i>length</i>                  | Length of <i>compacted_upd_lock_code</i> in case it is not null.                                      |
| <i>ulcCodePP</i>               | Pointer to pointer to <i>ulcCode</i> structure.                                                       |

**Description** VLSdecodeUpgradelockCode API decodes the upgarde lock code.

**Returns** The status code LS\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

**VLSdecodeUpgradelockCode Error Codes**

| Error Code         | Description                                                              |
|--------------------|--------------------------------------------------------------------------|
| LS_NORESOURCES     | If vendor identification is illegal.                                     |
| LS_NO_SUCCESS      | Failed to decrypt the license                                            |
| VLS_INTERNAL_ERROR | If error occur while allocating internal memory for <i>ucodeT</i> struct |

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**VLSucgDecodeLicense**

**Syntax**

```
int VLSucgDecodeLicense(
 VLSucg_HANDLE iHandle,
 char *AnyLicenseString,
 char *lic_string,
 int lic_string_length,
 ucodeT **ucodePP);
```

| Argument                | Description                                              |
|-------------------------|----------------------------------------------------------|
| <i>iHandle</i>          | 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. |



| Argument                 | Description                                                                                                                          |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>Lic_string_length</i> | Length of decoded license string returned.                                                                                           |
| <i>ucodePP</i>           | <ul style="list-style-type: none"> <li>Address of pointer to <i>ucodeT</i> struct</li> <li>Contains input license string.</li> </ul> |

**Description** VLSucgDecodeLicense API is contained in *lsdcod32.lib*. This library needs to be called for using VLSucgDecodeLicense API without the license meter.

Decodes the encrypted license string generated by upgrade code generator library. It also converts the license string into *ucodePP* struct.

---

**Note:** VLSucgDecodeLicense does not decodes/understand the normal (base) licenses.

---

**Returns** The status code VLScg\_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

#### VLSucgDecodeLicense Error Codes

| Error Code                    | Description                                                              |
|-------------------------------|--------------------------------------------------------------------------|
| VLSucg_INVALID_VENDOR_CODE    | If vendor identification is illegal.                                     |
| VLSucg_VENDOR_ENCRYPTION_FAIL | If vendor-customized encryption fails                                    |
| VLSucg_MALLOC_FAILURE         | If error occur while allocating internal memory for <i>ucodeT</i> struct |
| VLSucg_FAIL                   | On Failure.                                                              |

For a complete list of the error codes, see Appendix E, “Error Codes for Upgrade License Functions,” on page 461.



# Chapter 10

---

## Usage Log Functions

The usage log functions control and manipulate the usage log file.

The following table summarizes the usage log functions:

**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.
- 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, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

**VLSgetUsageLogFileName**

- Syntax**
- ```
int VLSgetUsageLogFileName(  
    char *hostName ,  
    char *fileName );
```

Argument	Description
<i>hostName</i>	The host name of the computer containing the license server that is using the log file.
<i>fileName</i>	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, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

Chapter 11

Utility Functions

The utility functions are only available on UNIX platforms:

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,  
    void             *eventHandler,  
    long             repeat_event);
```

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.

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, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSDisableEvents

Syntax

```
int VLSDisableEvents(  
    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, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.

VLSeventSleep

Syntax

```
void VLSeventSleep (unsigned int seconds));
```

Argument	Description
<i>seconds</i>	Time in <i>seconds</i> to sleep.

Description	<p>This function is called for disabling the license operations for an allotted time and interferes with the system alarms.</p> <p>VLSeventSleep must be used in conjunction with VLSdisableAutoTimer.</p>
Returns	<p>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, Appendix C, “Sentinel LM Error and Result Codes,” on page 431.</p>

Chapter 12

System Initialization Functions

This chapter discusses the following system initialization API functions that you must use for strengthening the security of your protected applications:

- `sntlInitStandaloneSystem` - Sets up the system for the stand-alone licensing requirements.
- `sntlInitNetworkSystem` - Sets up the system for the network licensing requirements.

Note: You need to decide whether you would like your protected application to run on stand-alone computers, networked computers, or both. It is important for determining which API function to integrate in your application installers to initialize the system at the time of installation.

`sntlInitStandaloneSystem`

Description

The function is used for initializing the client system for successful use of a local request license. If the protected application does not find the initialization information set up by this function at the time of locally requesting a license, it would generate an error.

Use of this function in your application installer will ensure that your protected application is not installed more than once on a system.

This API is meant to be executed before running any local request application using trial or any time tampering-enabled license (such as commuter, perpetual and grace license).

It is a must to initialize the system in admin mode before running any local request application or checking out a token for normal user.

Syntax

```
int sntlInitStandaloneSystem(  
    char          *GUID,  
    char          *featureName ,  
    char          *featureVersion);
```

Argument	Description
<i>GUID</i>	The application installer’s GUID ^a .
<i>featureName</i>	The application feature name.
<i>featureVersion</i>	The application feature version.

a. Stands for Global Unique Identifier—a unique 128-bit number that is produced by the Windows operating system or by some Windows applications to identify a particular component, application, file, database entry, and/or user.

Note: For UNIX users, the API parameters are not significant and can be passed as NULL. In case any values are specified they shall be ignored on UNIX systems. The function must be called only once to initialize the system.

On Windows, this function needs to be called for every standalone application with unique values of input parameters, else function will return an error.

Returns

The status code `ERR_KEY_INFO_SUCCESS` is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure.

Refer to section "Status Codes" for more details on return codes.

Requirements

Header:

Include *lsinit.h*, *cnvtstr.h*

Library:

Windows - *lsinit32.lib*, *lsinit32.dll* (both MT and MD versions)

UNIX - *liblsinit.a*, *liblsinit.so*

Example

A sample file in C (*initdemo.c*) is provided to illustrate the use of the function.

- For Windows, it is installed at <installdir>\SafeNet Sentinel\Sentinel LM\<Sentinel LM version>\English\MsvcDev\Samples.
- For UNIX, it is installed at <installdir>\examples.
- "On Windows, a command utility (*lsgeninit.exe*) is also provided that helps in setting up the system for stand-alone licensing. Refer to section on "Lsgeninit utility" for more details.

Note: This function is available only for the applications protected using Sentinel LM version 8.x or higher. If you plan to ship Sentinel LM version 8.x libraries to your existing users in the field, you must upgrade your application installer to call this function.

sntlInitNetworkSystem

Description This function sets the requisite permissions for the Sentinel LM registry keys (if applicable) and files at the time of system initialization.

On Windows, if you decide to integrate the license server in your own installer, there is no need to call this function because the network system initialization is done using the Sentinel LM Server Installer (version 8.0 or higher).

On UNIX, network initialization is done automatically by SLM server, and there is no need for explicit initialization.

For Windows

Syntax `int sntlInitNetworkSystem (void)`

Returns The status code `ERR_KEY_INFO_SUCCESS` is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure.

Refer to section "Status Codes" for more details on return codes.

Requirements

Header

Include *lsinit.h*, *cnvtstr.h*

Library

Windows - *lsinit32.lib*, *lsinit32.dll* (both MT and MD versions)

UNIX - *lsinit32.a*, *lsinit32.so*

Example A sample file in C (*initdemo.c*) is provided to illustrate the use of the function.

- For Windows, it is installed at <installdir>\SafeNet Sentinel\Sentinel LM\<Sentinel LM version>\English\MsvcDev\Samples.
- For UNIX, it is installed at <installdir>\examples.

Status Codes

The following error codes may be returned by any of the two APIs:

Status Code	Description
ERR_KEY_INFO_SUCCESS	Success
ERR_INVALID_INPUT	Invalid Input
ERR_INIT_LIB_FAIL	Library Initialization Failure
ERR_INIT_INFO_EXISTS	Already Initialized
ERR_TIME_TAMPER_INIT_FAIL	Time Tamper Initialization Failure
ERR_COMMUTER_PRS_INIT_FAIL	Commuter Initialization Failure
ERR_GRACE_PRS_INIT_FAIL	Grace License Initialization Failure
ERR_TRIAL_INIT_FAIL	Trial License Initialization Failure
ERR_INIT_SEC_FAIL	Permissions failure on secure information
ERR_KEY_INFO_EXIST	Registry information already exists (only for Windows)
ERR_INIT_KEY_FAIL	Failed to create registry information (only for Windows)
ERR_INIT_KEY_SEC_FAIL	Permission failure for secure registry (only for Windows)
ERR_INIT_OPEN_KEY_FAIL	Registry open failure (only for Windows)
ERR_INIT_SET_VALUE_FAIL	Write failure on secure registry (only for Windows)
ERR_INIT_QUERY_FAIL	Read failure on secure registry (only for Windows)
ERR_INIT_FILE_FAIL	File creation failure
ERR_INIT_FILE_OPEN_FAIL	File open failure
ERR_INIT_FILE_WRITE_FAIL	Read or Write failure on secure file
ERR_INIT_FILE_SEC_FAIL	Permissions failure on secure file

Status Code	Description
ERR_INIT_FILE_INFO_EXISTS	Initialization file information exists
ERR_INTERNAL_FAIL	Internal error
ERR_INSUFFICIENT_PERMISSIONS	Access denied for read or write secure data
ERR_TAMPER_DETECTED	Secure data is tampered

Lsgeninit Utility

On Windows, a command utility (lsgeninit.exe) is also provided that helps in setting up the system for stand-alone licensing. This utility is available at: **<INSTALLDIR>\SafeNet Sentinel\Sentinel LM\<Sentinel LM version>\English\Tools**

This utility requires a configuration file, containing the details about the GUID, feature-name and version, as command-line parameter. The configuration file can list up to 100 records, one record to a line. Each line in the format: '<GUID> <FeatureName> [<Version>] # comments'

GUID and FeatureName are required, the Version field is optional. Each feature and version should be listed only once.

