Sentinel LM Programmer's Reference Manual

RAINBOW® TECHNOLOGIES

Part Number 007-0069-001, Revision A
Software versions 7.3.0 and later

Revision	Action/Change	Date
A	Initial Release	January 2004

Rainbow Technologies Sales Offices

United States: *http://www.rainbow.com,* Tel: +1 949 450 7300 / Toll Free: +1 800 852 8569

Australia
Tel: +61 3 9882 8322
http://www.rainbowaustralia.com.au

Brazil
Tel: +55 11 6121 6455
http://www.rainbow.com/brasil

China
Tel: +86 10 8851 9191
http://www.isecurity.com.cn

France
Tel: +33 1 41 43 29 00
http://www.fr.rainbow.com

Germany
Tel: +49 89 32 17 98 0
http://www.de.rainbow.com

Hong Kong
Tel: +852 3157 7111
http://www.rainbow.com

India
Tel: +91 11 26917538
http://www.rainbowindia.co.in

Japan
Tel: +81 3 5719 2731
http://www.rainbowtech.co.jp

Korea
Tel: +82 31 705 8212
http://www.rainbow.com/korea

Mexico
Tel: +52 55 5575 1441
http://www.rainbow.com/latinamerica

Taiwan
Tel: +886 2 6630 9388
http://www.rainbow.com/taiwan

UK
Tel: +44 (0) 1932 579200
http://www.uk.rainbow.com

International Quality Standard Certification

Rainbow Technologies, Inc. Irvine, CA facility has been issued the ISO 9001 Certification, the globally recognized standard for quality, by Det Norske Veritas as of March 2002. Certificate Number CERT-02982-2000-AQ-HOU-RABR2.

European Community Directive Conformance Statement

CE

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

Contents

Preface

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

The Sentinel LM Manuals

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?
Sentinel LM Developer's Guide	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.

Manual	What's in it?	Who should read it?
Sentinel LM Programmer's Reference Manual	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.
Sentinel LM System Administrator's Online Guide	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.
Sentinel LM Start Here Guide	Information to get the developer up and running with Sentinel LM as quickly as possible. It contains installation instructions and a quick tour of Sentinel LM-Shell.	Any developer using Sentinel LM.
Sentinel LM Release Notes	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.

Note: For exact location of Sentinel LM documentation we suggest you to refer to *ReadMeFirst.pdf*.

Who Should Read This Manual?

The *Sentinel LM Programmer's Reference Manua*l 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.
`Courier`	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.
Italic Lettering	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.

How to Get the Most from 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.

Chapter/Appendix	Description
Appendix D – Error and Result Codes for License Generation Functions	Lists license generation function return codes.
Appendix E – Error and Result 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.

Getting Help

If you have questions that were not answered in this manual, please see the following sources for additional help.

Help Files

Several online help files are available for your use. From the **Start** menu, select **Rainbow Technologies** and then select the help file you are interested in.

Online Documentation

This *Sentinel LM Programmer's Reference Manual* you are currently reading is also available in portable document format (PDF) on the Sentinel LM CD in the *\Manuals* directory.

You need Adobe Acrobat Reader 4.0 or later to view and print PDF files. We recommend installing Acrobat Reader 5.0 or higher for better results. This version of Acrobat can be downloaded from *http://www.adobe.com*.

For the Windows versions of Sentinel LM, you may install the Reader from the Sentinel LM release by navigating to the *\Acrobat* directory on the Sentinel LM CD and starting the *AdbeRdr60_enu_full.exe* file. Running that file will install the Acrobat Reader on your hard disk.

For UNIX computers, you can obtain the Acrobat Reader from the Adobe web page, *http://www.adobe.com*. Follow the instructions on that Web page to choose and install the correct Reader for your particular UNIX platform. We also include the Windows version of the Reader on the UNIX release.

Once you have installed Acrobat Reader, you are ready to access the documentation PDF files, which are included on the Sentinel LM CD.

Contacting Rainbow Technologies Technical Support

Rainbow Technologies is committed to supporting the Sentinel LM. If you have questions, need additional assistance, or encounter a problem, please contact Technical Support:

Rainbow Technologies Technical Support Contact Information

Rainbow Technologies Customer Connection Center (C^3)	
http://c3.rainbow.com	
Americas	
Internet	http://www.rainbow.com/support
E-mail	techsupport@rainbow.com
United States	
Telephone	(800) 959-9954
Fax	(949) 450-7450
Europe	
E-mail	EUTechSupport@rainbow.com
France	
Telephone	0825 341000
Fax	44 (0) 1932 570743
Germany	
Telephone	01803 RAINBOW (7246269)
Fax	089 32179850
United Kingdom	
Telephone	0870 7529200
Fax	44 (0) 1932 570743

Rainbow Technologies Technical Support Contact Information (Continued)

Pacific Rim	
E-mail	techsupportpacrim@rainbow.com
Australia and New Zealand	
Telephone	(61) 3 9882 8322
Fax	(61) 3 9820 8711
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 6630 9388
Fax	(886) 2 6630 6858

Tip: Check the Rainbow Technologies Web site (*www.rainbow.com*) for the most up-to-date information about Sentinel LM, including FAQs and technical notes.

Export Considerations

Rainbow Technologies 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 Rainbow'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@rainbow.com

Or, you can write us at:

Rainbow Technologies, Inc.
50 Technology Drive
Irvine, CA 92618

Attn: Technical Publications Department

Thank you for your feedback. Keep in mind that these e-mail and mail addresses are only for documentation feedback. If you have a technical question, please contact Rainbow Technical Support (see "Contacting Rainbow Technologies Technical Support" on page xxv).

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, 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 Rainbow 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 Rainbow's hardware keys. Therefore, the program must be run under Windows, even when generating licenses to be used under UNIX. More generally, all users of the 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 132.) 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:

- VLSinitialize
- 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 VLSinitialize.

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

The next function you want to call is LSRequest.

This API takes several parameters. *PublisherName* identifies your company. *FeatureName* identifies your product and *Version* identifies the version number for that product. 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 267.

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, odbccp32.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
lsapiw32.lib	Dual network and stand-alone client library for Windows applications. This library allows you either to access the stand-alone license locally or acquire a license from a remote license server over the network.
lssrv32.lib	This library is the same as *lsapiw32.lib*.
lsclws32.lib	The network client library for Windows applications. This library allows your application to acquire licenses via network only.
lsnnet32.lib	The stand-alone client library for Windows applications. This library allows you to acquire stand-alone licenses on a local computer only.

Windows Dynamic Linked Libraries and Import Libraries

Windows Dynamic Libraries and Import Libraries

Library	Description
lsapiw32.dll	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.
lsapiw32.lib	This library is the import library for *lsapiw32.dll,* (Microsoft format).
lssrv32.dll	This library is the same as *lsapiw32.dll*.
lsclws32.dll	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 *lsapiw32.dll*, you may use the *lsapiw32.lib* import library supplied with the installation.
lsnnet32.dll	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 *lsapiw32.dll*, you may use the *lsapiw32.lib* 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 397.

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 132.

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	*server.o*
lsdecode	*lsde.o*
lslic	*lslic.c*
lsmon	*lsmon.c*
lswhere	*lswhere.c*
Challenge-response	*crexamp.c, chalresp.[c h], md4.[c h]*

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

Customization Sample Files on Windows

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

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 127.

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
- Challenge-response
- Client configuration
- Client query
- Feature query
- Client utility
- Error handling
- Tracing Sentinel LM operation
- Redundancy
- Queuing
- Commuter
- Capacity

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

Note: Length limitations exist on feature name and version depending on the type of license you want to issue to your customer. See the *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	• *featureName* is NULL • *version* is NULL Both feature and version cannot be NULL at the same time.
VLS_CALLING_ERROR	• *lshandle* 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 *hostName* 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 397.

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
lshandle	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	*lshandle* 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 *hostName* 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 397.

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 397.

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
licenseSystem	Unused. Use LS_ANY as the value of this variable. LS_ANY is specified to indicate a match against installed license systems.
publisherName	A string giving the publisher of the product. Limited to 32 characters and cannot be NULL. Company name and trademark may be used.
featureName	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.
version	*Version* of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 11 characters.
unitsReqd (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 *unitsReqd*. If *unitsReqd* is NULL, a value of 1 unit is assumed.

Argument	Description
logComment	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.
challenge	The challenge structure. If the challenge-response mechanism is not being used, this pointer *must* 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.
lshandle (OUT)	The handle for this request is returned in *lshandle*. 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 *lshandle* 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	• *lshandle* is NULL. • *challenge* argument is non-NULL, but cannot be understood. Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	• *featureName* is NULL • *version* is NULL. Both feature name and version cannot be Null at the same time.
VLS_NO_LICENSE_ GIVEN	• *unitsReqd* is zero • *lshandle* is not a valid handle. License is only available at license server that does not match mode settings, e.g. network license available when stand-alone mode, etc.
VLS_NO_SUCH_ FEATURE	License server does not have license that matches requested feature and version.
LS_INSUFFICIENTUNITS	• License server does not currently have sufficient licensing units for requested feature to grant license. • The *units_reqd* 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.
VLS_APP_NODE_ LOCKED	Requested feature is node locked, but request was issued from unauthorized machine.

LSRequest Error Codes (Continued)

Error Code	Description
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 *hostName* 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 397.

See Also

- “VLSrequestExt2” on page 45

- "Challenge-response Mechanism" on page 50
- "VLSsetTimeoutInterval" on page 67
- "VLSqueuedRequest and VLSqueuedRequestExt" on page 274

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
lshandle	This must be the handle previously returned by the corresponding LSRequest call.
unused1	Unused. Use LS_DEFAULT_UNITS as the value.
unused2	Unused. Use NULL as the value.
unused3	Use NULL as the value.
unused4	Use NULL as the value.

Description

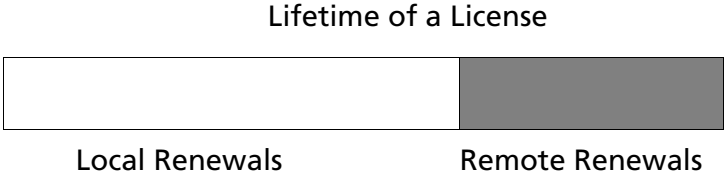
Currently the client library defaults to automatic license renewal. You do not need to call LSUpdate unless you disable automatic license renewals. Even when automatic license renewal is active, you should check periodically on the update status by calling VLSgetRenewalStatus**.** (See "VLSgetRenewalStatus" on page 77.)

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 74 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	• *lshandle* is a queued handle. Cannot use LSUpdate to update a queued handle. • *challenge* 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 *hostName* 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 397.

See Also

- "VLSbatchUpdate" on page 47
- "VLSsetTimeoutInterval" on page 67
- "VLSdisableLocalRenewal" on page 75
- "VLSenableLocalRenewal" on page 76
- "VLSisLocalRenewalDisabled" on page 76
- "VLSgetRenewalStatus" on page 77
- "VLSsetRemoteRenewalTime" on page 80

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
lshandle	The handle returned by the corresponding LSRequest.
units_consumed	Number of units required to be released.
log_comment	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	*lshandle* 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 *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 397.

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 397.

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
licenseSystem	Unused. Use LS_ANY as the value of this variable.
publisherName	• Cannot be Null • A string giving the publisher of the product. Limited to 32 characters. Company name and trademark may be used.
featureName	Name of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 24 characters.
version	*Version* of the feature for which the licensing code is requested. May consist of any printable characters. Limited to 11 characters.
unitsReqd (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 *unitsReqd*. If *unitsReqd* is NULL, a value of 1 unit is assumed.
logComment	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
challenge	The *challenge* structure. If the challenge-response mechanism is not being used, this pointer *must* 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 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.
lshandle	The handle for this request is returned in *lshandle*. 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 *lshandle* must be allocated by the caller.
serverInfo	This information is passed to the license server for use in server hook functions. See "VLSeventAddHook" on page 367.

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

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

VLSrequestExt Error Codes

Error Code	Description
VLS_APP_UNNAMED	• *featureName* is NULL • *version* is NULL Both feature name and version cannot be Null at the same time.
VLS_CALLING_ERROR	• *lshandle* is NULL • *challenge argument* is non-NULL • Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_NO_LICENSE GIVEN	• *unitsReqd* is zero • *lshandle* 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 *hostName* 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 397.

See Also

- "Challenge-response Mechanism" on page 50
- "VLSeventAddHook" on page 367

VLSrequestExt2

See "VLSrequestExt2" on page 306.