Tip: Specify 'RandomGUID' keyword in the configuration file to have a GUID automatically generated for any feature.

This utility also requires a stub binary "lsgeninitstub.exe" to be present at the same location from where lsgeninit utility is executed.

On successful execution of lsgeninit utility, it generates the installer file "lsinit.exe". This installer file should be executed on the target system. If executed with "-debug" option, it returns the initialization status for each record."

Appendix A

Sample Applications

Sentinel LM installs a few sample applications on your computer. These sample applications can be located at `\SafeNet Sentinel\Sentinel LM\8.X.X\English\MsvcDev\Sample`.

On the UNIX platforms the following components/files are available:

Customization Sample Files on UNIX

Component	File(s)
linking	<i>Makefile</i>
lslic	<i>lslic.c</i>
lsmon	<i>lsmon.c</i>
lswhere	<i>lswhere.c</i>
Challenge-response	<i>crexamp.c, chalresp.[c h], md4.[c h]</i>

On the Windows platforms the following components/files are available:

Customization Sample Files on Windows

Component	File(s)
the license server	<i>lservdown.[c dsp], lserv.h</i>
licence generator	<i>echoid32.dsp, echomain.c</i>
lslic	<i>lslic.[c dsp]</i>

Customization Sample Files on Windows

Component	File(s)
Ismon	<i>Ismon.[c dsp]</i>
Iswhere	<i>Iswhere.[c dsp dsw ncb opt]</i>
Challenge-response	<i>crexamp.c, chalresp.[c h], md4.[c h]</i>
Sample function macros	<i>dots, bounce</i>

Appendix B

Customization Features

The Sentinel LM package is optionally shipped with a number of precompiled object modules to enable you to re-link the license manager and the code generator executable, and override certain predefined Sentinel LM characteristics.

There are compatibility issues for object files generated by different versions of compilers on Microsoft Windows platforms. Therefore, *server.o* and *lsc-gen.o* files are not included in the Windows distribution. Please contact Technical Support (see page xxiii) for information about customization tools availability for your version of Windows developer platforms.

The following table summarizes the customizing functions:

Customizing Functions

Functions	Description
VLSserverVendorInitialize	Initializes the server.
VLSeventAddHook	Registers an event handler with the server.
VLSconfigureTimeTamper	Defines the criteria on which time tampering is detected.
VLSisClockSetBack	Notifies the license server to check whether the clock has been set back.
VLSencryptLicense	Encrypts license codes.

Customizing Functions (Continued)

Functions	Description
VLSdecryptLicense	Decrypts license codes.
VLSencryptMsg	Encrypts messages.
VLSdecryptMsg	Decrypts messages.
VLSchangePortNumber	Changes the port number.

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\SafeNet Sentinel\Sentinel LM\MsvcDev\custom*. In this directory you will find the Makefile *custom32.mak*. Make a copy of this file and name it **MAKE-FILE**. Edit this file. Add your customized object files in the following section:

```
# For now, use the default functions from the Sentinel
LM 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
	✓	✓	

Description 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
eventName	Specifies the type of event. <ul style="list-style-type: none">■ 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_PODY right after the license release is processed by the server.
(*handlerFuncPtr) (VLShandlerStruct *, char*, char *, int)	The function pointer.
identifier	The client identifier 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*, *relpshk1.c*, *reqprhk2.c*, *reqpshk2.c*, and *relprhk2.c*. The client portion of the project can be found in *hookdemo.c*.

```

/*****
/*
/* Copyright (C) 2004 SafeNet, 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 VLSserverVendorInitialize(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 Sentinel LM license server can be configured to detect tampering of the system clock.

In case of UNIX systems Sentinel LM 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. Sentinel LM calls the function `VLSconfigureTimeTamper` before performing any time tamper checks. However, even on UNIX systems, Sentinel LM may not be able to detect time tampering if the system is running for a long time (in the time tampered mode) before the Sentinel LM server has been started. Also the Sentinel LM server may stop detecting time tampering if all the files (that it is checking) start having the same time stamp.

`VLSconfigureTimeTamper` function can be used to modify the default behavior of Sentinel LM regarding time tamper checking. You need to perform the following steps:

1. Write your own `VLSconfigureTimeTamper` 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 `VLSisClockSetBack` which returns 0 if the system clock has not been set back, and 1 otherwise.
3. In the *Makefile* in the examples directory, 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 Sentinel LM built-in system clock tamper checking function, or use one provided by you. [VLS_ENABLE_DEFAULT_TM_TAMPER]
<i>gracePeriod</i>	Useful only in case <i>tmTamperMethod</i> is VLS_ENABLE_DEFAULT_TM_TAMPER. If Sentinel LM 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 to 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: If both *percentViolations* and *numViolationsForError* are used, the lower evaluated value will be used.

VLSisClockSetBack

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 VLSconfigureTimeTamper 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 Sentinel LM utility, **lsde-code**. 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 (OUT)</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) 2004 SafeNet, Inc.
/*       All Rights Reserved
/*
/*
/*****
/* Usage of VLSEncryptLicense() */
#include <stdio.h>
#include <string.h>
#include "lctest.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 (OUT)</i>	The decrypted license code to be returned.
<i>buffSize</i>	Size of the decrypted text buffer.

Description See VLSencryptLicense above.

Example file:

```
/*
 *
 * Copyright (C) 2004 SafeNet, Inc.
 * All Rights Reserved
 *
 */
/*****
 * Usage of VLSdecryptLicense() */
#include <stdio.h>
#include <string.h>
#include "lstest.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 Sentinel LM 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 (OUT)</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 VLSdecryptMsg 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 VLSdecryptMsg.

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 Sentinel LM client library.

Returns 0 if successful; other value on failure.

Example file:

```
/*
 * Copyright (C) 2004 SafeNet, 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 (OUT)</i>	The decrypted message text.
<i>buffSize</i>	Size of the decrypted text buffer.

Description

See VLSencryptMsg on the previous page.

Example file:

```
/* ***** */
/*
/* Copyright (C) 2004 SafeNet, Inc.
/* All Rights Reserved
/*
/* ***** */
/* Usage of VLSDecryptMsg() */
#include <stdio.h>
#include <string.h>
#include "lctest.h"
int VLSDecryptMsg(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.

VLSchangePortNumber

Client	Server	Static Library	DLL
	✓	✓	

Changes the port number.

Syntax

```

int  VLSchangePortNumber(
int  currentPort);

```

Argument	Description
<i>currentPort</i>	Current port number.

Description

Sets port number to *newPort*. This function should be called only once, at license server start-up time.

The licensed application can obtain or reset its port number through the client library function calls, VLSgetServerPort and VLSsetServerPort. These setup 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) 2004 SafeNet, 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);
}getCustomHostId
```

Customizing the Host ID

Sentinel LM 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    machineID;
    unsigned long   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 `VLSServerVendorInitialize` 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%X", 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.

Customizing Upgrade Licenses

As a developer you may want to write your own encryption and decryption algorithm. The Sentinel LM upgrade license generator library allows you to write your own algorithm using the following API:

VLSEncryptUpgradeLicense

Syntax

```
int VLSEncryptUpgradeLicense
(
    char  *original_text,
    char  *encrypted_text_buffer,
    int   buff_size
);
```

Definition

This API can be used to write encryption algorithm for upgrade licenses over the default encryption algorithm provided by the upgrade license generator library.

Argument	Type	Description
<i>original_text</i>	IN	The string to be encrypted
<i>encrypted_text_buffer</i>	OUT	The resultant string after encryption
<i>buff_size</i>	IN	Length of the output buffer.

Returns

Error Code	Description
0	Success
Any other value	Failure

VLSdecryptUpgradeLicense

Syntax

```
int VLSdecryptUpgradeLicense
(
    char *original_text,
    char *decrypted_text_buffer,
    int  buff_size
);
```

Definition

This API can be used to write decryption algorithm for upgrade licenses over the default decryption algorithm provided by the upgrade license generator library.

Argument	Type	Description
original_text	IN	The string to be decrypted
encrypted_text_buffer	OUT	The resultant string after encryption
buff_size	IN	Length of the output buffer.

Returns

Error Code	Description
0	Success
Any other value	Failure

Setting License Server Information

A customizable API, VLSsetServerInfo, is provided to allow the developer to customize his server by setting vendor -specific information in his license server which can be returned to the client using the VLSgetServInfo call. See “Retrieving Information About a License Server (VLSgetServInfo)” on page 130.

Setting Vendor Specific Information in a License Server (VLSsetServerInfo)

By using this call and rebuilding the license server (as you do when customizing the license server for custom locking), you can set a string value of up

to 50 characters to be written to the license server to serve as identification. (This string can be returned by the VLSgetServInfo call in the *vendor_info* field of the VLSServInfo structure to verify that the license server is the correct one.)

Syntax

```
LSERV_STATUS VLSsetServerInfo(  
char          **vendorInfo);
```

Argument	Direction	Description
vendorInfo	OUT	String of up to 50 characters to write to the license server as identification.

Returns Returns zero if successful.

Customizing Stand-alone License File Names (VLSsetFileName)

Sentinel LM reads a number of files to determine what licenses are available and how the license server should operate. For stand-alone applications, these files are:

- *lservrc* - license file, which contains one or more license strings.
- *lservrcnf* - license server configuration file, which contains license server options.

Although the names of these files can be changed by the developer by using the appropriate environment variable, this can cause a conflict if multiple stand-alone applications from different developers are installed on the same computer since only one environment variable affects the entire computer. If environment variables aren't used, the default license file can be overwritten by one from a different developer when a new application is installed.