VLSreleaseExt

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

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

```
VLSserverInfo    *serverInfo);
```

Argument	Description
lshandle	The handle returned by the corresponding LSRequest.
unused1	Unused. Use the value, LS_DEFAULT_UNITS.
logComment	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.
serverInfo	This information is passed to the license server for use in server hook functions. See "VLSeventAddHook" on page 367.

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	*lshandle* 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 397.

See Also "VLSeventAddHook" on page 367

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
numHandles	Specifies the number of handles.
lshandle (in)	Array of *numHandles* handles, allocated by caller.
unused1	Currently ignored—pass in a NULL.
unused2	Currently ignored—pass in a NULL.
unused3	Use NULL as the value.
unused4	Use NULL as the value.
status (out)	Array of *numHandles* 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	*challenge* 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 *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_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 67
- "VLSdisableLocalRenewal" on page 75
- "VLSenableLocalRenewal" on page 76
- "VLSisLocalRenewalDisabled" on page 76
- "VLSsetRemoteRenewalTime" on page 80

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
ulReserved	LSAPI requires this to be set to 0.
ulChallengedSecret	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.
ulChallengeSize	Number of characters in *ChallengeData*. This value cannot be 0.
ChallengeData	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.
ulResponseSize	Number of characters in the response to the challenge.
ResponseData	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 202, "VLScgSetNumSecrets" on page 203, and "VLScgSetSecrets" on page 202.

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 *machineID* to default values.
VLSgetMachineID	Sets *machineID* values for the current host.
VLSmachineIDtoLockCode	Computes the *machineID* locking code.
VLSgetServerNameFrom Handle	Retrieves the license server's name based on *handle_id*.
VLSinitServerList	Initializes a list of default license servers to search for a license.
VLSgetServerList	Retrieves the default license server list.
VLSinitServerInfo	Initializes the license serverInfo data structure to default values.
VLSsetHostIdFunc	Sets the host ID function.
VLSsetBroadcastInterval	Configures broadcast behavior.
VLSgetBroadcastInterval	Retrieves broadcast behavior parameters.
VLSsetTimeoutInterval	Configures timeout behavior.
VLSgetTimeoutInterval	Retrieves timeout behavior parameters.

Client Configuration Functions (Continued)

Function	Description
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 74.

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
serverName	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
libls.a	Valid host name	Client should communicate with the license server on *serverName*.
	NULL	Client should determine *serverName* using default mechanism.
	NO-NET	Calling error.
libnonet.a	Valid host name	Calling error.
	NULL	Communicate with integrated license server.
	NO-NET	Communicate with integrated license server.
liblssrv.a	Valid host name	Client should communicate with the license server on *serverName*.
	NULL	Client should determine *serverName* 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 397.

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
outBuf (out)	Contains a single license server name, space allocated by caller.
outBufSz	Size of *outBuf*.

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	*outBuf* 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	*outBuf* 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 397.

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
machineID	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	*machineID* is NULL.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
lock_selector_in	User provided mask specifying locking criteria to be read.
machineID	User provided machine ID from which locking criteria will be read.
lock_selector_out	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 397.

VLSmachineIDtoLockCode

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

```
LS_STATUS_CODE  VLSmachineIDtoLockCode(
VLSmachineID    *machineID,
unsigned long   lock_selector,
unsigned long   *lockCode);
```

Argument	Description
machineID	Machine ID used to generate lock code.
lock_selector	Bit mask defining the different lock criteria to retrieve
lockCode	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 397.

VLSgetServerNameFromHandle

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

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

Argument	Description
handle_id	The handle returned by LSRequest or VLSrequestExt
outBuf (OUT)	User allocated buffer to receive license server name
outBufSz	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	*outBuf* is NULL.
LS_BADHANDLE	Invalid handle.
LS_BUFFER_TOO_SMALL	*outBuf* 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 397.

VLSinitServerList

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

```
LS_STATUS_CODE  VLSinitServerList(
char            *serverList,
int             optionFlag);
```

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

Description This function initializes a list of default license servers to contact whenever a call is made to get a license. *serverList* should be in the same format as the last parameter of the VLSdiscover call, and have the same syntax. See "VLSdiscover" on page 107 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 397.

VLSgetServerList

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

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

Argument	Description
outBuf (OUT)	User allocated buffer to receive license server name
outBufSz	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	*outBuf* is NULL.
LS_BUFFER_TOO_SMALL	*outBuf* 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 397.

VLSinitServerInfo

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

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

Argument	Description
serverInfo	User allocated buffer that will contain initialized VLSserverInfo.

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

Note: This function must be called before calling VLSrequestExt or VLSreleaseExt.

Returns The status code LS_SUCCESS is always returned.

VLSsetHostIdFunc

Client	Server	Static Library	DLL
✔		✔	✔

Sets the host ID function.

Syntax

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

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

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
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 397.

VLSsetBroadcastInterval

Client	Server	Static Library	DLL
✔		✔	✔

Sets the broadcast interval.

Syntax

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

Argument	Description
interval	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
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 397.

VLSgetBroadcastInterval

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.

VLSsetTimeoutInterval

Client	Server	Static Library	DLL
✔		✔	✔

Sets the timeout interval.

Syntax

```
LS_STATUS_CODE VLSsetTimeoutInterval(
long           interval);
```

Argument	Description
interval	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 397.

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
featureName	Name of the feature.
version	*Version* of the feature.
timeInSecs	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	• *featureName* is NULL • *version* 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 397.

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
sharedId/ teamId	Must be one of the following values: • VLS_CLIENT_HOST_NAME_ID • VLS_USER_NAME_ID • VLS_X_DISPLAY_NAME_ID • VLS_VENDOR_SHARED_ID
mySharedIdFunc	Pointer to a function that will return the *sharedID* 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
VLS_CALLING_ERROR	*mySharedIdFunc* is NULL.
VLS_UNKNOWN_SHARED_ID	Invalid *sharedId*/ *teamId*; 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 397.

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
sharedId/ teamId	Must be one of the following values: • VLS_CLIENT_HOST_NAME_ID • VLS_USER_NAME_ID • VLS_X_DISPLAY_NAME_ID • VLS_VENDOR_SHARED_ID
sharedIdValue	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 397.

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 397.

See Also

- “LSUpdate” on page 35
- “VLSenableLocalRenewal” on page 76

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 397.

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 75

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 *hostName* 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
featureName	Name of the feature.
version	*Version* of the feature.
timeInSecs	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	• *featureName* is NULL • *version* 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 397.

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
lshandle	The handle returned by LSRequest or VLSrequestExt
state	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 *state*. 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 397.

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 397.

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
user_name	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.
host_id	The host ID of the computer on which the user is working. This can be changed using the VLSsetHostIdFunc call.
group	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, *DefaultGrp* is returned.

Member	Description
start_time	The time at which the current license code was issued by the license server.
hold_time	Specifies the hold time, in seconds, if any.
end_time	The time at which the user's current license will expire if not renewed.
key_id	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.
host_name	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.
x_display_ name	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.
shared_id_ name	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.
num_units	Number of units consumed by the client so far.
q_wait_time	Unused.
is_holding	Has the value, VLS_TRUE, if the user's current license is being held after its expiration. Otherwise, the value is VLS_FALSE.
is_sharing	Total number of clients sharing this particular license, including the current client being queried. If sharing is disabled, *is_sharing* will be 0.
is_commuted	Total number of clients that have "checked out" a license from the network.
structSz	Size of VLSclientInfo structure.

Member	Description
team_capacity	Total capacity of the team.
total_resv_team_capacity	Total capacity reserved for the team.
reserved_team_capacity_in_use	Capacity given to clients from reserved capacity.
ureserved_team_capacity_in_use	Capacity given to clients from unreserved capacity.
user_capacity_from_reserved	Capacity given to users from reserved capacity.
user_capacity_from_unreserved	Capacity given to users from unreserved capacity.
total_team_tokens_resv	Total reserved units.
reserved_team_tokens_in_use	Units given to teams from reserved pool.
unreserved_team_tokens_in_use	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
featureName	Name of the feature.
version	*Version* of the feature.
index	Used to specify a particular client.
unused1	Use NULL as the value.
clientInfo (out)	The structure in which information will be returned. Space allocated by caller.

Description After this call, *clientInfo* contains information about all clients' features. Since it is possible for multiple clients of a particular feature to be active on the network, *index* is used to retrieve information about a specific client. The suggested use of this function is in a loop, where the first call is made with *index* 0 which retrieves information about the first client. Subsequent calls, when made with 1, 2, 3, and so on, will retrieve information about other clients of that feature type. So long as the index is valid, the function returns the success code, LS_SUCCESS. Otherwise, it returns the 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	• featureName is NULL • *version* is NULL Both feature name and version cannot be NULL at the same time.
VLS_CALLING_ERROR	• clientInfo parameter is NULL • *index* 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 *hostName* 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 397.

VLSgetHandleInfo

Client	Server	Static Library	DLL
✔		✔	✔

Returns information about a client feature.

Syntax

```
LS_STATUS_CODE  VLSgetHandleInfo(
LS_HANDLE       lshandle,
VLSclientInfo   *clientInfo);
```

Argument	Description
lshandle	The handle returned by LSRequest or VLSrequestExt
clientInfo (out)	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 397.

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
lshandle	The handle returned by any Request API call.
totalKeysIssued	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
LS_BADHANDLE	Invalid handle. Handle has already been released and destroyed from previous license versions or too many handles have been allocated.
LS_BUFFER_TOO_SMALL	*in_use_ p* parameter is NULL.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

See Also "VLSgetFeatureInfo" on page 96.

Feature Query Functions

The following table summarizes the feature query functions:

Feature Query Functions

Function	Description
VLSgetFeatureInfo	Retrieves feature licensing information from the license server.
VLSgetVersions	Retrieves licensed version information for a feature.
VLSgetFeatureFrom Handle	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;
} VLSfeatureInfo;
```

Member	Description
structSz	Size of VLSfeatureInfo structure.
feature_name	Name of the feature whose information is retrieved. Maximum 24 characters.
version	Feature *version*. Maxmimum 11 characters.

Member	Description
lic_type	Type of license either trial or normal.
trial_days_count	Number of trial days.
birth_day	Day of the license start date.
death_day	The time when the feature expires. The constant, VLS_NO_EXPIRATION, is returned if the license does not have any expiration date.
num_licenses	The total number of licenses the license server is authorized to issue.
total_resv	Number of licenses reserved using group reservations.
lic_from_resv	Number of reserved licenses issued to clients.
lic_from_free_pool	Number of unreserved licenses issued to clients.
qlic_from_free_pool	Number of unreserved licenses issued to queued clients.
is_node_locked	Depending on the locking scheme of the feature, this returns one of the following constants: • VLS_NODE_LOCKED (client locked to license server) • VLS_CLIENT_NODE_LOCKED (client locked) • VLS_FLOATING (license server locked) • VLS_DEMO_MODE (unlocked)
concurrency	Unused.
sharing_crit	Returns the license sharing criteria, which can be one of the following constants: • VLS_NO_SHARING • VLS_USER_NAME_ID • VLS_CLIENT_HOST_NAME_ID • VLS_X_DISPLAY_NAME_ID • VLS_VENDOR_SHARED_ID

Member	Description
locking_crit	The license server locking criteria, which can be one of the following constants: • 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
holding_crit	The license holding criteria, which can be one of the following constants: • 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).
hold_time	The hold time specified for licenses issued for this feature.
num_subnets	The number of subnet specifications provided for the site.
site_license_info	A space-separated list of subnet wildcard specifications.
meter_value	Unused.
vendor_info	The vendor-defined information string. The maximum length of vendor_info string can be 395 characters.
cl_lock_info	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, *cl_lock_info* could be: 0x8ef38b91(20#) 0xa4c7188d 0x51f8c94a(10#).
key_life_time	The license lifetime for this feature (in seconds).

Member	Description
sharing_limit	The limit on how many copies of the licensed application can share the same license.
soft_num_licenses	The soft limit (for alerts) on the number of concurrent users of this feature.
is_standalone	Returns VLS_TRUE if this is a stand-alone license or VLS_FALSE if this is a network license.
check_time_tamper	Returns VLS_TRUE if this feature is time tamper proof or VLS_FALSE if not time tamper proof.
is_additive	Returns VLS_TRUE if this is an additive license or VLS_FALSE if this is an exclusive license.
isRedundant	Validates if the license is actually redundant.
majority_rule	Checks whether majority rule is on or off.
num_servers	Number of redundant license servers.
isCommuter	Commuter licenses.
log_encrypt_level	Encryption level in the network license for the license server's usage log file.
avg_queue_time	Average time the past or present clients are in the queue. (Not implemented.)
queue_length	Length of the queue.
tot_lic_reqd	Required number of licenses for all queued clients.
commuted_keys	Number of commuter keys that have been checked out.
commuter_keys_left	Number of computer keys left.
server_locking_info	Stores the server locking information.
capacity_flag	The capacity flag can be one of the following constants: • VLScg_CAPACITY_NONE - for no capacity • VLScg_CAPACITY_NON_POOLED - for non-pooled capacity • VLScg_CAPACITY_POOLED - for pooled capacity
capacity	Total capacity of the license.
total_resv_capacity	Total reserved capacity.

Member	Description
in_use_capacity_ from_reserved	Capacity given to clients from reserved capacity.
in_use_capacity_ from_unreserved	Capacity given to clients from unreserved capacity.

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
name	Name of the feature.
version	*Version* of the feature.
index	Used to specify a particular client.
unused1	Use NULL as the value.
featureInfo (out)	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 397.

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	• featureInfo is NULL • *index* is negative • Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
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 *hostName* was specified.

VLSgetFeatureInfo 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 397.

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
featureName	Name of the feature.
bufferSize	Specifies the size of *versionList*.
versionList (out)	An array containing the version list. Space should be allocated by the caller.
unused1	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	*featureName* 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 *hostName* 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 397.

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
handle	*Handle* returned by license request API.
buffer (out)	*Buffer* to hold the feature name. Space allocated by caller.
bufferSize	Size of the *buffer*.

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	• *buffer* parameter is NULL. • Size of feature information exceeds *bufferSize* parameter.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
handle	*Handle* returned by LSRequest or VLSrequestExt.
buffer (OUT)	*Buffer* to hold the feature version. Space allocated by caller.
bufferSize	Size of the *buffer*.

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	• *buffer* parameter is NULL. • Size of feature information exceeds *bufferSize* parameter.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

VLSgetTimeDriftFromHandle

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

```
LS_STATUS_CODE VLSgetTimeDriftFromHandle(
LS_HANDLE      lshandle,
long           *secondsServerAheadOfClient (*out*));
```

Argument	Description
lshandle	Handle returned by LSRequest, VLSrequestExt, or VLSqueuedRequest.
secondsServerAheadOf Client	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	*secondsServerAheadOfClient* parameter is NULL.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 397.

VLSgetFeatureTimeLeftFromHandle

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

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

Argument	Description
lshandle	Handle returned by LSRequest or VLSrequestExt.
secondsUntilTheFeature Expires	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	*secondsUntilTheFeatureExpires* 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 *hostName* 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 397.

VLSgetKeyTimeLeftFromHandle

Client	Server	Static Library	DLL
✔		✔	✔

Syntax

```
LS_STATUS_CODE VLSgetKeyTimeLeftFromHandle(
LS_HANDLE       lshandle,
unsigned long   *secondsUntilTheKeyExpires);
```

Argument	Description
lshandle	Handle returned by LSRequest or VLSrequestExt.
secondsUntilTheFeature Expires	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	*secondsUntilTheKeyExpires* parameter is NULL.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
feature_name	Name of the feature.
version	*Version* of the feature.
reserved1	Use any value.
server_list_len	Specifies the size of *server_list*.
server_list (OUT)	Space separated list of license server names.
optionFlag	A three bit flag which guides the behavior of VLSdiscover in finding the license servers. Details are discussed later.
query_list	A colon separated list of *hostNames* 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 397.

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 66.

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
licenseString	String containing licensing information.
unused1	Use NULL as the value.
unused2	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 shutdown 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	*licenseString* 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 *hostName* 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 397.

See Also

"VLSdeleteFeature" on page 114.

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
licenseString	String containing licensing information.
unused1	Use NULL as the value.
unused2	Use NULL as the value.
unused3	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	*licenseString* 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 *hostName* 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 397.

See Also "VLSdeleteFeature" on page 114.

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
featureName	Name of the feature.
version	*Version* of the feature.
unused2	Unused.
unused3	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	• *featureName* is NULL • *version* 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 *hostName* 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 397.

See Also "VLSaddFeature" on page 110.

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
ulInfoCode	Unused.
szVersion	The version of the Sentinel LM client library.
szProtocol	The communication protocol being used for application/ license server communication.
szPlatform	Platform of the client application.
szUnused1	Unused.
szUnused2	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	*pInfo* is NULL.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

VLSshutDown

Client	Server	Static Library	DLL
✔		✔	✔

Shuts down license server at specified *hostname*.

Syntax

```
LS_STATUS_CODE VLSshutDown(
char           *hostname);
```

Argument	Description
hostname	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 *UserName*	—	Same *UserName* or *UserId*	—
	Win 95/98 (non-Admin)	Same *UserName* or *SameHost*	—	Same *UserName*	—
	WinNT (non-Admin)	Same *UserName*	—	Same *UserName*	—
	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	*hostName* 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 *hostName* 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 397.

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
feature_name	Name of the feature.
version	*Version* of the feature.
unused1	Use any value.
bufferSize	Specifies the size of the buffer.
server_names (OUT)	Space separated list of license server names.
broadcastFlag	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 397.

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
feature_name	Name of the feature.
version	Version of the feature. Must be unique.
trialperiod (OUT)	Number of seconds left in the trial license. Points to integer in the *trialperiod* parameter.
unused1	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	• feature_name is NULL. • version is NULL • trialperiod 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 and Result Codes for License Generation Functions,” on page 415.

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

Argument	Description
unused1	Reserved. Uses NULL as the value.
unused2	Reserved. Uses NULL as the value.
tmtmpr_info	Contains the time tampering related information on the server machine.
machine_id	Machine ID structure. To be used in conjunction with lock_mask to obtain the locking code of the server machine. See VLSmachineIDtoLockCode 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 informationusing 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
server_name	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.
srv_info	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.
unused1	Reserved. Uses NULL as the value.
unused2	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 397.

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;
```

Error Handling

The following table summarizes the error-handling functions:

Error-Handling Functions

Function	Description
VLSerrorHandle	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 397. 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
flag	To turn on error handling, use VLS_ON. To turn off error handling, use VLS_OFF. Default: VLS_ON.

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

Returns The status code LS_SUCCESS is always returned. For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 397.

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
lshandle	Handle returned by LSRequest or VLSrequestExt.
value	Error code.
buffer (out)	*Buffer* to store message.
bufferSize	Size of the *buffer*.

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	*buffer* is NULL *bufferSize* is zero or negative.

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 397.

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
errorCode	The error code to be handled.
featureName	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	• *myErrorHandler* parameter is NULL • LSErrorType is an invalid error type.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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 *stderr.*

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

Returns

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

VLSsetUserErrorFile Error Codes

Code	Description
VLS_CALLING_ERROR	Could not open *msgFile*.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
traceLevel	The default value of *traceLevel* is VLS_NO_TRACE. Other valid values are: • 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 149.
- 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

Example:

```
*********************************************************
/*  Copyright (C) 2004 Rainbow Technologies, Inc. */
/*                All Rights Reserved             */
/*                                                */
/*This Module contains Proprietary Information of */
/*Rainbow Technologies,Inc and should be treated as*/
/* Confidential                                    */
*************************************************

#include <stdio.h> /* For scanf(), sprintf() etc.*/
#include "lscgen.h" /* For the code generator API.*/
/* The fixed feature name of licenses generated by this
example * program. */
#define VLS_CGENXMPL_FEATURE_NAME  "CGENXMPL"
/*Mnemonic used for setting code structure for long codes.*/
#define VLS_LONG_CODE_TYPE_STR  "1"

/*
* Utility function to print code generator API errors to
* stderr.
* It also calls the code generator library cleanup function on
* the handle if necessary.
*/
static int VLSPrintErrors (VLScg_HANDLE *iHandle, int retCode)

{
  if (*iHandle != VLScg_INVALID_HANDLE) {
     (void) VLScgPrintError(*iHandle, stderr);
     (void) VLScgCleanup(iHandle);
}
return retCode;
} /* VLSPrintErrors() */
/*
* A simple example to illustrate the use of the code
* generation API to generate license strings.
* This is a command line utility that generates license
* codes for a fixed feature name,  "CGENXMPL".
* It prompts the user for the expiration date and then calls
* the code generator API functions to generate an
* appropriate license for CGENXMPL.
* To build this example, compile and then link with the
```

```
* appropriate code generator API library - 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;/* VLScg_SHORT_CODE/VLScg_LONG_CODE/
              VLScg_SHORT_NUMERIC_CODE   */

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];
int num_nl_clients;
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_CODE_COMP_LEN+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 log_encrypt_level;
int elan_key_flag;

/* Fields for internal use, or unused */
int vendor_code;
int version_num;
int licensing_crit;
/*Fields for multi_key for short numeric codegen version >=2 */
int num_features;
```

```
int key_type;

/* Fields for capacity Licensing */
int capacity_flag;
int capacity_units;
unsigned long capacity;
} codeT;
```

Member	Description
code_type	Pointer to CodeT struct
additive	License type can be additive or exclusive.
client_server_lock_mode	Locking mode can be: • 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
holding_crit	Criterion for held licenses can be: • VLScg_HOLD_NONE • VLScg_HOLD_VENDOR • VLScg_HOLD_CODE
sharing_crit	Criterion for sharing of non-capacity licenses, can be: • VLScg_NO_SHARING • VLScg_USER_SHARING • VLScg_HOSTNAME_SHARING • VLScg_XDISPLAY_SHARING • VLScg_VENDOR_SHARING Criterion for sharing of capacity licenses, can be: • VLScg_NO_TEAM • VLScg_USER_BASED_TEAM • VLScg_HOSTNAME_BASED_TEAM • VLScg_XDISPLAY_BASED_TEAM • VLScg_VENDOR_BASED_TEAM
server_locking_crit1	• Server lock selector/criterion (group 1) • Allows 2 hostid's per server

Member	Description
server_locking_ crit2	• Server lock selector/criterion (group 2) • Allow 2 hostid's per server
client_locking_ crit	• Client lock selector/criterion • Allow 1 hostid per client
standalone_flag	Specifies if the license is stand-alone or network.
out_lic_type	Specifies if the license is • Encrypted • Expanded readable • Concise readable
clock_tamper_ flag	If set then the license does not allow time tampering
feature_name	Name of the feature
feature_version	Version of the feature
birth_day	day of the month (1-31)
birth_month	1 - 12 or JAN - DEC
birth_year	2003 to...; minimum birth year can be 2003.
death_day	max day of the month (1-31)
death_month	1 - 12 or JAN - DEC
death_year	2003 to... ; minimum death year can be 2003.
num_servers	Identifies the number of license servers. 1serverallowed for single server application and maximum 11 servers allowed for redundant server application.
server_lock_info1	Stores information in ascii
server_lock_info2	Stores information in ascii
num_nl_clients	Number of nodelocked clients, maximum of 7 nodelocked clients allowed.
nl_client_lock_ info	Stores information in ascii
num_keys	Number of concurrent keys

Member	Description
soft_limit	0 to num_keys
keys_per_node	Number of keys alloted to each client for a network mode license
num_subnets	The number of subnet specifications provided for the site.
site_lic_info	Stores information in binary
share_limit/ team-limit	• Number of clients/users who can share a single license key . • Used as team limit in case of capacity license.
key_life_units	Determines lifetime least count
long key_lifetime	Absolute value in minutes
key_hold_units	Flag which determines heldtime least count
key_holdtime	Absolute value in minutes
num_secrets	Number of Challenge response secrets
secrets	Stores information in ascii
vendor_info	The vendor-defined information string. The maximum length of vendor_info string can be 395 characters.
licType	Trial or Normal license type
trialDaysCount	Life of trial license.
use_auth_code	For multi-keys or short numeric codes
numeric_type	For short numeric codes • 0 - non-numeric • 1 - general short numeric • 2 - general numeric • 10 and above specific type for codegen_version=7
isRedundant	Validates if the license is actually redundant.
majority_rule	Checks whether majority rule is on or off.
isCommuter	Commuter licenses.
log_encrypt_level	For encryption level in the license code.

Member	Description
vendor_code	Vendor identification code
version_num	Version number
meter_value	Fields for multi_key for short numeric codegen version >=2
num_features	Number of features in case of multi key
key_type	Single key/Multi key for short numeric only
capacity_flag	Specifies if the license is a capacity or non-capacity license. Values can be: • VLScg_CAPACITY_NONE • VLScg_CAPACITY_NON_POOLED • VLScg_CAPACITY_POOLED
capacity_units	Flag which determines capacity least count
long capacity	The capacity of this license.