The VLSsetFileName call is now available to set these file names from within the application.

Note: VLSsetFileName should be used before calling VLSinitialize.

This call can only be used with stand-alone or integrated client libraries.

Syntax

```
LS_STATUS_CODE VLssetFileName(  
    VLS_FILE_TYPE fileType,  
    unsigned char *fileName,  
    unsigned char *unused1,  
    unsigned long *unused2);
```

Argument	Direction	Description
filetype	IN	Selects the type of license file you are going to provide a customized name for. Set to one of the following: ■ VLS_LSERVRC ■ VLS_LSERVRCCNF
fileName	IN	Custom name you want to use.
unused1		Reserved. Use NULL for this value.
unused2		Reserved. Use NULL for this value.

Returns

The status code LS_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for the failure. Possible error codes returned by this call include: VLS_INVALID_FILETYPE and VLS_NOT_APPROPRIATE_LIBRARY.

For a complete list of error codes, see “Sentinel LM Error and Result Codes” on page 431.

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 the Microsoft Visual C++ compiler. Please contact SafeNet if you are using another compiler and require assistance.

Files Required

The following files are required for rebuilding the license server:

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>lsmainwa.c</i>	C source file containing entry point to license server application.
<i>lserv.h</i>	Sentinel LM include file installed during Sentinel LM installation.
<i>lserv95.res</i>	Resource file for license server application.
<i>lserv9x.lib</i>	Sentinel LM static library installed during Sentinel LM installation.
<i>lserv9x.dsp</i>	Microsoft Visual C++ 6.0 project file.
<i>lserv9x.dsw</i>	Microsoft Visual C++ 6.0 workspace file.

Note: *lserv95.res*, *lserv9x.lib*, *lserv9x.dsp*, and *lserv9x.dsw* files can be slightly different when working with Windows NT or Windows 2000.

Required Changes to Server Source Code

A Sentinel LM license server with custom locking code will differ from a default license server because `VLServerVendorInitialize` is redefined so that it will call `VLSetHostIdFunc`. `VLServerVendorInitialize` is called during server startup for both default license servers and custom license servers, but the default version does not call `VLSetHostIdFunc`.

`VLSetHostIdFunc` 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.

`VLServerVendorInitialize` is automatically called during server startup. However, for servers that initialize custom locking code, `VLServerVendorInitialize` is redefined to call `VLSetHostIdFunc(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 SafeNet, Inc., that contain all the necessary files. Please see “Files Required” on page 424. 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 *echoid.exe*” on page 427.
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 the Microsoft Visual C++ compiler. Please contact SafeNet if you are using another compiler and require assistance.

Files Required for *echoid.exe*

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

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>lscgen.h</i>	Sentinel LM include file installed during Sentinel LM installation.
<i>lserv.h</i>	Sentinel LM include file installed during Sentinel LM installation.
<i>lsapiw32.lib</i>	Import library for Win32 run-time <i>DLL</i> that is installed during Sentinel LM 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 `VLSgetMachineID`, call `VLSsetHostIdFunc(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 SafeNet, Inc., that contain all the necessary files. Please see “Files Required for echoid.exe” on page 426. 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 echoid.exe” on page 427.
4. Choose **Rebuild All** from the Build menu.

Step 3 - Modifying Client Application

The client application should also make a call to `VLSsetHostIdFunc`. 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 echoid.exe” on page 427.

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

1. Decide on an algorithm for generating a 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 424.
3. Rebuild *echoid.exe*. See “Step 2 - Rebuilding *echoid.exe*” on page 426.
4. Edit *echoid.dat* so that the custom locking criteria is a criteria mask. This step may not be needed if the 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 *lservc*.
8. Modify the client application and rebuild it. See “Step 3 - Modifying Client Application” on page 427.
9. Execute the client application.

Adding Additional Security to Licenses Generated by **WlscGen**

In previous Sentinel LM releases, the developer could add an additional layer of security to licenses created by **lscgen** by customizing **lscgen** using the *custom32.mak* make file. This feature was not available for licenses generated by **WlscGen**, the Windows-interface license code generator, because there was no way to customize **WlscGen**. To provide this same functionality

to **WlscGen**, we now provide a make rule in the *custom32.mak* make file to build a *wlscgen.dll* file to customize **WlscGen**.

Here is how to customize **WlscGen**:

1. Build a custom DLL named *Wlscgen.dll*, which uses the same object files as those used for customizing the license server and **lscgen**, defined by the variable `ENCRYPT_LIC_OBJ` in the *custom32.mak* makefile provided in the *\custom* directory.
2. The makefile will copy the DLL file to the default location (the Sentinel LM *\Tools* directory) of *WlscGen.exe* so that the licenses generated by **WlscGen** will use the customized license encryption. If this is not the location of *WlscGen.exe* on your computer, move the DLL file to the directory containing *WlscGen.exe*.

Appendix C

Sentinel LM Error and Result Codes

The following table lists LSAPI client function return codes and their default messages:

LSAPI Client Function Return Codes

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
0x0	00	LS_SUCCESS		Successful completion of function call.
0xC8001001	046	LS_BADHANDLE	Bad index	Handle given to function represents an invalid licensing system context.
0xC8001002	047	LS_INSUFFICIENTUNITS	Could not locate enough licensing resources.	Not enough sufficient resources to satisfy LSRequest.
0xC8001003	048	LS_LICENSESYSNOT AVAILABLE	Licensing System not available.	Licensing system itself is unavailable.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
0xC8001004	049	LS_LICENSETERMINATED	License terminated because renewal time expired.	LSupdate failed. License expired due to time-out.
0xC8001005	044	LS_NOAUTHORIZATION AVAILABLE	Could not find the specified client for the feature.	License server does not recognize this feature name.
0xC8001006	051	LS_NOLICENSES AVAILABLE	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.
0xC8001008	053	LS_NO_NETWORK	Unable to talk to the host specified. Verify client/server communication.	Network communication problems encountered.
0xC8001009	034	LS_NO_MSG_TEXT	The specified filename can not be found on license server.	LSGetMessage unable to retrieve message text.
0xC800100A	055	LS_UNKNOWN_STATUS	Unknown error code, cannot provide error message.	Unknown or unrecognized status code was passed to LSGetMessage.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
0xC800100B	056	LS_BAD_INDEX	Bad index	Invalid index specified in LSEnumProviders or any query functions.
0xC800100C	057	LS_NO_MORE_UNITS	No additional units are available.	Additional licenses/units requested are unavailable.
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.
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.
0xC800100F	060	LS_NO_SUCCESS	No success in achieving the target.	No success in achieving the target.
1	001	VLS_NO_LICENSE_GIVEN	Unable to obtain licensing key.	Other internal error not listed above. 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.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
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.
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.
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.
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.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
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.
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.
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.
11	011	VLS_CALLING_ERROR	Error in calling the function. Check the calling parameters.	VLS_CALLING_ERROR is returned when ever an incorrect value has been provided as a parameter for the API. You need to check the calling parameter of the Sentinel LM API.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
12	012	VLS_INTERNAL_ERROR	Internal error in licensing or accessing feature.	VLS_INTERNAL_ERROR is an internal error message and is returned whenever some client internal function fails in performing some operation. Default: Display error message, return error code.
13	013	VLS_SEVERE_INTERNAL_ERROR	Severe internal error in licensing or accessing feature.	VLS_SEVERE_INTERNAL_ERROR is internal to Sentinel LM client library and is returned whenever client library is unable to retrieve either the system time or while constructing some internal message for client-server processing. Default: Display error message, return error code.
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.
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.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
16	016	VLS_UNKNOWN_SHARED_ID	Unknown shared id specified.	The supplied sharing criteria is unknown. Default: Display error message, return error code.
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.
18	018	VLS_NO_SUCH_FEATURE	No license string is available.	The license server does not recognize the given feature, version and capacity. Default: Display error message, return error code.
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.
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.
21	021	VLS_LOCAL_UPDATE	The last update was done locally.	The last update was done locally.
22	022	VLS_REMOTE_UPDATE	The last update was done remotely.	The last update was performed by contacting the Sentinel LM license server.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
23	023	VLS_VENDORIDMIS MATCH	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.
24	024	VLS_MULTIPLE_VENDORID_FOUND	Feature licensed by multiple vendors.	The license system has licenses for the same <i>feature, version</i> , and it is not clear from the requested operation which license the requestor is interested in.
25	025	VLS_BAD_SERVER_MESSAGE	Could not understand message received from the license server. Verify Client and License server versions match.	VLS_BAD_SERVER_MESSAGE is returned when the client or server is unable to decrypt or understand the message send or received. In case of commuter license error 88 (VLS_TERMINAL_SERVER_FOUND) is returned when one tries to check out a license on terminal server machine.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	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.
27	027	VLS_NOT_AUTHORIZED	Unauthorized operation requested.	The specified operation is not permitted - authorization failed.
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.
29		VLS_UNKNOWN_TAG_TYPE	Tag type is not known to server.	The server does not know of this tag type.
30		VLS_INVALID_TAG_TYPE	Tag type is incompatible with requested operation.	The tag type is invalid for the operation requested.
31		VLS_UNKNOWN_TAG	Supplied tag is not known to the license server on host	The server doesn't know this tag.
32		VLS_UPDATE_TAGGED_KEY_ERROR	Invalid attempt to update a tagged key.	Failure to update a tagged key.
33		VLS_TAGS_NOT_SUPPORTED	License server on host does not support tags.	Server does not support tags.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
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.
35	035	VLS_LOG_FILE_NAME_NOT_CHANGED	Cannot change specified log filename on license server.	Log file name was not changed.
36	036	VLS_FINGERPRINT_MISMATCH	Machine's fingerprint mismatched.	The fingerprint identification of requesting computer does not match with the system.
37	037	VLS_TRIAL_LIC_EXHAUSTED	Duration or usage of a trial license is exhausted.	Trial license usage exhausted or trial license has expired.
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.
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.
40	040	VLS_QUEUED_HANDLE	The specified handle is a queued handle.	The LS_HANDLE is a queued handle.
41		VLS_ACTIVE_HANDLE	<i>lshandle</i> is active handle.	<i>lshandle</i> is active handle.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
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.
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.
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.
45	045	VLS_CLIENT_NOT_AUTHORIZED	Client is not authorized for the specified action.	Client not authorized to make the specified request.
46		VLS_BAD_DISTB_CRIT	Distribution criteria given is not correct	Invalid distribution criteria.
47		VLS_LEADER_NOT_PRESENT	Current leader is not known.	Unknown leader.
48		VLS_SERVER_ALREADY_PRESENT	Server already exists in the server pool.	Attempted to add a license server that is already in the pool.
49		VLS_SERVER_NOT_PRESENT	The given server name does not exist in the server pool.	Attempted to delete a license server that is not in the pool.
50		VLS_FILE_OPEN_ERROR	File open error.	File can not be open.
51		VLS_BAD_HOSTNAME	Bad Host Name.	<i>hostName</i> is not valid.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
52		VLS_DIFF_LIB_VER	Could not understand the message received from license server on host. Client-server version mismatch?	Version mismatch between license server API and client API.
53		VLS_NON_REDUNDANT_SRVR	A non-redundant server contacted for redundant server related information.	License server is non-redundant and therefore cannot support this redundancy-related operation.
54		VLS_MSG_TO_LEADER	Message forwarded to the leader server.	The message has been forwarded to the leader; this is not an error.
55		VLS_CONTACT_FAILOVER_SERVER	Update Failure. Contact another fail-over server.	An update failed. The contact server may have died or been modified.
56		VLS_UNRESOLVED_IP_ADDRESS	IP address given cannot be resolved.	<i>IP_address</i> is valid, but could not be resolved.
57		VLS_UNRESOLVED_HOSTNAME	Hostname given is unresolved.	<i>IP_address</i> is valid, but could not be resolved.
58		VLS_INVALID_IP_ADDRESS	Invalid IP address format.	<i>IP_address</i> is not valid.
59		VLS_SERVER_FILE_SYNC	Server is synchronizing the distribution table.	The license server is synchronizing the distribution table—this is not an error.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
60		VLS_POOL_FULL	The server pool already has the maximum number of servers.No more servers can be added.	Pool already has maximum number of license servers. No more license servers can be added.
61		VLS_ONLY_SERVER	The server pool has only one server. It cannot be deleted.	Pool will not exist if this license server is removed.
62		VLS_FEATURE_INACTIVE	The feature is unavailable on the server or server is non-redundant.	Feature is inactive on specified license server.
63		VLS_MAJORITY_RULE_FAILURE	The token for feature cannot be issued because of majority rule failure.	Majority rule failure prevents token from being issued.
64		VLS_CONF_FILE_ERROR	Configuration file modifications failed. Check the given parameters.	Error in configuration file.
65		VLS_NON_REDUNDANT_FEATURE	A non-redundant feature given for redundant feature related operation.	Feature is non-redundant and thus cannot be used in this redundancy-related operation.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
66		VLS_NO_TRIAL_INFO	No Trial usage info.	No Trial usage info.
67		VLS_TRIAL_INFO_FAILED	Trial usage query failed.	Trial usage query failed.
68		VLS_ELM_LIC_NOT_ENABLE	Elan License of feature is Inactive.	Elan license is not enabled.
69		VLS_NOT_LINKED_TO_INTEGRATED_LIBRARY	Application is not linked with integrated library	Requested operation requires linking to the integrated library not the shared library (DLL or SO).
70		VLS_CLIENT_COMMUTER_CODE_DOES_NOT_EXIST	Client commuter license does not exist.	The client commuter authorization does not exist.
71		VLS_CLIENT_ALREADY_EXISTS	Client already exists on server.	The client already exists.
72		VLS_NO_MORE_COMMUTER_CODE	This is not really an error code.	There are no features to return from the VLSgetCommuterInfo call; this is not an error.
73		VLS_GET_COMMUTER_INFO_FAILED	Failed to get client commuter info on server	Failed to get commuter information.
74		VLS_UNABLE_TO_UNINSTALL_CLIENT_COMMUTER_CODE	Unable to uninstall the client commuter license.	This error message is returned when the SLM server fails to remove the installed commuter. VLSuninstallAndReturnCommuterCode failed.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
75		VLS_ISSUE_COMMUTER_CODE_FAILED	Unable to issue a commuter license to client.	This error message is returned when: <ul style="list-style-type: none"> • Either the SLM server fails to issue the commuter code to the client i.e. the server fails to issue the requested commuter code to client. • The client fails to install the normal commuter code on the requested client machine.
76		VLS_UNABLE_TO_ISSUE_COMMUTER_CODE	Unable to issue a commuter license to client.	The license server is not allowed to issue commuter authorization for the requested feature and version. This error code is returned when the commuter request (check-in/check-out) has been made.
77		VLS_NOT_ENOUGH_COMMUTER_KEYS_AVAILABLE	Not enough commuter tokens available on server.	Not enough keys available to check out more commuter authorizations.
78		VLS_INVALID_INFO_FROM_CLIENT	Invalid commuter info from Client.	Invalid lock information provided by the client.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
79		VLS_CLIENT_ALREADY_EXIST	Client already exists on server.	Server has already checked out one commuter authorization for this client.
80		VLS_COMMUTER_CODE_DOES_NOT_EXIST	Client commuter license does not exist.	No commuter authorization exists for this feature and version.
81		VLS_COMMUTER_CODE_ALREADY_EXIST	Client commuter license already exists on.	Client has already had commuter authorization with this feature and version.
82		VLS_SERVER_SYNC_IN_PROGRESS	Server synchronization in progress. Please wait.	License server synchronization in process.
83		VLS_REMOTE_CHECKOUT	This commuter license is checked out remotely, so it can't be checked-in!	License is a remotely checked out license.
84		VLS_UNABLE_TO_INSTALL_COMMUTER_CODE	Error installing the remote authorization code.	The commuter authorization could not be installed on the commuter's computer.
85		VLS_UNABLE_TO_GET_MACHINE_ID_STRING	Error getting the locking information for the client.	Unable to compute the commuter's computer fingerprint.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
86		VLS_INVALID_MACHINEID_STR	Invalid remote locking code string.	The requested machine ID string (fingerprint) could not be calculated from the remote computer's locking criteria passed to the call.
87		VLS_EXCEEDS_LICENSE_LIFE	Cannot issue commuter code. License expiration is less than the requested days for commuter code.	Commuter authorization expiration is greater than the license expiration itself.
88		VLS_TERMINAL_SERVER_FOUND	Standalone application cannot run on terminal server.	A terminal server was found, you cannot run stand-alone licenses on it.
89		VLS_NOT_APPROPRIATE_LIBRARY	Application is not linked to either integrated or standalone library.	Application is not linked to either the integrated library or the stand-alone library.
90		VLS_INVALID_FILETYPE	Invalid file type.	The filetype specified is invalid.
91		VLS_NOT_SUPPORTED	Application is communicating with an old server.	Application is communicating with an old server that does not support this feature.
92		VLS_INVALID_LICENSE	License string is invalid	License string is invalid

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
93		VLS_DUPLICATE_LICENSE	License string is duplicate	License string is duplicate
94		VLS_INSUFFICIENT_USER_CAPACITY	Insufficient user capacity	License server does not currently have sufficient user capacity available for this team member.
95		VLS_TEAM_LIMIT_EXHAUSTED	Team limit exhausted	Team limit exhausted
96		VLS_INSUFFICIENT_TEAM_CAPACITY	Insufficient team capacity	License server does not currently have sufficient team capacity available.
97		VLS_CANNOT_DELETE_UPGRADED_LIC	Cannot delete an upgraded license.	Cannot delete an upgraded license.
98		VLS_UPGRADE_NOT_ALLOWED	The feature does not allow upgrade.	The feature cannot be upgraded.
99		VLS_FEATURE_MARKED_FOR_DELETION	The feature exists on this system and is already marked for a commuter check-in.	The feature exists on this system and is already marked for a commuter check-in.
101		VLS_TEAM_EXCLUDED	The team has been excluded from accessing the feature.	The team has been excluded from accessing the feature.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
102		VLS_NETWORK_SRVR	A network server contacted for standalone server related information	
103		VLS_PERPETUAL_LICENSE	The contacted feature is a Perpetual License	The feature has a perpetual license associated with it.
104		VLS_COMMUTER_CHECKOUT	The commuter code has been checked out	The commuter license code has been checked out.
105		VLS_REVOKE_ERR_NO_FEATURE	License with given feature name and feature version does not exist on the server.	License with the given feature/version does not exist on the license server.
106		VLS_REVOKE_ERR_CORRUPT_MESSAGE	The client message received by the server was corrupted.	The message received by the license server was corrupted.
107		VLS_REVOKE_ERR_OUT_OF_VALID_RANGE	The received number Of licenses to revoke is out of range.	The revocation request was beyond the range.
108		VLS_REVOKE_ERR_MD5_PLUGIN_LOAD_FAIL	Error Loading the MD5 plug-in DLL at the server.	An error was encountered while loading the MD5 plug-in DLL at the license server.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
109		VLS_REVOKE_ERR_MD5_PLUGIN_EXEC_FAIL	Error in executing the authentication plug-in.	An error was encountered in executing the authentication plug-in.
110		VLS_REVOKE_ERR_INSUFFICIENT_FEATURE_LICENSES	This feature has less number of total licenses.	The feature has less number of total licenses.
111		VLS_REVOKE_ERR_INSUFFICIENT_DEFAULT_GROUP	Default group does not has sufficient licenses, reconfigure your user reservation file.	The default group does not have sufficient licenses. You can reconfigure your user reservation file.
112		VLS_REVOKE_ERR_INSUFFICIENT_FREE_IN_DEFAULT	Currently required number of licenses are not free for revoke in the default group.	Currently the required number of licenses are not free for revocation in the default group.
113		VLS_REVOKE_ERR_INVALID_SESSION_ID	Invalid session id received from in the client.	An invalid session ID was sent by the client in packet.
114		VLS_REVOKE_ERR_INVALID_PASSWORD	Invalid password for revocation.	The password supplied for revocation was invalid.
115		VLS_REVOKE_ERR_INTERNAL_SERVER	Revocation failed - internal server error.	Revocation Failed! - Internal Server Error.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
116		VLS_REVOKE_ERR_INFIMATE_GRP_DIST	Infinite revocation not possible with enabled group distribution.	It is not possible to revoke infinite licenses with group distribution enabled.
117		VLS_REVOKE_ERR_INFIMATE_LIC_IN_USE	All licenses must be free for infinite revocation.	All licenses must be free for infinite revocation.
118		VLS_REVOKE_ERR_INFIMATE_LIC_FINITE_REQ	License has infinite keys only infinite revoke request allowed.	The license has infinite keys. Only infinite revocation request allowed.
119		VLS_REVOKE_ERR_TICKET_GENERATION	Revocation ticket generation failed.	Failed to generate the revocation ticket.
120		VLS_REVOKE_ERR_CODE_GEN_VERSION_UNSUPPORTED	Revocation not supported on old license codes added on the server.	License revocation is not supported on the older versions of the License Code Generator.
121		VLS_REVOKE_ERR_RDNT_LIC_UNSUPPORTED	Revocation not supported for redundant licenses.	License revocation is not supported for redundant licenses.
122		VLS_REVOKE_ERR_CAPACITY_LIC_UNSUPPORTED	Revocation not supported for capacity licenses.	License revocation is not supported for capacity licenses.