Basic Functions

The following table summarizes the basic functions for this library:

Basic Functions

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

VLScgInitialize

Syntax

```
int VLScgInitialize(
VLScg_HANDLE *iHandleP);
```

Argument	Description
iHandleP	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 and Result Codes for License Generation Functions," on page 415.

See Also "VLScgCleanup" on page 144

VLScgCleanup

Syntax

```
int VLScgCleanup(
VLScg_HANDLE *iHandleP);
```

Argument	Description
iHandleP	The pointer to the instance handle for this library.

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

Returns The status code VLScg_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgReset

Syntax

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

Argument	Description
iHandleP	The instance handle for this library.
codeP	Name of the structure.

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

Returns The status code VLScg_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

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
iHandleP	The pointer to the instance handle for this library.
numMsgsP (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 and Result Codes for License Generation Functions,” on page 415.

VLScgGetErrorLength

Syntax

```
int VLScgGetErrorLength(
VLScg_HANDLE  iHandle,
int           msgNum,
```

```
int          errLenP);
```

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgGetErrorMessage

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
msgBuf (OUT)	A user allocated buffer into which the reference message will be copied.
bufLen	The byte length of the message copied into *msgBuf*.

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 and Result Codes for License Generation Functions," on page 415.

VLScgPrintError

Syntax

```
int VLScgPrintError(
VLScg_HANDLE   iHandle,
FILE           *file);
```

Argument	Description
iHandle	The instance handle for this library.
file	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 and Result Codes for License Generation Functions," on page 415.

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 VLScgSetFeatureName	Sets the name of the feature to be licensed.
VLScgAllowFeatureVersion VLScgSetFeatureVersion	Sets the version number to be licensed.
VLScgAllowLicenseType VLScgSetLicenseType	Controls the license type.
VLScgAllowTrialLicFeature VLScgSetTrialDaysCount	Sets the number of trial days.
VLScgAllowAdditive VLScgSetAdditive	Sets the license to exclusion or additive.
VLScgAllowKeyLifeUnits VLScgSetKeyLifetimeUnits	Sets unit of time used to specify time between license renewals.
VLScgAllowStandAloneFlag VLScgAllowNetworkFlag VLScgSetStandAloneFlag	Sets whether license will be for stand-alone or network computer.
VLScgAllowLogEncryptLevel VLScgSetLogEncryptLevel	Controls the network license encryption level for the license server's usage log file.

Functions of the CodeT Struct (Continued)

Function	Description
VLScgAllowSharedLic/ VLScgAllowTeamCriteria VLScgSetSharedLicType/ VLScgSetTeamCriteria	• Enables shared licenses and sets sharing criteria for non-capacity license. • Enables team licenses and sets team criteria for capacity license.
VLScgAllowShareLimit/ VLScgAllowTeamLimit VLScgSetShareLimit/ VLScgSetTeamLimit	• Sets the number of users that can share a non-capacity license. • Sets the number of team members that can share a token in case of capacity license.
VLScgAllowCommuterLicense VLScgSetCommuterLicense	Enables commuter licenses to be checked out.
VLScgAllowNumKeys VLScgSetNumKeys	Sets the number of concurrent licenses allowed.
VLScgAllowLockModeQuery VLScgSetClientServerLockMode	Sets locking mode for the license server computer. Installs client server lock mode in codeP.
VLScgAllowRedundantFlag VLScgSetRedundantFlag	Controls whether the license will be used with redundant license servers.
VLScgAllowMajorityRuleFlag VLScgSetMajorityRuleFlag	Controls whether the majority of redundant license servers must be running.
VLScgAllowMultipleServerInfo VLScgSetNumServers	Fields for information on various license servers.
VLScgAllowServerLockInfo VLScgSetServerLockInfo1	Sets license server primary locking code. Installs license server lock code in primary lock.
VLScgSetServerLock Mechanism1	Sets license server primary fingerprint criteria. Installs license server's fingerprint criteria in primary lock.
VLScgSetServerLock Mechanism2	Sets license server secondary fingerprint criteria. Installs license server's fingerprint criteria in secondary lock.

Functions of the CodeT Struct (Continued)

Function	Description
VLScgSetServerLockInfo2	Sets license server secondary locking code. Installs server lock code in secondary lock.
VLScgAllowLockMechanism VLScgSetClientLockMechanism	Sets client's fingerprint criteria.
VLScgAllowClientLockInfo VLScgSetClientLockInfo	Sets the client locking code.
VLScgSetNumClients	Sets the number of client locking codes to be specified.
VLScgAllowClockTamperFlag VLScgSetClockTamperFlag	Controls action on detection of clock being set back on the machine.
VLScgAllowOutLicType VLScgSetOutLicType	Sets the license output format.
VLScgSetLicType	Sets the license type.
VLScgAllowHeldLic VLScgSetHoldingCrit	Enables/disables license hold time and determines where that hold time is specified.
VLScgAllowCodegenVersion VLScgSetCodegenVersion	Sets the version of license codes to generate. Checks if the current license code setting allows multiple features.
VLScgAllowMultiKey VLScgSetKeyType	Controls whether a license will be single or multi-feature.
VLScgAllowSecrets VLScgSetSecrets VLScgSetNumSecrets	Sets the value of the specified challenge-response secrets. Sets the total number of secrets for the challenge-response.
VLScgAllowVendorInfo VLScgSetVendorInfo	Sets vendor-defined information in the license.
VLScgAllowKeysPerNode VLScgSetKeysPerNode	Sets the number of license tokens per node for the specified number of clients.

Functions of the CodeT Struct (Continued)

Function	Description
VLScgAllowSiteLic VLScgSetSiteLicInfo VLScgSetNumSubnets	Sets address of subnets licensed application will be restricted to. Sets the number of subnets the licensed application is restricted to.
VLScgAllowNumFeatures VLScgSetNumFeatures	Sets the number of features.
VLScgAllowSoftLimit VLScgSetSoftLimit	Sets soft limit number.
VLScgAllowKeyHoldUnits VLScgSetKeyHoldtimeUnits	Sets units of time to be used to specify license hold time.
VLScgAllowKeyLifetime VLScgSetKeyLifetime	Sets time between license renewals.
VLScgAllowKeyHoldtime VLScgSetKeyHoldtime	Sets the time a license will be held.
VLScgAllowLicBirth VLScgSetLicBirthMonth VLScgSetLicBirthDay 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. Sets the year of the license start date.
VLScgAllowLicExpiration VLScgSetLicExpirationMonth VLScgSetLicExpirationDay VLScgSetLicExpirationYear	Sets month license expires.The months should be specified in the range of 0-11. Sets day month the license expires. Sets the year the license expires.
VLScgSetNumericType	Sets the value of numeric type.
VLScgSetLoadSWLicFile	Sets and loads the software license file (*lscgen.lic*).

VLScgSetCodeLength

Syntax

```
int VLScgSetCodeLength(
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 *code_type* member of *codeT* struct. Legal values are: • VLScg_SHORT_CODE_STRING = "0" • VLScg_LONG_CODE_STRING = "1" • VLScg_SHORT_NUMERIC_CODE = "2"

Description

Sets the license code length to short or long.

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

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

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

VLScgSetCodeLength Error Codes

Error Code	Description
VLScg_INVALID_INPUT	If either *codeP* or *flag* are NULL.
VLScg_INVALID_INT_TYPE	Value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_SHORT_CODE_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_LONG_CODE_STRING.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowFeatureName

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetFeatureName

Syntax

```
int VLScgSetFeatureName(
VLScg_HANDLE   iHandle,
codeT          *codeP,
char           *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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 and Result Codes for License Generation Functions,” on page 415.

VLScgAllowFeatureVersion

Syntax

```
int VLScgAllowFeatureVersion
VLScg_HANDLE  iHandle,
codeT         *codeP);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetFeatureVersion

Syntax

```
int VLScgSetFeatureVersion(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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
VLScg_RESERV_STR_ERROR	If the string is a reserved string.
VLScg_INVALID_CHARS	If the string characters are not printable.
VLScg_EXCEEDS_MAX_VALUE	If 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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowLicenseType

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetLicenseType

Syntax

```
int VLScgSetLicenseType(
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* is used to set the *code_type* member of *codeT* struct. The values are: • VLScg_NORMAL_LIC_STRING - Non-trial license = "0" • VLScg_TRIAL_LIC_STRING - Trial license = "1"

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

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

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 VLScg_NORMAL_LIC_STRING

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowTrialLicFeature

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	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.

VLScgSetTrialDaysCount

Syntax

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

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

Description Sets the number of trial days to the count specified by the *daysStr* parameter.

The count string defines a window of time during which the application can run after the first time the license is requested.

Returns The status code VLScg_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowAdditive

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetAdditive

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	The value of *flag* indicates whether the license to be generated is additive/exclusive. The legal values are: • VLScg_ADDITIVE = "0" • VLScg_EXCLUSIVE = "1"

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowKeyLifetime

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetKeyLifetime

Syntax

```
int VLScgSetKeyLifetime(
VLScg_HANDLE   iHandle,
codeT          *codeP,
char           *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Absolute value in minutes of license lifetime. Maximum depends on lifetime units set by VLScgSetKeyLifetimeUnits. See "VLScgSetKeyLifetimeUnits" on page 213.

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
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 3.
VLScg_LESS_THAN_MIN_VALUE	If value is less than or equal to 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowStandAloneFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgAllowNetworkFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

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: • VLScg_NETWORK_STRING = “0” • VLScg_STANDALONE_STRING = “1”

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowLogEncryptLevel

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetLogEncryptLevel

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Allowed value are: • "0" • "1" • "2" • "3" • "4"

Description Controls the encryption level to the network licenses for the license server's usage log file.

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

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 and Result Codes for License Generation Functions," on page 415.

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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	This *flag* enables shared licenses and specifies the sharing criteria. Legal values are: • VLScg_NO_SHARING_STRING = "0" • VLScg_USER_SHARING_STRING = "1" • VLScg_HOSTNAME_SHARING_STRING = "2" • VLScg_XDISPLAY_SHARING_STRING = "3" • VLScg_VENDOR_SHARING_STRING = "4" - Vendor defined / customized. Need to customize the client library for this.

Description

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

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

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 VLScg_VENDOR_SHARING_STRING.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than VLScg_NO_SHARING_STRING.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.

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

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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
decimalNUM	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. *NOLIMITSTR* 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
VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowCommuterLicense

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetCommuterLicense

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Valid values are: • VLScg_NOT_ISSUE_COMMUTER_CODES_STRING = "0" • VLScg_ISSUE_COMMUTER_LICENSE_CODE_STRING = "1"

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 VLScg_ISSUE_COMMUTER_CODES_STRING
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_NOT_ISSUE_COMMUTER_CODES_STRING.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowNumKeys

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetNumKeys

Syntax

```
int VLScgSetNumKeys(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info,
int           num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Sets the number of concurrent licenses: should be from 0 to *NOLIMITSTR* for no limit.
num	Should be 0 in case of single feature and from 0 to "*no_of_features* -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
VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If value of *info* exceeds maximum number of license tokens allowed. Maximum value for long codes is 32766 and maximum value for short codes is 254.
VLScg_LESS_THAN_MIN_VALUE	If value of *info* is less than 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowLockModeQuery

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetClientServerLockMode

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	The *flag* values are: • VLScg_FLOATING_STRING - License server is locked = "0" • VLScg_BOTH_NODE_LOCKED_STRING - Clients and license server are locked = "1" • VLScg_DEMO_MODE_STRING - Trial license (no locking) = "2" • VLScg_CLIENT_NODE_LOCKED_STRING - Only clients are locked = "3"

Description Sets whether license server is locked, clients and license server are both locked, only clients are locked, or neither license server nor clients are locked. Validates the value of *flag* and installs it in the license code structure.

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowRedundantFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Valid values are: • VLScg_NON_REDUNDANT_CODE_STRING - Non-redundant license = “0” • VLScg_REDUNDANT_CODE_STRING - Redundant license = “1”

Description Controls whether the license will be used with redundant license servers.

Redundancy allows the total number of licenses to remain available to the enterprise even if one or more license servers fail. License balancing allows the developer’s end user to set up an initial distribution of license tokens among different sites. The 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
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_REDUNDANT_CODE_STRING.
VLScg_LESS_THAN_MIN_ VALUE	If value is less than VLScg_NON_REDUNDANT_CODE_STRING.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowMajorityRuleFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetMajorityRuleFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Valid values are: • VLScg_MAJORITY_RULE_FOLLOWS_STRING - Sets the *majority_rule_flag* = "1" • VLScg_MAJORITY_RULE_NOT_FOLLOWS_STRING - Unsets the *majority_rule_flag* = "0"

Description Controls whether the majority of redundant license servers must be running.

If the number of redundant license servers running is less than half of the number of license servers specified in the license file, then all servers will stop servicing all old and new clients. For example, if 7 redundant license servers are specified, at least 4 of them must be running to satisfy the majority rule.

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowMultipleServerInfo

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetNumServers

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
str	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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowServerLockInfo

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
lockCode	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."
num	Position in *server_lock_info1* where *lockCode* is stored starting from *0* to *num_servers-1* where num_servers is set using VLScgSetNumServers. • *server_lock_info1* is an array of 11 elements storing the primary locking codes for 11 servers.

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
VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds the maximum number of license servers. The value set using the API VLScgSetNumServers
VLScg_LESS_THAN_MIN_VALUE	If the value is less than minimum number of license servers.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetServerLockMechanism1

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
criterion	The lock code to install. Value should be in hex format.
server	Position in array which is storing the primary locking criterias 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 *number of server* is too large. The value set using the API VLScgSetNumServers.
VLScg_LESS_THAN_MIN_VALUE	If the *number of server* is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetServerLockMechanism2

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
criterion	The lock code to install (in hex).
server	Position in array which is storing the secondary locking criterias of the 11 servers. Its value should also vary from 0 - *num_servers* where *num_servers* 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 *number of server* is too large. The value set using the API VLScgSetNumServers.
VLScg_LESS_THAN_MIN_VALUE	If the *number of server* is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetServerLockInfo2

Syntax

```
int VLScgSetServerLockInfo2(
VLScg_HANDLE   iHandle,
codeT          *codeP,
char           *lockCode,
int            num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
lockCode	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."
num	Position in *server_lock_info2* where *lockCode* is stored starting from *0* to *num_servers-1* where num_servers is set using VLScgSetNumServers. • *server_lock_info2* is an array of 11 elements storing the secondary locking codes for 11 servers.