LSAPI Client Function Return Codes (Continued)

Sentinel LM Error Number	Shell Error No.	Return Code	Default Message	Description
123		VLS_REVOKE_ERR_UNEXPECTED_AUTH_CHLG_PKT	Error unexpected challenge packet received from server.	Unexpected error! The challenge packet was received from the license server.
124		VLS_REVOKE_ERR_TRIAL_LIC_UNSUPPORTED	Revocation not supported for trial licenses.	License revocation is not supported for trial licenses.
125		VLS_REQUIRED_LOCK_FIELDS_NOT_FOUND	Not all required lock selectors are available	The local request required locking criteria was not found.
126		VLS_NOT_ENOUGH_LOCK_FIELDS	Total lock selectors available is less than min. number	The minimum number of locking criteria for the local request are not available.
127		VLS_REMOTE_CHECKOUT_NOT_ALLOWED_FOR_PERPETUAL	Remote checkout is not allowed for perpetual licenses	Perpetual licenses cannot be remotely checked out.
128		VLS_GRACE_LIC_INSTALL_FAIL	Feature cannot run as installation of grace license is failed.	Unable to install the grace period license on the client system at the time of license request.
129		VLS_NOT_SUPPORTED_IN_NONET_LIBRARY	The API is not supported in no-net library.	This API function is not supported with no-net mode of communication.

Appendix D

Error Codes for License Generation Functions

The following table lists Sentinel LM license code generation return codes and their default messages:

Sentinel LM License Code Generation Return Codes

Error No	Return Code	Default Message	Description
0	VLScg_SUCCESS	Success	Success
2	VLScg_NO_FEATURE_NAME	Feature Name must be specified. It cannot be empty.	If feature_name is NULL.
3	VLScg_INVALID_INT_TYPE	Expected an integer value, found "XXX"	If value is not numeric.
4	VLScg_EXCEEDS_MAX_VALUE	Value entered is greater than the maximum supported value.	If value exceeds maximum value.
5	VLScg_LESS_THAN_MIN_VALUE	Value entered is less than the minimum supported value.	If value is less than the minimum value.
6	VLScg_EXCEEDS_MAX_STRLEN	Length of <value> is greater than <value>.	Length exceeds the maximum string length.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
7	VLScg_NOT_MULTIPLE	Value should be a multiple of "XXX".	If value is not a correct multiple.
8	VLScg_INVALID_DEATH_YEAR	Expiration date cannot be less than "XXX".	If year is invalid.
9	VLScg_INVALID_BIRTH_YEAR	Start year cannot be less than "XXX".	If year is too early.
10	VLScg_INVALID_DATE	Date is not valid.	If value is not valid for the month.
11	VLScg_INVALID_HEX_TYPE	Wrong value entered. (Should be hexadecimal)	If value is not in hexadecimal format.
12	VLScg_INVALID_IP_TYPE	Wrong value entered. IP address should be specified in dot form.	If value is not in dot format.
13	VLScg_INVALID_YEAR	Invalid year entered.	If year is invalid.
14	VLScg_RESERV_STR_ERR	The string is a reserved string.	If the string is a reserved string.
15	VLScg_INVALID_RANGE	Value violates the valid range of input.	If value is not in the range allowed and if value is not a valid character.
16	VLScg_INVALID_CHARS	Invalid characters.	If string is not valid.
17	VLScg_SHORT_STRING	License string \"<value>\" too small to parse.	License string too small to parse.
18	VLScg_PREMATURE_TERM	Premature termination of license code. Please check.	Premature termination of license string. Please check.
19	VLScg_REMAP_DEFAULT	Failed to remap default strings from configuration file for license \"<value>\".	Failed to remap default strings from configuration file for license.
20	VLScg_DECRYPT_FAIL	Decryption failed for license code.	Decryption failed for license code.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
21	VLScg_DYNAMIC_DECRYPT_FAILURE	Decryption failed for dynamically added license code.	Decryption failed for dynamically added license string.
22	VLScg_INVALID_CHKSUM	Checksum validation failed for license code. Please verify the license code.	Checksum validation failed for license string.
23	VLScg_FIXED_STR_ERROR	Default fixed string error.	Default fixed string error.
24	VLScg_SECRET_DECRYPT_FAILURE	Decryption failed for secrets. Verify the configuration file for readable licenses.	Decryption failed for secrets.
25	VLScg_SIMPLE_ERROR	Error in license code. Please check.	Error in license string. Please check.
26	VLScg_MALLOC_FAILURE	Out of heap memory.	Out of heap memory.
27	VLScg_INTERNAL_ERROR	Internal error.	Internal error.
28	VLScg_UNKNOWN_LOCK	Unknown lock mechanism.	If the locking criteria is unknown.
29	VLScg_VALUE_LARGE	Value: <value> too large	Value too large
30	VLScg_INVALID_INPUT	Invalid input.	If either <i>codeP</i> or <i>flag</i> are NULL.
31	VLScg_MAX_LIMIT_CROSSED	Maximum limit crossed.	No more handles left.
32	VLScg_NO_RESOURCES	No resources left.	If no resources are available.
33	VLScg_BAD_HANDLE	Bad file handle.	Bad file handle.
34	VLScg_FAIL	Operation failed.	If operation failed.
35	VLScg_INVALID_VENDOR_CODE	Invalid Vendor Code. Please contact your Sentinel LM distributor.	If vendor identification is illegal.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
36	VLScg_VENDOR_ENCRYPTION_FAIL	Vendor-customized encryption failed.	If vendor-customized encryption fails.
37	VLScg_INVALID_EXP_DATE	License Expiration Date must be greater than Start Date.	Expiration Date must be greater than Start Date.
38	VLScg_INVALID_EXP_YEAR	License Expiration Year must be greater than Start Year.	License Expiration Year must be greater than Start Year.
39	VLScg_INVALID_EXP_MONTH	License Expiration Month must be greater than Start Month.	License Expiration Month must be greater than Start Month.
40	VLScg_LICMETER_EXCEPTION	Unknown exception in accessing Sentinel LM license meter(s).	Unknown value in accessing the license meter.
41	VLScg_LICMETER_DECREMENT_OK	Your Sentinel LM license meter(s) have been decremented by <value> units. You now have <value> units left out of an initial count of <value> units.	Your Sentinel LM license meter(s) have been decremented.
42	VLScg_LICMETER_ACCESS_ERROR	Error accessing Sentinel LM license meter(s). Please make sure the Sentinel System Driver is properly installed and a license meter is attached to the parallel port or USB port.	Error accessing the license meter.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
43	VLScg_LICMETER_COUNTER_TOLOW	Too few units (Normal License Count=%d/ Trial License Count= %d) left in your Sentinel LM license meter(s) to generate requested license. %d units required.	Few units left in your Sentinel LM license meter(s) to generate requested license.
44	VLScg_LICMETER_CORRUPT	Your Sentinel LM license meter(s) are corrupted.	License meter is corrupted.
45	VLScg_LICMETER_VERSION_MISMATCH	Your Sentinel LM license meter has an invalid version.	License meter has an invalid version.
46	VLScg_LICMETER_EMPTY	All <value> units of your Sentinel LM license meter(s) have been used up. License generation will fail.	Sentinel LM license meter(s) have been used up.
47	VLScg_PORTSERV_EXCEPTION	Unknown exception (<value>) in accessing Sentinel LM portable server(s).	Unknown exception in accessing Sentinel LM license server(s) for commuter licenses.
48	VLScg_PORTSERV_ACCESS_ERROR	Error accessing Sentinel LM portable server(s). Please make sure one is attached.	Error accessing Sentinel LM license server(s) for a commuter license.
49	VLScg_PORTSERV_VERSION_MISMATCH	Your Sentinel LM license server has an invalid version (<value>.<value>) for commuter licenses. Expected <value>.<value>.	Your Sentinel LM portable server has an invalid version.
50	VLScg_PORTSERV_CORRUPT	Your Sentinel LM portable server(s) are corrupted.	Your Sentinel LM license server(s) for commuter licensing is corrupted.
51	VLScg_EXPIRED_LICENSE	Your software license has expired.	Your software license has expired.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
52	VLScg_INVALID_LICTYPE	Invalid License Type.	Invalid License Type.
53	VLScg_INVALID_TRIALDAYS	Invalid Trial Days.	Invalid Trial Days.
54	VLScg_INVALID_TRIAL_COUNT	Invalid Trial License Count.	Invalid Trial License Count.
55	VLScg_TRIALMETER_EMPTY	All <value> units of your Sentinel LM Trial license meter(s) have been used up.	All units of your Sentinel LM Trial license meter(s) have been used up.
56	VLScg_TRIAL_SUCCESS	Your Sentinel LM Trial license meter(s) have been decremented by <value> units. You now have <value> units left.	Your Sentinel LM Trial license meter(s) have been decremented.
57	VLScg_NO_NETWORK_AUTHORIZATION	Your Sentinel LM license meter(s) have No authorization to generate Network Licenses.	Server does not recognize this network.
58	VLScg_NO_ENABLE_FEATURE	No feature is enabled.	Enable feature not specified.
59	VLScg_VI18N_INITIALIZE_FAIL	Error in updating locale.	Error in updating locale.
60	VLScg_INVALID_NUM_SERVERS	Invalid number of servers.	Invalid number of servers.
61	VLScg_NO_CAPACITY_AUTHORIZATION	Your Sentinel LM license meter(s) have no authorization to generate capacity licenses.	Your Sentinel LM license meter(s) have no authorization to generate capacity licenses.
62	VLScg_UPGRADE_NOT_ALLOWED	Your Sentinel LM license meter(s) have no authorization to generate upgrade licenses.	Your Sentinel LM license meter(s) have no authorization to generate upgrade licenses.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
70	VLScg_LICMETER_NOT_SUPPORTED	Your Sentinel LM License Meter is not supported.	Your Sentinel LM License Meter is not supported.
71	VLScg_INCORRECT_BASE_FEATURE_VAL	The base feature is too high for supporting desired number of multi-features.	The base feature is too high for supporting desired number of multi-features.
72	VLScg_ENCRYPTION_FAIL	License code encryption failed.	License code encryption failed.
73	VLScg_INVALID_CAPACITY_SETTINGS	Invalid capacity setting done in code structure.	Invalid capacity setting done in code structure.
74	VLScg_EXP_DATE_BEFORE_START_DATE	License expiration date is set before the license start date.	License expiration date is set before the license start date.