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
VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If value is too large. The value set using the API VLScgSetNumServers.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowLockMechanism

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetClientLockMechanism

Syntax

```
int VLScgSetClientLockMechanism(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *criterion,
```

```
int          client_num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
criterion	Mask defining which fields of *machineID* are to be used for locking. Value should be in hex format.
client_num	*client_num* is the position in the array storing the locking mechanisms for the clients. The value will vary from 0 to num_nl_clients where num_nl_clients is the number of clients set using VLScgSetNumClients.

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
VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If value is too large. The value set using the API VLScgSetNumClients.
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowClientLockInfo

Syntax

```
int VLScgAllowClientLockInfo(
VLScg_HANDLE   iHandle,
```

```
codeT          *codeP);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetClientLockInfo

Syntax

```
int VLScgSetClientLockInfo(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *lockCode,
int           num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
lockCode	This buffer is used to set the lock code information for clients.
num	Number of clients: should be from 0 to maximum number of clients specified -1. *num* is the position in the array storing the locking codes for the clients. The value will vary from 0 to num_nl_clients where num_nl_clients 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
VLScg_INVALID_HEX_TYPE	If value is not in hexadecimal format.
VLScg_EXCEEDS_MAX_VALUE	If number is greater than *num_nl_clients* - 1. Number of node locked clients.
VLScg_LESS_THAN_MIN_VALUE	If number is less than 0.
VLScg_INVALID_IP_TYPE	If value is not in dot format.
VLScg_UNKNOWN_LOCK	If the locking criteria is unknown.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetNumClients

Syntax

```
int VLScgSetNumClients(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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
VLScg_INVALID_INT_TYPE	If input is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum number of 7 clients.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowClockTamperFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetClockTamperFlag

Syntax

```
int VLScgSetClockTamperFlag(
VLScg_HANDLE   iHandle,
codeT          *codeP,
```

```
char            *flag);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Valid values are: • VLScg_NO_CHECK_TAMPER_STRING - Do not check clock tamper = "0" • VLScg_CHECK_TAMPER_STRING - Check clock tamper = "1"

Description Controls action on detection of clock being set back on the machine.

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

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowOutLicType

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetOutLicType

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Valid values are: • VLScg_ENCRYPTED_STRING = “0” • VLScg_EXPANDED_READABLE_STRING = “1” • VLScg_CONCISE_READABLE_STRING = “2”

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

The license code contains all of the information that defines the license agreement between you and your customer: how many users can run the application at a time, whether the license will expire after a specific number of days, whether the application can only run on a specific computer, and so

on. Encrypted license strings contain this information about the license agreement, but cannot be read by your customers.

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

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

Returns

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

VLScgSetOutLicType 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 and Result Codes for License Generation Functions," on page 415.

VLScgSetLicType

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
lictype	Set the type of license. • VLScg_TRIAL_LIC_STRING = "1" • VLScg_NORMAL_LIC_STRING = "0"

Description Sets the type of license to either trial or normal.

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

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowHeldLic

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	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.

VLScgSetHoldingCrit

Syntax

```
int VLScgSetHoldingCrit(
VLScg_HANDLE  iHandle,
codeT          *codeP,
```

```
char          *flag);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	The *flag* is used to set the criteria for held licenses. Values are: • VLScg_HOLD_NONE_STRING = "0" - Held licenses not allowed. • VLScg_HOLD_VENDOR_STRING = "1" - Client API specifies hold time. • VLScg_HOLD_CODE_STRING = "2" - License code specifies hold time.

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowCodegenVersion

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Sets the possible values for *version_num flag*.

Description Sets the version of license codes to generate. Checks if the current license code setting allow multiple features.

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

VLScgSetCodegenVersion Error Codes

Error Code	Description
VLScg_INVALID_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds MAX_CODEGEN_VERSION.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowCapacityLic

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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 and Result Codes for License Generation Functions," on page 415.

VLScgSetCapacityFlag

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
Flag	The value of flag is used to set the capacity_flag of codeT struct. Legal values are- • VLScg_CAPACITY_NONE_STRING • VLScg_CAPACITY_NON_POOLED_STRING • VLScg_CAPACITY_POOLED_STRING

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
VLScg_SUCCESS	Success
VLScg_INVALID_INT_TYPE	If value is not numeric
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_CAPACITY_POOLED
VLScg_LESS_THAN_MIN_ VALUE	If value is lower than VLScg_CAPACITY_NONE

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowCapacity

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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 and Result Codes for License Generation Functions," on page 415.

VLScgSetCapacityUnits

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
units	Capacity specification units from 0 to 4. The values are: • If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022. • If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220. • If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200. • If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000. • If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000.

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 VLScg_CAPACITY_UNITS_MAX_VALUE
VLScg_LESS_THAN_MIN_VALUE	If value is less than VLScg_CAPACITY_UNITS_MIN_VALUE

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

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 • If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022. • If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220. • If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200. • If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000. • If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000. **NOLIMITSTR or EMPTY("/0") String can be used to specify infinite capacity.**

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowMultiKey

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	*Flag* used to set the *code_type* member of *codeT* struct. The values are: • VLScg_SINGLE_KEY_STRING = "0" • VLScg_MULTI_KEY_STRING = "1"

Description Controls whether a license will be single or multi-feature license code types.

Single Feature: Predefined short, numeric license codes where the license code is for a single feature. Notice, if you select Predefined-Single Feature, the feature name must be no more than 2 numeric digits. Most of the attributes are already defined for you and cannot be modified.

Multi Feature: Predefined short, numeric license types where multiple features (value between 2 - 11) can be placed into a single license code. Notice, if you select Predefined-Multi Feature, the feature name must be no more than 2 numeric digits. Most of the attributes are already defined for you and cannot be modified.

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

VLScgSetKeyType Error Codes

Error Code	Description
VLScg_INVALID_INT_TYPE	If value is not a decimal number.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MULTI_KEY_STRING.
VLScg_LESS_THAN_MIN_ VALUE	If value is lower than VLScg_SINGLE_KEY_STRING.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowSecrets

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetSecrets

Syntax

```
int VLScgSetSecrets(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *valu,
int           num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
valu	Any printable ASCII text.
num	Number of secrets: should be from 0 to *num_secrets* -1. *num* is the position in the array storing the secrets. The value varies from 0 to num_secrets-1, where num_secrets 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
VLScg_INVALID_CHARACTERS	If string is not valid.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum. The value set by VLScgSetNumSecrets
VLScg_LESS_THAN_MIN_VALUE	If the value is lower than minimum.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetNumSecrets

Syntax

```
int VLScgSetNumSecrets(
VLScg_HANDLE iHandle,
codeT        *codeP,
char         *valu);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
valu	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 com-

pute 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
VLScg_INT_TYPE	If value is not numeric.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds VLScg_MAX_NUM_SECRETS.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowVendorInfo

Syntax

```
int VLScgAllowVendorInfo
VLScg_HANDLE  iHandle,
codeT         *codeP);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetVendorInfo

Syntax

```
int VLScgSetVendorInfo(
VLScg_HANDLE  iHandle,
codeT         *codeP,
```

```
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowKeysPerNode

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
keys	Used to set the number of *keys* per node. Give any decimal value. Should be from 0. Give *NOLIMITSTR* for no limit.
num	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
VLScg_INVALID_INT_TYPE	If number is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If number is greater than *num_nl_clients* -1. *num_nl_clients* is a field of *CodeT* structure storing the number of clients.
VLScg_LESS_THAN_MIN_VALUE	If number is less than 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowSiteLic

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetSiteLicInfo

Syntax

```
int VLScgSetSiteLicInfo(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info,
int           num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Set the subnet address. You can use wildcard (e.g., *.123.*.28) to specify a range.
num	*num* is the position of subnet in the array maintaining the subnet info.The value varies from *0* to *num_subnets-1* where num_subnets is an element of *CodeT* struct set using VLScgSetNumSubnets.

Description Sets subnet address. See "VLScgSetNumSecrets" on page 203.

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
VLScg_INVALID_RANGE	If value is not in the range allowed and if value is not a valid character.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetNumSubnets

Syntax

```
int VLScgSetNumSubnets(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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 *num* is greater than *codeP* to *num_subnets*.
VLScg_LESS_THAN_MIN_VALUE	If *num* is less than 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowNumFeatures

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
flag	Sets the *flag* 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 VLScg_MAX_NUM_FEATURES.
VLScg_LESS_THAN_MIN_VALUE	If value is lower than VLScg_MIN_NUM_FEATURES.

For a complete list of the error codes, see Appendix D, “Error and Result Codes for License Generation Functions,” on page 415.

VLScgAllowSoftLimit

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	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.

VLScgSetSoftLimit

Syntax

```
int VLScgSetSoftLimit(
VLScg_HANDLE iHandle,
codeT        *codeP,
char         *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Sets soft limit: should be from 0 to *NOLIMITSTR* for no limit. NOLIMSTR 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
VLScg_INVALID_INT_TYPE	If information is not a non-negative integer.
VLScg_EXCEEDS_MAX_VALUE	If information exceeds maximum number of license tokens allowed. Maximum value for long codes is 32766 and maximum value for short codes is 254.
VLScg_LESS_THAN_MIN_VALUE	If information is less than 0.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowKeyLifeUnits

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetKeyLifetimeUnits

Syntax

```
int VLScgSetKeyLifetimeUnits(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Lifetime specification units of license tokens: from 0 to 3. The values are: • "0" - Multiple of 1 minute(s), maximum 15 minutes. • "1" - Multiple of 10 minute(s), maximum 150 minutes. • "2" - Multiple of 30 minute(s), maximum 450 minutes. • "3" - Multiple of 60 minute(s), maximum 900 minutes.

Description This function specifies the units of time used to specify the time between renewals. A license must be renewed by the application on a regular schedule or the license will be reclaimed. See "VLScgSetKeyLifetime" on page 161.

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowKeyHoldUnits

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Hold time specification units of license tokens: from 0 to 3. The values are: • "0" - Multiple of 1 minute(s), maximum 15 minutes • "1" - Multiple of 10 minute(s), maximum 150 minutes. • "2" - Multiple of 30 minute(s), maximum 450 minutes. • "3" - Multiple of 60 minute(s), maximum 900 minutes.

Description Network licenses may be held for a time when released by a specific user. During that time only the original requestor of the license can be granted the license again. This function sets the units of time used to specify the hold time.

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

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 and Result Codes for License Generation Functions," on page 415.

VLScgAllowKeyHoldtime

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Absolute values in minutes. Maximum depends on units set by VLScgSetKeyHoldtimeUnits. *NOLIMITSTR* 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 and Result Codes for License Generation Functions,” on page 415.

VLScgAllowLicBirth

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetLicBirthMonth

Syntax

```
int VLScgSetLicBirthMonth(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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
VLScg_INVALID_CHARACTERS	If not a valid string.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed month (exceeds 12).
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetLicBirthDay

Sets the day of the license start date.

Syntax

```
int VLScgSetLicBirthDay(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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
VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_DATE	If value is not valid for the month.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed day.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 1.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgSetLicBirthYear

Syntax

```
int VLScgSetLicBirthYear(
VLScg_HANDLE iHandle,
codeT        *codeP,
char         *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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
VLScg_INVALID_INT_TYPE	If value is not a non-negative integer.
VLScg_INVALID_YEAR	If year is invalid.
VLScg_INVALID_BIRTH_YEAR	If year is less than 2003.
VLScg_EXCEEDS_MAX_VALUE	If value exceeds maximum allowed year that is 2130.

For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

VLScgAllowLicExpiration

Syntax

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

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* 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.

VLScgSetLicExpirationMonth

Syntax

```
int VLScgSetLicExpirationMonth(
VLScg_HANDLE   iHandle,
codeT          *codeP,
char           *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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 and Result Codes for License Generation Functions," on page 415.

VLScgSetLicExpirationDay

Syntax

```
int VLScgSetLicExpirationDay(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	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 and Result Codes for License Generation Functions,” on page 415.

VLScgSetLicExpirationYear

Syntax

```
int VLScgSetLicExpirationYear(
VLScg_HANDLE   iHandle,
codeT          *codeP,
char           *info);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
info	Enter year in 4 digits (e.g., 2003) to avoid year 2000 problem. *NEVERSTRING* 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 and Result Codes for License Generation Functions,” on page 415.

VLScgSetNumericType

Syntax

```
int VlScgSetNumericType(
VLScg_HANDLE  iHandle,
codeT         *codeP,
int           num);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
num	Numeric type values are: • VLScg_NUMERIC_UNKNOWN = “0” • VLScg_NOT_NUMERIC = “1” • VLScg_MISC_SHORT_NUMERIC = “2” • VLScg_MISC_NUMERIC = “3”

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
VLScg_EXCEEDS_MAX_VALUE	Value exceeds the maximum value of 3.
VLScg_LESS_THAN_MIN_VALUE	If value is less than 0.
VLScg_INVALID_INT_TYPE	If the value is not a non-negative integer.

For a complete list of the error codes, see Appendix D, “Error and Result Codes for License Generation Functions,” on page 415.

VLScgSetLoadSWLicFile

Syntax

```
int VLScgSetLoadSWLicFile(
VLScg_HANDLE   iHandle,
char           *filename);
```

Argument	Description
iHandle	The instance handle for this library.
filename	Complete name and path of sw license file.

Description Sets and loads the software license file (*lscgen.lic*).

Returns The status code VLScg_SUCCESS is returned if successful. Otherwise, a specific error codes is returned indicating the reason for failure. For a complete list of the error codes, see Appendix D, "Error and Result Codes for License Generation Functions," on page 415.

License Generation Function

The following table summarizes the license generation function:

License Generation Function

Function	Description
VLScgGenerateLicense	Generates the license string.

VLScgGenerateLicense

Syntax

```
int VLScgGenerateLicense(
VLScg_HANDLE  iHandle,
codeT         *codeP,
char          **result);
```

Argument	Description
iHandle	The instance handle for this library.
codeP	The pointer to the *codeT* struct.
result	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 and Result Codes for License Generation Functions,” on page 415.

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
iHandle	The instance handle for this library.
AnyLicenseString	User provided license string to be decoded.
lic_string (OUT)	User allocated buffer to receive decoded license string.
lic_string_length	Length of decoded license string returned.
codeP	Pointing to *codeT* 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 *AnyString*. This function takes care of all memory allocations it uses.

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

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.
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 and Result Codes for License Generation Functions," on page 415.

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
iHandle	The instance handle for this library.
initialUnitsP	The number of units that were initially available.
unitsLeftP	The number of units remaining.
codegen_version	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 and Result Codes for License Generation Functions," on page 415.

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
iHandle	The instance handle for this library.
units	The number of licenses available.
codegen_version	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 and Result Codes for License Generation Functions," on page 415.

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 *hostName*, if possible.
VLSgetHostAddress	Accepts *hostName* 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 *server_name_list* will contain license server names (*hostNames* 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
licenseStr	The *license string* that will be added.
unused1	Should be NULL.
unused2	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	*licenseString* 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 *hostName* 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 397.

VLSaddFeatureExt

Syntax

```
int VLSaddFeatureExt (
unsigned char    *licenseString,
unsigned char    *DistCritString,
unsigned char    *unused1,
LS_CHALLENGE     *unused2);
```

Argument	Description
licenseString	The license string that will be added.
DistCritString	Distribution criteria string. The string will allocate the license to another license server if the main license server is locked.
unused1	Should be NULL.
unused2	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	*licenseString* 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 *hostName* 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 397.

VLSaddFeatureToFile

Syntax

```
int VLSaddFeatureToFile(
unsigned char   *licenseString,
unsigned char   *unused1 ,
unsigned char   *unused2,
unsigned char   *unused3);
```

Argument	Description
licenseString	The *license_string* character.
unused1	Should be NULL.
unused2	Should be NULL.
unused3	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	*licenseString* 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 *hostName* 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 397.

VLSaddServerToPool

Syntax

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

Argument	Description
server_name	Name of the license server to add to the pool.
server_addr	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	• *server_name* is NULL • *server_address* is NULL • Using stand-alone library. This function cannot be used with stand-alone library.
LS_NO_SUCCESS	Generic error indicating that the license server could not be added to the pool.
VLS_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	*hostName* 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 *hostName* 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 397.

VLSchangeDistbCrit

Syntax

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

Argument	Description
feature_name	Name of the feature. Must be unique.
version	*Version* of the feature.
dist_crit	*Dist_crit* consists of the names of license server, which will have licenses for the given *feature_name* and *version*. The *dist_crit* 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
LS_BAD_DIST_CRIT	Change *dist_crit* and allocate some keys to the deleted license server.
LS_NON_REDUNDANT_ SERVER_ CONTACTED	LSHOST is set to a non-redundant license server.
LS_BAD_PARAMETER	License server's name is NULL or an empty string.
LS_NO_AUTHORIZATION	License server does not recognize this feature name.
LS_NO_SUCH_FEATURE	*Feature_version* is non-existent.
LS_UNRESOLVED_SERVER_ NAME	License server's name cannot be resolved.
LS_MSG_TO_LEADER	The request has been sent to the leader license server.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

VLSdelServerFromPool

Syntax

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

Argument	Description
server_name	Name of the license server to delete from the pool.
server_addr	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	• *server_name* is NULL • *server_address* is NULL • Using stand-alone library. This function cannot be used with stand-alone library.
LS_NO_SUCCESS	Generic error indicating that the license server could not be deleted from the pool.
VLS_NON_REDUNDANT_ SRVR	License server is non-redundant and therefore cannot support this redundancy-related operation.
VLS_SERVER_NOT_PRESENT	Attempted to delete a license server that is not in the pool.
VLS_ONLY_SERVER	Cannot remove the last license server from the pool.
VLS_NO_SERVER_RUNNING	License server on specified host is not available for processing the license operation requests.
VLS_BAD_HOSTNAME	*hostName* 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 *hostName* 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 397.

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
feature_name	Name of the feature.
version	*Version* of the feature. Must be unique.
unused1	Should be NULL.
num_servers	Number of license servers for which *discoverInfo* array is allocated.
discoverInfo	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
option_Flag	The option flag is allowed to be logically ORed with other flags. However, this flag will have first priority. Valid flags are: • 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 *LSHOST/lshost.* • VLS_DISC_RET_ON_FIRST – If the combined query list is NULL, it returns the name of the first contacted license server in the *server_list,* 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. • VLS_DISC_PRIORITIZER_LIST – Treat the combined query list as a prioritized one, left most being the highest priority host. It returns in *server_list*, license servers sorted in the order of priority host. If this option is not specified, the combined query list is treated as random. • VLS_DISC_REDUNDANT_ONLY – Expecting reply only from redundant license servers. All non-redundant license servers will ignore the message. • VLS_DISC_DEFAULT_OPTIONS – This flag is a combination of the aforementioned flags. Use it if you are not sure which flag you want to specify.
sharing_crit	The license server will match client's internal information with the keys it is already granted. Values are: • VLScg_NO_SHARING • VLScg_USER_SHARING • VLScg_HOSTNAME_SHARING • VLScg_XDISPLAY_SHARING • VLScg_VENDOR_SHARING
vendor_list	Consists of server names. These license serves will be contacted. The names of all the license servers that have licenses for specified *feature_name* and *version* will be returned in *vendor_list* in the same order as in the original (before the call) *vendor_list*.

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	*num_servers* 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 397.

VLSgetDistbCrit

Syntax