Appendix E

Error Codes for Upgrade License Functions

The following table lists upgrade license code generation return codes and their default messages:

Upgrade License Code Generation Return Codes

Error No	Return Code	Default Message	Description
0	VLScg_SUCCESS	Success	Success
2	VLSucg_NO_FEATURE_NAME	Feature Name must be specified. It cannot be empty.	If feature_name is NULL.
3	VLSucg_INVALID_INT_TYPE	Expected an integer value, found "XXX"	If value is not numeric.
4	VLSucg_EXCEEDS_MAX_VALUE	Value entered is greater than the maximum supported value.	If value exceeds the maximum value.
5	VLSucg_LESS_THAN_MIN_VALUE	Value entered is less than the minimum supported value.	If value is less than the minimum value.

Upgrade License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
6	VLSucg_EXCEEDS_MAX_STRLEN	Length of <value> is greater than <value>.	Length of the string is greater than maximum supported.
7	VLSucg_NOT_MULTIPLE	Value should be a multiple of "XXX".	If value is not a correct multiple.
11	VLSucg_INVALID_HEX_TYPE	Wrong value entered. (Should be hexadecimal)	If value is not in hexadecimal format.
14	VLSucg_RESERV_STR_ERROR	<Value> is a reserved string.	The specified string is a reserved string.
16	VLSucg_INVALID_CHARS	Invalid characters.	Invalid characters in feature_name.
20	VLSucg_DECRYPT_FAIL	Decryption failed for license code.	Decryption failed for license code.
22	VLSucg_INVALID_CHKSUM	Checksum validation failed for license code. Please verify the license code.	Checksum validation failed for license string.
26	VLSucg_MALLOC_FAILURE	Out of heap memory.	If error occurs while allocating internal memory for ucodeT struct.
27	VLSucg_INTERNAL_ERROR	Internal error.	If any internal error occurs while generating the license string.
30	VLSucg_INVALID_INPUT	Invalid input.	If ucodeP is passed as NULL.
31	VLSucg_MAX_LIMIT_CROSSED	Maximum limit crossed.	Library has crossed the limit of maximum handles it can allocate.
32	VLSucg_NO_RESOURCES	No resources left.	If no resources are available.
33	VLSucg_BAD_HANDLE	Bad file handle.	If the handle passed is not a valid handle.

Upgrade License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
34	VLSucg_FAIL	Operation failed.	On Failure
35	VLSucg_INVALID_VENDOR_CODE	Invalid Vendor Code. Please contact your Sentinel LM distributor.	If vendor identification is illegal.
36	VLSucg_VENDOR_ENCRYPTION_FAIL	Vendor-customized encryption failed.	If vendor-customized encryption fails.
40	VLSucg_LICMETER_EXCEPTION	Unknown exception in accessing Sentinel LM license meter(s).	If error occur while accessing the dongle.
41	VLSucg_LICMETER_DECREMENT_OK	Your Sentinel LM license meter(s) have been decremented by <value> units. You now have <value> units left out of an initial count of <value> units.	Your Sentinel LM license meter(s) have been decremented.
42	VLSucg_LICMETER_ACCESS_ERROR	Error accessing Sentinel LM license meter(s). Please make sure the Sentinel System Driver is properly installed and a license meter is attached to the parallel port or USB port.	Error accessing the license meter.
44	VLSucg_LICMETER_CORRUPT	Your Sentinel LM license meter(s) are corrupted.	License meter is corrupted.
45	VLSucg_LICMETER_VERSION_MISMATCH	Your Sentinel LM license meter has an invalid version.	License meter has an invalid version.
46	VLSucg_LICMETER_EMPTY	All <value> units of your Sentinel LM license meter(s) have been used up. License generation will fail.	Sentinel LM license meter(s) have been used up.

Upgrade License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
52	VLSucg_INVALID_LICTYPE	Invalid License Type.	Invalid License Type.
59	VLSucg_VI18N_INITIALIZE_FAIL	Error in updating locale.	Error in updating locale.
61	VLSucg_NO_CAPACITY_AUTHORIZATION	Your Sentinel LM license meter(s) have no authorization to generate Capacity Licenses.	If not authorized to generate capacity licenses.
62	VLSucg_NO_UPGRADE_AUTHORIZATION	Your Sentinel LM license meter(s) have no authorization to generate Upgrade Licenses.	If not authorized to generate upgrade licenses.
63	VLSucg_NO_UPGRADE_CODE	Upgrade Code must be specified. It cannot be empty.	If the <i>upgrade_code</i> is passed as NULL or empty string.
64	VLSucg_INVALID_BASE_LIC_INFO	The information-feature name, version and vendor code, provided for base license is incorrect.	The information-feature name, version vendor code provided for base license is incorrect.
65	VLSucg_CAPACITY_UPD_NOT_ALLOWED	Capacity upgrade is not allowed, as the base lic is a non-capacity license.	Capacity upgrade is not allowed, as the base license is a non-capacity license.
66	VLSucg_INVALID_UPGRADE_CODE	Invalid upgrade code.	Invalid upgrade code.
67	VLSucg_LICMETER_COUNTER_TOLOW	Too few units (Normal License Count=%d) left in your Sentinel LM license meter(s) to generate requested license. %d units required.	If license meter count is less than the expected decrement count.
69	VLSucg_LICMETER_NOT_SUPPORTED	Your Sentinel LM License Meter is not supported.	Your Sentinel LM License Meter is not supported.

Appendix F

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 Sentinel LM 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.
- All characters in a license code must be 7-bit ASCII. This means no double-byte characters or accented characters may be used.

- 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 are 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.

We suggest you to refer to *Sentinel LM System Administrator Online Guide* for more details on alert specifications.

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 Sentinel LM utility, **lsusage**. Reports for encrypted log files can be generated by developers only using the **vusage** utility. See the *Sentinel LM 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 *Sentinel LM Developer's Guide* and the *Sentinel LM 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:

Log Entry Format

Server-LFE	License-LFE	Date	Time-stamp	Feature	Ver	Capacity Flag	Capacity	Trans	Num-keys	Keylife	User	Host	Team Capacity	User Capacity	LSver	Currency	Comment
------------	-------------	------	------------	---------	-----	---------------	----------	-------	----------	---------	------	------	---------------	---------------	-------	----------	---------

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.
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.
Capacity Flag	To show whether the licenses is a capacity license or a non-capacity license. <ul style="list-style-type: none"> • 0-non-capacity license • 1-non-pooled capacity license • 2-pooled capacity license
Capacity	Capacity per token for non-pooled license, capacity of license for a pooled license and for a non-capacity license.
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.

Elements of a Log File (Continued)

Element	Description
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 Sentinel LM license server.
Team capacity	Team Capacity issued against the request.
User capacity	User Capacity issued against the request.
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:

```
1 1 ODA= Mon Mar 17 14:06:12 2003 1047890172 bounce v
1 1000 0 2 0 jsmith jsmith-xp 7.3.0 1 - - - - - 1000
200 MQ== 1695726 MTgwNzI4MA==
```

This entry indicates that *Monday, March 17, 2003, at 14:06:12*, the user, *jsmith* finished using an application with the feature *bounce*, version *1*, *non pooled capacity* license and 1000 capacity per token. The license was returned after being used on computer *jsmith-xp*. 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 *7.3.0*, and *1* license token was used by the application. The team capacity being 1000 and the user capacity being 200.

If the maximum size of the log file has been specified using the **-z** option, Sentinel LM 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 8
 - feature licensing information 115, 117
 - security 404–407
- advanced client functions 41
- APIs
 - adding 8
 - advanced 2
 - capacity 333
 - client 21–141
 - client example 3
 - commuter 319
 - license code generation 143
 - queuing 296
 - quick 1
 - redundancy 260
 - standard 2
 - upgrade license code generator 353, 381
- applications
 - sample 18–398
- authenticating the license manager 50–51

B

- basic client licensing functions 24–29
- basic license code generation
 - functions 156
- broadcast intervals

- retrieving 68
- setting 67

C

- CHALLENGE structure, defined 50
- challenge-response mechanism 50–51
- CHALLENGERESPONSE structure,
 - defined 50
- changing
 - port number default 415–416
 - system time 404–407
- checking out remote commuter
 - authorization 328
- client API 21–141
 - example 3
- client configuration functions 2, 52
- client feature information, retrieving 86,
 - 89
- client libraries 22
- client library
 - initializing 29
 - retrieving information 121
 - tracing calls 141
- client query functions 2, 83
- client utility functions 2, 111–124
- clock, detecting changes 20, 404
- code struct field setting functions 148–206
- codeT 143

- commuter authorization, remote 328
- commuter licensing 319, 326
- configuration files
 - format of 468–472
- conventions, manual xxi
- custom host IDs, creating 416–420
- customizing functions 399
- customizing license file name 422
- customizing Sentinel LM
 - changing port numbers 415–416
 - creating a custom host ID 416–420
 - detecting time tampering 404–407
 - license code encryption 407–411
 - message encryption 412–415

D

- DECRYPT_LIC_OBJ Makefile variable 409
- DECRYPT_MSG_OBJ Makefile variable 412
- decrypting
 - license codes 407–411
 - messages 412–415
- deleting
 - feature licensing information 119
- destroying the handle for lscgen.h 157
- disable auto timer 82
- displaying error messages 138, 140
- dynamic switching 13

E

- ENCRYPT_LIC_OBJ Makefile variable 409
- ENCRYPT_MSG_OBJ Makefile variable 412
- encrypting
 - license codes 407–411
 - messages 412–415
- environment variables
 - LS_MAX_GRP_QLEN 303
 - LS_MAX_HOLD_SEC 303
 - LS_MAX_QLEN 303
 - LS_MAX_WAIT_SEC 303
 - LSDEFAULTDIR 467

- LSERVRC 472
- LSERVRC CNF 472
- LSFORCEHOST 10
- LSHOST 16, 54

- error codes
 - client functions 431
 - upgrade license generation
 - functions 461
- error handlers 17
- error handling 136–141
 - setting 139
- error handling functions 2
- error message display 140
- error messages, displaying 138
- errors, retrieving 158–162
- event handlers, registering with the server 402
- example files 402
- export information xxv

F

- feature licensing information
 - adding 115, 117
 - deleting 119
 - retrieving 100
- feature names, retrieving 104
- feature query functions 2, 91–111
- feature time left information
 - retrieving 108
- FeatureName parameter 3, 9
- file formats 467–474
 - configuration 468–472
 - license codes 467
 - log 472–474
- files
 - lservrc 467
 - lshost 54
- functions
 - basic client 24–29
 - capacity license 333

- client configuration 2, 52
- client query 2, 83
- client utility 2, 111–124
- commuter license 319
- customizing 399
- error handling 2
- feature query 2, 91–111
- license queuing 296
- redundancy 260
- upgrade license 353

G

- get remote computer locking info 326

H

- hold time
 - setting 70
- host ID
 - customizing 416–420
 - setting 66
- host names
 - retrieving 56
 - setting 53

I

- initializing fields of the machineID 58
- initializing the client library 29
- initializing the server 401
- initializing the server info 66
- installing remote commuter
 - authorization 330

K

- key time left information
 - retrieving 110
- keys
 - renewing 35

L

- libraries
 - client 22

- integrated 22
- network 22
- stand-alone 22
- UNIX 16
- license code generation API 143
- license codes
 - encrypting and decrypting 407–411
 - file format 467
- license generation function return
 - codes 453
- license manager
 - authenticating 50–52
 - usage logging 472–474
- license server
 - APIs
 - license code generation* 143
 - locating 112
- LICENSE_LIBS macro 17
- licenses
 - lifetime of 36
 - local vs. remote renewal of 76
 - releasing 39, 45
 - renewing 35
 - requesting 31, 42
 - single-call licensing
 - disabling* 28
- lifetime of a license 36
- local license renewal 77
- locating the license server 112
- log file format 472–474
- LS_LIBVERSION structure, defined 121
- LS_MAX_QLEN 303
- LSAPI client function return codes 431
- lscgen.h handle
 - destroying 157
- LSDEFAULTDIR environment variable 467
- lserv.h file 17
- LSERVRC environment variable 472
- lservrc file 467

LSFORCEHOST environment variable 10, 14
LSGetMessage 138
LSHOST environment variable 14, 16, 54
lshost file 54
LSRelease 39
LSRequest 31
LSUpdate 35
lsusage utility 472

M

machine names, retrieving 112
macros
 LICENSE_LIBS 17
 NO_LICENSE 6, 17
Makefile 17–18, 400
Makefile variables
 DECRYPT_LIC_OBJ 409
 DECRYPT_MSG_OBJ 412
 ENCRYPT_LIC_OBJ 409
 ENCRYPT_MSG_OBJ 412
messages, encrypting and decrypting 412–415

N

network mode 13
NO_LICENSE macro 17
NO-NET 14

P

port numbers
 changing the default 415–416
 retrieving 58
printing errors 158–162
programs, sample 18–19
PublisherName parameter 9

Q

quick client functions 24
Quick-API 1

R

redundant license server 259
registering an event handler 402
releasing licenses 39, 45
remote commuter authorization 328
remote renewal period 36
remote renewal time, setting 81
renewing license keys 35
requesting licenses 31, 42
retrieving
 broadcast intervals 68
 client feature information 86, 89
 client library information 121
 errors 158–162
 feature licensing information 100
 feature names 104
 feature time left information 108
 license time left information 110
 machine names 112
 server host names 56
 server port numbers 58
 time drift information 107
 time-out intervals 69
 version information 103, 105
Returns 40

S

sample applications 18–397
sample programs 18
sample32.mak file 18
security
 adding 19, 404–407, 412
Sentinel LM
 APIs
 capacity license 333
 client 21–141
 commuter license 319
 license code generation 143
 license queuing 293
 redundancy 259

- upgrade license* 353
- architecture 1–2
- customizing
 - changing port numbers* 415–416
 - creating a custom host ID* 416–420
 - detecting time tampering* 404–407
 - license code encryption* 407–412
 - message encryption* 412–415
- security 19
 - adding* 404–407, 412
- servers
 - detecting 10
 - initializing 401
 - retrieving host names 56
 - retrieving port numbers 58
 - setting
 - host names* 53
- setting
 - broadcast intervals 67
 - code struct fields 148–206
 - error handling 139
 - hold time 70
 - host ID 66
 - remote renewal time 81
 - server names 53
 - time-out intervals 68
- shared IDs 71, 74
- shutting down lserv 122
- single-call licensing
 - disabling 28
- stand-alone mode 13
- standard client functions 29
- Standard-API 2
- structure definitions
 - CHALLENGE 50
 - CHALLENGERESPONSE 50
 - LS_LIBVERSION 121
 - VLSclientInfo 84
 - VLSfeatureInfo 94
 - VLSrevocationTicketInfoT 256
 - system time, detecting changes 20, 404–407
- T**
 - time clock, detecting changes 20, 404–407
 - time drift information
 - retrieving 107
 - time-out intervals
 - retrieving 69
 - setting 68
 - tracing client-library calls 141
 - tracing Sentinel LM operation 141
- U**
 - unicodeT Struct 355
 - ulcCode 382
 - UNIX
 - libraries 16
 - Makefile 17–18
 - updating 47
 - updating licenses 47
 - Upgrade License Code Generation Return Codes 461
 - usage logging 472–474
 - using the Sentinel LM client API 21
 - utilities
 - lsusage 472
- V**
 - variable 472
 - variables
 - environment
 - LSDEFAULTDIR 467
 - LSERVRC 472
 - LSFORCEHOST 10
 - LSHOST 16, 54
 - Makefile
 - DECRYPT_LIC_OBJ 409