```
int VLSgetDistbCrit (
char     *feature_name,
char     *feature_version,
char     *dist_crit,
int      *distcrit_buflen);
```

Argument	Description
feature_name	Name of the feature.
feature_version	*Version* of the feature. Must be unique.
dist_crit (OUT)	*Dist_crit* consists of the names of license server, which will have licenses for the given *feature_name* and *version*. The *dist_crit* string must be null-terminated.
distcrit_buflen	Size of memory allocated for *dist_crit*.

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	• *feature_name* is NULL • *version* is NULL • *dist_crit* is NULL • *dist_crit_len* is zero or negative • *challenge* argument is non-NULL, but cannot be understood. 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	*dist_crit* 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 *hostName* 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 397.

VLSgetDistbCritToFile

Syntax

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

Argument	Description
feature_name	Name of the feature.
feature_version	*Version* of the feature.
file_name	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	• *feature_name* is NULL • *file_name* is NULL. Using stand-alone library. This function cannot be used with stand-alone library.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature, 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 *hostName* 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	*feature_version* 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 397.

VLSgetFeatureInfoToFile

Syntax

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

Argument	Description
feature_name	Name of the feature.
version	*Version* of the feature. Must be unique.
file_name	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	• *file_name* is NULL • *feature_name* is NULL.
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 *hostName* 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 397.

VLSgetHostName

Syntax

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

Argument	Description
IP_address	IP addresses to be converted to *hostname*.
hostname (OUT)	IP address to be converted to *hostname*.
HostNameBufLen	The length of the message copied into *hostname*.

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	• *IP_address* is NULL • *hostName* is NULL • *hostNameBufLen* is NULL Using stand-alone library. This function cannot be used with stand-alone library.
VLS_INVALID_IP_ADDRESS	*IP_address* is not valid.
VLS_UNRESOLVED_IP_ADDRESS	*IP_address* is valid, but could not be resolved.
LS_BUFFER_TOO_SMALL	Length of *hostName* returned exceeds *hostNameBufLen*.
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 397.

VLSgetLeaderServerName

Syntax

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

Argument	Description
leader_name	Current lead license server's name. Return types: • 0 = Success. Found leader license server name. • 1 = Contact license server is not a redundant license server. • 2 = Other error.

Description Returns the current lead license server's name by contacting any license server. The license server to be contacted is selected by VLSgetServerName call. So a license server's name must be set before a call is made to this function.

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	• *leader_name* is NULL • *leadername_len* is NULL.
LS_BUFFER_TOO_SMALL	*leadername_len* 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 *hostName* 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 397.

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 *IP_AddressBuff.*

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	• *IPaddressBuf* is NULL • *IPAddrBufLen* is NULL. • Using stand-alone library. This function cannot be used with stand-alone library.
VLS_UNRESOLVED_HOSTNAME	• *hostname* is valid, but could not be resolved. • IPX protocol is current.

Note: If you wishe 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 397.

VLSgetLicSharingServerList

Syntax

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

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

Description Returns the license server names which are sharing tokens for a given feature name and version. The *server_name_list* will contain license server names (*hostNames* or IPX addresses).

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

VLSgetLicSharingServerList Error Codes

Error Code	Description
VLS_CALLING_ERROR	• *feature_name* is NULL • *feature_version* is NULL • *server_list* is NULL • *server_list_len* is zero. • License server's name is NULL or an empty string. Both feature name and version cannot be NULL at the same time.

VLSgetLicSharingServerList Error Codes (Continued)

Error Code	Description
LS_BUFFER_TOO_SMALL	*server_list_len* is smaller than license server name that will be returned.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches the requested feature, 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 *hostName* 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 397.

VLSgetPoolServerList

Syntax

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

Argument	Description
**outBuf (OUT)*	Output buffer.
outBufSz	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 397.

VLSsetBorrowingStatus

Syntax

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

Argument	Description
feature_name	Name of the feature.
feature_version	Version of the application. Must be unique.
state	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 397.

VLSsetServerLogState

Syntax

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

Argument	Description
event	The event you want to log. Event may be: • LOG_SRVR_UP (the license server is running) • LOG_LDR_ELECT (a pool leader has been elected) • LOG_HRT_BT (license server bbeat) • LOG_BORROW_REQ_RESP (token borrowing request and response) • LOG_USG_NOTIFY (follower notifies pool leader of token use) • LOG_CHNG_DIST_CRT (token distribution criteria has changed) • LOG_DIST_CRT_SYNC (the pool servers are synchronizing distribution criteria) • LOG_CFG_FILE (the *LSERVRLF* file changed) • LOG_SRVR_DOWN (license server is down) • LOG_MOD_SERVER (addition or deletion of a license server to or from the pool) • LOG_ADD_DEL_LIC (redundant license has been added or deleted to or from the pool)
state	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
VLS_CALLING_ERROR	License server's name is NULL or an empty string.
VLS_DIFF_LIB_VER	Version mismatch between license server API and client API.
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.
VLS_NOT_AUTHORIZED	The user making the request is invalid.
VLS_BAD_SERVER_ MESSAGE	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 397.

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

#include "lserv.h"
Static  LS_Handle  ls_handle;
/* Prototype of timer handler function             */
void TimerHandler ();
int main(argc, argv)
int argc;
```

```
char **argv;
{
 char feature_name [] = "My Application";
 char version_name [] = "1.0";
 LS_STATUS_CODE returnCode = 0;
 int number_of_units_requested = 1;
 VLSqueuePreference   queue_preference;
 int request_flag
if(VLS_INITIALIZE()){ /*Initialize the LS API */return(1);
}

request_flag = VLS_REQ_GET | VLS_REQ_QUEUE;
/* Stay in queue at most 30 minutes   */
queue_preference.wait_time = 1800;

/* Once license available for this client, reserve it
for 5 minutes queue_preference.hold_time = 600;
queue_preference.priority_num = 1;  */ Not used     */

/* Don't queue me if there are 5 or more entries  on the queue*/
queue_preference.absPosition = 5;

/* Don't queue me if there are 2 or more entries from my
reservation group on the queue  */
queue_preference.grpPosition = 2;

/* Request key from 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 VLSqueuedRequestExt	An integrated request for an authorized license code from the license server. Use this API to: • Request a license, with option to queue (*requestFlag* = VLS_REQ_GET | VLS_REQ_QUEUE). • Request a license without queuing (*request-Flag* = VLS_REQ_GET). This option has the same effect as calling a non-queuing API request (LSRequest, VLSrequestExt*, etc.*). • Request to be placed in the queue, even if the license server has available licenses (*requestFlag* = VLS_REQ_QUEUE).
VLSgetQueuedClientInfo	Retrieves the current information of a queued client, such as the number of requested *licenses*, *feature_name*, *version*, and *index*.
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
wait_time	Maximum time, the client can be in queue.
hold_time	After allotment, the maximum time interval for which the server will keep the requested units reserved for this client.

VLSqueuePreference Struct (Continued)

Member	Description
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	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
identifier	The name of the server hook which the user wants to call.
inBuf	String passed to the server.
outBuf	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
user_name	The login name of the user using the application, where MAXLEN is set to 64 characters.
host_name	Name of the host/computer where the user is running the application, where MAXLEN is set to 64 characters.
x_display_name	Name of the X display where the user is displaying the application, where MAXLEN is set to 64 characters.
shared_id_name	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.
group_name	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, *DefaultGrp* is returned.
host_id	The host ID of the computer on which the user is working.

VLSqueuedClientInfo Struct (Continued)

Member	Description
server_start_time	*server_start_time* is the start time of the license token.
server_end_time	*server_end_time* is the end time of the license token. server_end_time should be interpretted as as start_time + 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
license_system	A license requested in the system. Pointer to the string which uniquely identifies a particular license system.
publisher_name	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.
product_name	Feature name. The name of the product requesting licensing resources. Cannot be NULL and must be unique.
version	*Version* for which licenses are requested. Must be unique for the associated feature
units_reqd	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
log_comment	A string that is written by the license manager to the comment field of the usage log file.
challenge	Pointer to a *challenge* structure. The challenge-response will also be returned.
lshandle	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.
qPreference	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.
requestFlag	Valid values are: • VLS_REQ_GET - specifies a non-queuing request (without queuing the client). If license is not available, client will not be queued. • VLS_REQ_QUEUE - specifies to queue the client (without returning with the license). Even if license is available, client will be queued. 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).
server_info	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	• *request_flag* specifies queuing but *qPreference* is NULL. • *lshandle* is NULL. • *challenge* argument is non-NULL, but cannot be understood. • *publisherName* is NULL
VLS_APP_UNNAMED	• *product_name* is NULL • version is NULL
VLS_NO_LICENSE_GIVEN	• *units_reqd* is zero. • Invalid handle specified. • Generic error indicating that the license is not granted.
LS_NOLICENSESAVAILABLE	All licenses in use.
LS_INSUFFICIENTUNITS	License server does not currently have sufficient licensing units for the requested feature to grant a license.
VLS_NO_SUCH_FEATURE	License server does not have a license that matches requested feature, 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 *hostName* 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 397.

VLSgetQueuedClientInfo

Syntax

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

Argument	Description
feature_name	Feature name of the client for which we are requesting information.
version	*Version* for which licenses are requested. Must be unique, for the associated feature.
index	*Index* of the client with the license server, for a particular feature.
client_info	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	• *client_info* parameter is NULL. • *index* is negative. • Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.
VLS_APP_UNNAMED	• *feature_name* is NULL • *version* is NULL 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 *hostName* 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 397.

VLSremoveQueuedClient

Syntax

```
int VLSremoveQueuedClient(
unsigned char  *feature name
unsigned char  *version
int            qkey_id);
```

Argument	Description
feature_name	Feature name of the client for which we are requesting information.
version	*Version* for which licenses are requested.
qkey_id	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	• *qkey_id* parameter cannot be negative. • Attempted to use stand-alone mode with network only library, or network mode with stand-alone library.

VLSremoveQueuedClient Error Codes (Continued)

Error Code	Description
VLS_APP_UNNAMED	• *feature_name* is NULL • *version* is NULL 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 *hostName* 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 397.

VLSremoveQueue

Syntax

```
int VLSremoveQueue(
unsigned char    *feature name
unsigned char    *version);
```

Argument	Description
feature_name	Identifies the license whose queue needs to be removed.
version	*Version* for which licenses are requested. Must be unique.

Description

This API will provide a mechanism to delete the complete queue for a specified license.

VLSremoveQueue will be available to:

- The user-account who started the license server, which actually signifies when the client was put in the queue.
- The root/administrator account.

Returns

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

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	• *feature_name* is NULL • *version* is NULL 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 *hostName* 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 397.

VLSgetHandleStatus

Syntax

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

Argument	Description
lshandle	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	*lshandle* is an ambiguous handle; it is not exclusively active or exclusively queued.
VLS_ACTIVE_HANDLE	*lshandle* is an active handle.
VLS_QUEUED_HANDLE	*lshandle* is a queued handle.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

VLSupdateQueuedClient

Syntax

```
int VLSupdateQueuedClient(
LS_HANDLE       lshandle,
long            *absExpiryTime,
unsigned char   *unused1,
LS_CHALLENGE    *unused2);
```

Argument	Description
lshandle	The handle previously returned by VLSqueuedRequest. The status of the handle must be VLS_QUEUED_HANDLE or an error will occur.
absExpiryTime	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. Possible values for *absExpiryTime* are: • Zero = license is not available. • Non-zero = license is available and will stop calling the API.
unused1 *unused2*	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	• *absExpiryTime* is NULL. • Handle cannot be active. • *challenge* argument is non-NULL, but cannot be understood.
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 397.

VLSgetQueuedLicense

Syntax

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

Argument	Description
lshandle	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.
log_comment	A string that is written by the license manager to the comment field of the usage log file.

Argument	Description
challenge	The challenge-response for this operation. Pointer to a *challenge* 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	*challenge* 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 397.

VLSinitQueuePreference

Syntax

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

Argument	Description
qPreference	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 271.

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

Error Code	Description
VLS_CALLING_ERROR	*qPreference* is NULL.

For a complete list of the error codes, Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
VLSgetCommuterInfo	Returns the commuter license information.
VLSgetAndInstallCommuter-Code	Obtains the commuter code from the license server and issues the commuter authorization to the client side persistence database
VLSuninstallAndReturn CommuterCode	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
commuter_code_version	Version of commuter code
codegen_version	Version of the code generator used
feature_name	Name of the feature
feature_version	Version of the feature
birth_day	Start day (1-31)
birth_month	Start month (1-12)
birth_year	Start year
death_day	End day (1-31)
death_month	End month (1-12)
death_year	End year
num_of_licenses	Number of licenses
locking_crit	Locking criteria of the client
lock_info	Locking information of the client
vendor_info	The vendor-defined information string. Maximum length of this string can be 395 characters.
issuing_server	License checked out from <servername>
key_life_time	The license lifetime for this feature (in seconds).
protocol_type	Type of protocol used
status	• 1 - Active • 0 - Inactive

VLSgetCommuterInfo

Syntax

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

Argument	Description
feature_name	Name of the feature.
version	*Version* of the feature.
index	Used to specify a particular client.
commuter_info	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 397.

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
feature_name	Name of the feature.
*feature*_version	*Version* of the feature.
units_reqd	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 *unitsReqd*. If *unitsReqd* is NULL, a value of 1 unit is assumed.
duration	Displays the number of days for which the license has to be checked out.
lock_mask	Mask defining which fields are to be used for locking. On entry, *lock_mask* 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 *lock_mask* for Disk ID or host name before sending this value to the license server.
log_comment	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.
challenge	The challenge-response for this operation. Pointer to a *challenge* structure. The challenge-response will also be returned in this structure.

Description Obtains the commuter code from the license server and installs the stand-alone commuter authorization at the client.

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

VLSgetAndInstallCommuterCode Error Codes

Error Code	Description
VLS_CALLING_ERROR	• *duration* is NULL • *lock_mask* is NULL.

For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

VLSuninstallAndReturnCommuterCode

Syntax

```
int VLSuninstallAndReturnCommuterCode(
unsigned char  *feature_name,
unsigned char  *feature_version,
unsigned char  *log_comment);
```

Argument	Description
feature_name	Name of the feature.
feature_version	*Version* of the feature.
log_comment	A string to be written by the license server to the comment field of the usage log file. Pass a NULL value for this argument if no log comment is desired.

Description Uninstalls the commuter authorization from the client and returns the commuter authorization to the license server.

Returns The status code VLScg_SUCCESS is returned if successful. For a complete list of the error codes, see Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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.

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. If NULL is passed as the locking criteria in the VLSgetMachineIDString call, then VLSgetMachineIDString picks up all the fingerprint info that is available on the computer.

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
lock_selector	Bitmask identifying what criteria you would like to be contained in the Machine ID string. See "lock_selector Values" on page 300 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.

Argument	Description
machineIDString	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.
bufSz	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 397.

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
feature_name	The feature name of the license to check out from the license server.
feature_version	The feature version of the license to check out from the license server.
units_reqd	Number of units required for the license.
duration	Number of days the commuter authorization will last. This value may be superseded by the maximum limit allowed by the license.
lock_mask	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.
log_comment	A comment which will be placed inside the log file on the license server.
machineIDString	Machine ID string generated by the remote computer desiring the commuter authorization.
commuter_code	The actual commuter authorization code. This string should be passed on to the remote computer desiring the commuter authorization for installation.
challenge	A challenge and response for the given license on the license server. Set to NULL if you are not using this feature.
requestInfo	Reserved. Use NULL for this value.
commuterInfo	To be used in future for hooks.
reserved1	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 397.

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 299). 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
commuter_code	The commuter authorization that was generated by a computer with network access to the license server.
reserved1	Reserved. Use NULL for this value.
reserved2	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 397.

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
VLSgetCapacityFrom Handle	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   *unused2);
```

Argument	Description
license_system	Unused. • Use LS_ANY as the value of this variable. • LS_ANY is specified to indicate a match against the installed license system.
publisher_name	• A string identifying the publisher of the product. Limited to 32 characters and cannot be NULL. • Company name and trademark may be used.