- DECRYPT_MSG_OBJ* 412
- ENCRYPT_LIC_OBJ* 409
- ENCRYPT_MSG_OBJ* 412
- version information
 - retrieving 103, 105
- Version parameter 3, 9
- VLsaddFeature 115, 262
- VLsaddFeatureExt 264
- VLsaddFeatureToFile 117, 265
- VLsaddServerToPool 267
- VLsbatchUpdate 47
- VLScgAllowAdditive 173
- VLScgAllowCapacity 212
- VLScgAllowCapacityLic 210
- VLScgAllowClientLockInfo 201
- VLScgAllowClockTamperFlag 203
- VLScgAllowCodegenVersion 209
- VLScgAllowCommuterLicense 184
- VLScgAllowCommuterMaxCheckoutDays 186
- VLScgAllowFeatureName 168
- VLScgAllowFeatureVersion 170
- VLScgAllowGracePeriod 242
- VLScgAllowGracePeriodFlag 241
- VLScgAllowHeldLic 208
- VLScgAllowKeyHoldtime 231
- VLScgAllowKeyHoldUnits 230
- VLScgAllowKeyLifetime 175
- VLScgAllowKeyLifeUnits 228
- VLScgAllowKeysPerNode 221
- VLScgAllowLicBirth 233
- VLScgAllowLicenseType 171
- VLScgAllowLicExpiration 236
- VLScgAllowLocalRequestLockCrit 247
- VLScgAllowLocalRequestLockCritFlag 246
- VLScgAllowLockMechanism 199
- VLScgAllowLockModeQuery 188
- VLScgAllowLogEncryptLevel 179
- VLScgAllowMajorityRuleFlag 191
- VLScgAllowMultiKey 216
- VLScgAllowMultipleServerInfo 193
- VLScgAllowNetworkFlag 177
- VLScgAllowNumKeys 186–187
- VLScgAllowOutLicType 205
- VLScgAllowOverdraft 245
- VLScgAllowOverdraftFlag 244
- VLScgAllowPerpetualFlag 177
- VLScgAllowRedundantFlag 190
- VLScgAllowSecrets 217
- VLScgAllowServerLockInfo 194
- VLScgAllowSharedLic 180
- VLScgAllowShareLimit 182
- VLScgAllowSiteLic 223
- VLScgAllowSoftLimit 227
- VLScgAllowStandAloneFlag 176
- VLScgAllowTeamCriteria 180
- VLScgAllowTrialLicFeature 172
- VLScgAllowVendorInfo 220
- VLScgCleanup 157
- VLScgDecodeLicenseRevocationTicket 254
- VLScgGenerateLicense 249–250
- VLScgGetErrorLength 160
- VLScgGetErrorMessage 160
- VLScgGetLicenseMeterUnits 252
- VLScgGetNumErrors 159
- VLScgGetTrialLicenseMeterUnits 253
- VLScgInitialize 157
- VLScgPrintError 162
- VLScgReset 158
- VLScgSetAdditive 174
- VLScgSetCapacityFlag 211
- VLScgSetCapacityUnits 213
- VLScgSetClientLockInfo 201
- VLScgSetClientLockMechanism 200
- VLScgSetClientServerLockMode 189
- VLScgSetClockTamperFlag 204
- VLScgSetCodegenVersion 210

- VLScgSetCodeLength 167
- VLScgSetCommuterLicense 185
- VLScgSetCommuterMaxCheckoutDays 186
- VLScgSetFeatureName 169
- VLScgSetFeatureVersion 170
- VLScgSetGracePeriodDays 243
- VLScgSetGracePeriodFlag 242
- VLScgSetGracePeriodHours 243
- VLScgSetHoldingCrit 208
- VLScgSetKeyHoldtime 232
- VLScgSetKeyHoldtimeUnits 230
- VLScgSetKeyLifetime 175
- VLScgSetKeyLifetimeUnits 229
- VLScgSetKeysPerNode 222
- VLScgSetKeyType 216
- VLScgSetLicBirthDay 234
- VLScgSetLicBirthMonth 233
- VLScgSetLicBirthYear 235
- VLScgSetLicenseType 171
- VLScgSetLicExpirationDay 238
- VLScgSetLicExpirationMonth 237
- VLScgSetLicExpirationYear 239
- VLScgSetLicType 207
- VLScgSetLoadSWLicFile 241
- VLScgSetLocalRequestLockCrit 248
- VLScgSetLocalRequestLockCritFlag 247
- VLScgSetLogEncryptLevel 179
- VLScgSetMajorityRuleFlag 191
- VLScgSetNumClients 202
- VLScgSetNumericType 240
- VLScgSetNumFeatures 224–226
- VLScgSetNumKeys 187
- VLScgSetNumSecrets 219
- VLScgSetNumServers 193
- VLScgSetOutLicType 205
- VLScgSetOverdraftFlag 244
- VLScgSetOverdraftHours 245
- VLScgSetOverdraftUsers 246
- VLScgSetRedundantFlag 190
- VLScgSetSecrets 218
- VLScgSetServerLockInfo1 195
- VLScgSetServerLockInfo2 198
- VLScgSetServerLockMechanism1 196
- VLScgSetServerLockMechanism2 197
- VLScgSetSharedLicType 181
- VLScgSetShareLimit 183
- VLScgSetSiteLicInfo 223
- VLScgSetSoftLimit 227
- VLScgSetStandAloneFlag 178
- VLScgSetTeamCriteria 181
- VLScgSetTrialDaysCount 173
- VLScgSetVendorInfo 220
- VLScchangeDistbCrit 268
- VLScchangePortNumber 415
- VLScchangeUsageLogFileName 387
- VLSCleanup 41
- VLSclientInfo 84
- VLSccommuterInfo struct 320
- VLScconfigureTimeTamper 404–405
- VLScdecodeUpgradelockCode 383
- VLScdecryptLicense 410
- VLScdecryptMsg 414
- VLScdeleteFeature 119
- VLScdeleteFeatureExt 348
- VLScdelServerFromPool 269
- VLScdisableAutoTimer 82
- VLScdisableEvents 390
- VLScdisableLicense 28
- VLScdiscover 11, 112
- VLScdiscoverExt 272
- VLScenableLocalRenewal 77
- VLScencryptLicense 408
- VLScencryptMsg 412
- VLScerrorHandle 137
- VLSceventAddHook 401–402
- VLSceventSleep 391
- VLScfeatureInfo 94

VLsgenerateUpgradeLockCode 380
VLSgetAndInstallCommuterCode 323
VLSgetBroadcastInterval 68
VLSgetCapacityFromHandle 350
VLSgetCapacityList 344
VLSgetClientInfo 86
VLSgetClientInfoExt 346
VLSgetCommuterCode 328
VLSgetCommuterCode call 328
VLSgetCommuterInfo 322
VLSgetContactServer 56
VLSgetDistbCrit 275
VLSgetDistbCritToFile 277
VLSgetFeatureFromHandle 104
VLSgetFeatureInfo 100
VLSgetFeatureInfoExt 342
VLSgetFeatureInfoToFile 279
VLSgetFeatureTimeLeftFromHandle 108
VLSgetHandleInfo 89
VLSgetHandleStatus 311
VLSgetHostAddress 284
VLSgetHostName 281
VLSgetKeyTimeLeftFromHandle 110
VLSgetLeaderServerName 282
VLSgetLibInfo 121
VLSgetLicInUseFromHandle 90
VLSgetLicSharingServerList 285
VLSgetMachineID 60–61
VLSgetMachineIDString 326–327
VLSgetPoolServerList 287
VLSgetQueuedClientInfo 306
VLSgetQueuedLicense 314
VLSgetServerList 65
VLSgetServerNameFromHandle 63
VLSgetServerPort 58, 415
VLSgetTimeDriftFromHandle 107
VLSgetTimeoutInterval 69
VLSgetTrialPeriodLeft 126
VLSgetUsageLogFileName 388
VLSgetVersionFromHandle 105
VLSgetVersions 103
VLSinitialize 29
VLSinitMachineID 58
VLSinitQueuePreference 317
VLSinitServerInfo 66
VLSinitServerList 64
VLSinstallCommuterCode 330
VLSinstallCommuterCode call 330
VLSisClockSetBack 407
VLSisLocalRenewalDisabled 77
VLSlicense 24
VLSmachineIDtoLockCode 62
VLSqueuedRequest 300
VLSqueuedRequestExt 301
VLSreleaseExt 45
VLSremoveQueue 310
VLSremoveQueuedClient 308
VLSrequestExt 42
VLSrevokeLicense 132
VLSscheduleEvent 389, 393
VLSserverVendorInitialize 401
VLSsetBorrowingStatus 288
VLSsetBroadcastInterval 67
VLSsetContactServer 53
VLSsetFileName 422
VLSsetHoldTime 12, 70
VLSsetRemoteRenewalTime 81
VLSsetServerLogState 290
VLSsetServerPort 57, 415
VLSsetSharedId 71
VLSsetSharedIdValue 73
VLSsetTeamId 71
VLSsetTeamIdValue 73
VLSsetTimeoutInterval 68
VLSsetTraceLevel 17, 141
VLSsetUserErrorFile 140
VLSshutDown 122
VLSsucgAllowBaseFeatureName 363

VLSucgAllowBaseFeatureVersion 365
VLSucgAllowUpgradeCapacity 373
VLSucgAllowUpgradeCode 367
VLSucgAllowUpgradeFlag 369
VLSucgAllowUpgradeVersion 371
VLSucgCleanup 357
VLSucgDecodeLicense 384
VLSucgGenerateLicense 377
VLSucgGetErrorLength 360
VLSucgGetErrorMessage 361
VLSucgGetLicenseMeterUnits 379
VLSucgGetNumErrors 359
VLSucgInitialize 357
VLSucgPrintError 362
VLSucgReset 358
VLSucgSetBaseFeatureName 364
VLSucgSetBaseFeatureVersion 366
VLSucgSetUpgradeCapacity 376
VLSucgSetUpgradeCapacityUnits 374
VLSucgSetUpgradeCode 368
VLSucgSetUpgradeFlag 370
VLSucgSetUpgradeVersion 372
VLSuninstallAndReturnCommuterCode 3
 25
VLSupdateQueuedClient 312
VLSwhere 124

W

Web site, Technical Support xxiii
Wlscgen utility
 additional security 428