Argument	Description
product_name	• Name of the feature for which a license code is requested. • May consist of any printable characters and cannot be NULL. • Limited to 24 characters.
version	• Version of the feature for which a license code is requested. • May consist of any printable characters. Limited to 11 characters. • Version can be NULL.
units_reqd	• 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. • 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 *units_reqd.* If *units_reqd* is NULL, a value of 1 unit is assumed. • To use the capacity licensing it is necessary that units required be always 1.
log_comment	• 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.
challenge	• The challenge structure. If challenge-response mechanism is not being used, this pointer must be NULL. • The response to the challenge is provided in the same structure, provided a license was issued, i.e., provided the function VLSrequestExt2 returns LS_SUCCESS.

Argument	Description
lshandle	• The handle for this request is returned in *lshandle.* 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 *lshandle* must be allocated by the caller.
serverInfo	• This information is passed to the license server for use in server hook functions. • VLSinitServerInfo must be called to initialize serverinfo.
team_capacity_reqd	• Required team capacity • 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. • If the request is made for a non-capacity license, this must be passed as NULL.
capacity_reqd	• Required user capacity • 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. • If the request is made for a non-capacity license this must be passed as NULL.
unused1	Reserved for future use.
unused2	Reserved for future use.

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 exhuasted
- User capacity has been exhausted
- Team capacity has been exhauseted 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	• *featureName* is NULL • *version* is NULL Both feature name and version cannot be NULL at the same time.
VLS_CALLING_ERROR	• *lshandle* is NULL • *challenge argument* is non NULL • Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
VLS_NO_LICENSE_GIVEN	• *unitsReqd* is zero • *lshandle* is not a valid handle
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	• License server has determined that the client system lock has been modified. • The license for this feature has time tampering protection enabled, so the license operation is denied.

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 *hostName* was specified.
VLS_NO_SERVER_FILE	• No license server has been set • Unable to determine which license server to use.
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 397.

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
feature_name	Name of the feature.
version	Version of the feature.
capacity	Capacity of the feature.
index	Used to specify a particular feature.
unused1	Use NULL as value.
unused2	Use NULL as value.
feature_info	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	• *featureinfo* is NULL • *index* is negative • Attempted to use stand-alone mode with network-only library, or network mode with stand-alone library.
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 *hostName* 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 397.

VLSgetCapacityList

Syntax

```
VLSgetCapacityList(
ifndef LSNOPRONTO
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
feature_name	Name of the feature.
feature_version	Version of the feature.
index	Returns the index of the license up to which the capacity has been retrieved based on the bufferSize
bufferSize	Specifies the size of *capacityList*.
capacityList	An array containing a list of all the capacities available for this feature and version, separated by space. The caller should allocate the space.
log_comment	Use NULL as value.
unused1	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	*featureName* 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 *hostName* 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 397.

VLSgetClientInfoExt

Syntax

```
VLSgetClientInfoExt (
unsigned char   *feature_name,
unsigned char   *version,
unsigned long   *capacity,
int             index,
char            *log_comment,
VLSclientInfo   *client_info);
```

Argument	Description
feature_name	Name of the feature.
version	*Version* of the feature.
capacity	*Capacity* of the feature.
index	Used to specify a particular client.
log_comment	Comment.
client_info	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	• *featureName* is NULL. • *version* is NULL Both feature name and version cannot be NULL at the same time.
VLS_CALLING_ERROR	• *clientInfo* parameter is NULL • *index* 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_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 *hostName* 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 397.

VLSdeleteFeatureExt

Syntax

```
VLSdeleteFeatureExt(
unsigned char  *feature_name,
unsigned char  *version,
unsigned long  *capacity,
unsigned char  *log_comment,
LS_CHALLENGE   *challenge);
```

Argument	Description
feature_name	Name of the feature.
version	*Version* of the feature.
capacity	*Capacity* of the feature.
log_comment	Unused
challenge	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	• *featureName* is NULL. • *version* 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.

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 *hostName* 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 397.

VLSgetCapacityFromHandle

Syntax

```
VLSgetCapacityFromHandle(
LS_HANDLE       lshandle,
unsigned long LSFAR *team_capacity,
unsigned long LSFAR *user_capacity
unsigned long LSFAR *license_capacity);
```

Argument	Description
handle	Handle
team_capacity	Team capacity allocated to the handle and issued by the server
user_capacity	User capacity allocated to the handle and issued by the server
license_capcity	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 397.

VLSsetTeamId

See “VLSsetSharedId/ VLSsetTeamId” on page 70.

VLSsetTeamIdValue

See “VLSsetSharedIdValue/ VLSsetTeamIdValue” on page 72.bb

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 *ucodeT* 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 *msgNum* 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 *base_feature_name* in the *ucodeT* struct.
VLSucgAllowBase FeatureVersion	Identifies whether the corresponding VLSucgSetBaseFeatureVersion should be called or not.
VLSucgSetBaseFeature Version	Sets the value of *base_feature_version* in the *ucodeT* struct.
VLSucgAllowUpgrade Code	Identifies whether the corresponding VLSucgSetUpgradeCode API should be called or not
VLSucgSetUpgrade Code	Sets the value of *base_lock_code* in the *ucodeT* struct to the value in the *upgrade_code*
VLSucgAllowUpgrade Flag	Identifies whether the corresponding VLSucgSetUpgradeFlag should be called or not
VLSucgSetUpgrade Flag	Sets the value of *upd_flags* in the *ucodeT* struct.
VLSucgAllowUpgrade Version	Identifies whether the corresponding VLSucgSetUpgradeVersion should be called or not
VLSucgSetUpgrade Version	Sets the value of *upd_version* in the *ucodeT* struct.
VLSucgAllowUpgrade Capacity	Identifies whether the corresponding VLSucgSetUpgradeCapacityUnits and VLSucgSetUpgradeCapacity should be called or not
VLSucgSetUpgrade CapacityUnits	Sets the value of *capacity_units* in the *ucodeT* struct.
VLSucgSetUpgrade Capacity	Sets the value of *capacity_increment* in the *ucodeT* struct.

Upgrade License Code Generator Related Functions (Continued)

Function	Description
VLSucgGenerate License	Generates the upgrade license string for the given *ucodeT* struct
VLSucgGetLicense MeterUnits	Returns the number of license generation units available in the attached dongle
VLSGenerateUpgrade LockCode	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
structSz	Size of the structure.
Vendor_code	Internal use
version_num	Upgrade license code generation library version

ucodeT Struct (Continued)

Member	Description
base_feature_name	Feature Name of the base license that needs to be upgraded
base_feature_version	Feature Version of the base license that needs to be upgraded
lock_code	A unique code to identify base licenses which needs to be upgraded.
generation_time	This value shall be set automatically during the license generation time in GMT. It details about the time of license generation.
generation_sequence	This value shall be set at license generation time along with generation_time to ensure that on a fast system, even if two licenses are generated at the same time, this value should be different.
upd_flags	Bit-wise flag. Will control what will be updated • VLSucg_UPGRADE_VERSION • VLSucg_UPGRADE_CAPACITY • VLSucg_UPGRADE_ALL
upd_version	New version for this feature.
capacity_units	Flag which determines capacity least count
capacity_increment	Capacity increment for this feature.
Unused	For future use.
Unused	For future use.

VLSucgInitialize

Syntax

```
int VLSucgInitialize(
VLSucg_HANDLE   *iHandle);
```

Argument	Description
iHandle	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 and Result Codes for Upgrade License Functions," on page 423.

VLSucgCleanup

Syntax

```
int VLSucgCleanup(
VLSucg_HANDLE  *iHandle);
```

Argument	Description
iHandle	Instance handle for this library

Description Destroys the handle created using VLSucgInitialize.

VLSucgCleanup cleanups 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 and Result Codes for Upgrade License Functions," on page 423.

VLSucgReset

Syntax

```
int VLSucgReset(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* 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 ucodeP is passed as NULL

For a complete list of the error codes, see Appendix E, "Error and Result Codes for Upgrade License Functions," on page 423.

VLSucgGetNumErrors

Syntax

```
int VLSucgGetNumErrors(
VLSucg_HANDLE  iHandle,
int            *numMsgsP);
```

Argument	Description
iHandle	Instance handle for this library
numMsgsP	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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgGetErrorLength

Syntax

```
int VLSucgGetErrorLength(
VLSucg_HANDLE  iHandle,
int            msgNum,
int            *errLenP);
```

Argument	Description
iHandle	Instance handle for this library
msgNum	The number of the message whose length is to be queried, starts from 0.
errLenP	The length of messages identified by *msgNum*

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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgGetErrorMessage

Synatx

```
int VLSucgGetErrorMessage(
VLSucg_HANDLE  iHandle,
char           *msgBuf,
int            bufLen);
```

Argument	Description
iHandle	Instance handle for this library
msgBuf	A user allocated buffer into which the reference message will be copied
bufLen	The byte length of the message copied into *msgBuf*

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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgPrintError

Syntax

```
int VLSucgPrintError(
VLSucg_HANDLE  iHandle,
FILE           *file);
```

Argument	Description
iHandle	Instance handle for this library
file	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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgAllowBaseFeatureName

Syntax

```
Int VLSucgAllowFeatureName(
VLSucg_HANDLE   iHandle,
ucodeT          *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* 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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgSetBaseFeatureName

Syntax

```
int VLSucgSetBaseFeatureName(
VLSucg_HANDLE  iHandle,
ucodeT         *ucodeP,
char           *feature_name);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* struct
feature_name	• Any printable ASCII text except #. • Maximum of 24 characters.

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
VLSucg_INVALID_CHARS	Invalid characters in feature_name.
VLSucg_NO_FEATURE_NAME	If feature_name is NULL.
VLSucg_RESERV_STR_ERROR	If the feature_name is a reserved string.
VLSucg_EXCEEDS_MAX_ VALUE	If the length of feature_name string exceeds maximum allowed length(24 char).

For a complete list of the error codes, see Appendix E, “Error and Result Codes for Upgrade License Functions,” on page 423.

VLSucgAllowBaseFeatureVersion

Syntax

```
int VLSucgAllowBaseFeatureVersion(
VLSucg_HANDLE  iHandle,
ucodeT         *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* 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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgSetBaseFeatureVersion

Syntax

```
int VLSucgSetBaseFeatureVersion(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP,
char          *feature_version);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* struct
feature_version	• Any printable ASCII text except #. • Maximum of 11 characters.

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 *feature_version* characters are not printable.
VLSucg_RESERV_STR_ERROR	If the *feature_version* is a reserved string.
VLSucg_EXCEEDS_MAX_ VALUE	If the length of *feature_version* string exceeds maximum allowed length(11 char).

For a complete list of the error codes, see Appendix E, “Error and Result Codes for Upgrade License Functions,” on page 423.

VLSucgAllowUpgradeCode

Syntax

```
int VLSucgAllowUpgradeCode(
VLSucg_HANDLE   iHandle,
ucodeT          *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library
ucodeP	The pointer to *ucodeT* 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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgSetUpgradeCode

Syntax

```
int VLSucgSetUpgradeCode(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP,
char          *upgrade_code);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* struct.
upgrade_code	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 *upgrade_code* string exceeds maximum allowed length.
VLSucg_FAIL	On Failure.

For a complete list of the error codes, see Appendix E, "Error and Result Codes for Upgrade License Functions," on page 423.

VLSucgAllowUpgradeFlag

Syntax

```
int VLSucgAllowUpgradeFlag(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* 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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgSetUpgradeFlag

Syntax

```
int VLSucgSetUpgradeFlag(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP,
char          *flag);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* struct.
flag	The value of flag is used to set the upd_flags of ucodeT struct. Legal values are bit combinations of • VLSucg_UPGRADE_VERSION • VLSucg_UPGRADE_CAPACITY • VLSucg_UPGRADE_ALL

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 and Result Codes for Upgrade License Functions,” on page 423.

VLSucgAllowUpgradeVersion

Syntax

```
int VLSucgAllowUpgradeVersion(
VLSucg_HANDLE   iHandle,
ucodeT          *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* 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 and Result Codes for Upgrade License Functions," on page 423.

VLSucgSetUpgradeVersion

Syntax

```
int VLSucgSetUpgradeVersion(
VLSucg_HANDLE   iHandle,
ucodeT          *ucodeP,
char            *upgrade_version);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* struct.
upgrade_version	• Any printable ASCII except #. • Maximum of 11 characters.

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 VLScg_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 ucodeP or upgrade_version is passed as NULL. Also if the upgrade _version does not contain a valid string.
VLSucg_INVALID_CHARS	If upgrade_version characters are not printable.
VLSucg_RESERV_STR_ERROR	If the upgrade_version is a reserved string.
VLSucg_EXCEEDS_MAX_ VALUE	If the length of upgrade_version string exceeds maximum allowed length.

For a complete list of the error codes, see Appendix E, "Error and Result Codes for Upgrade License Functions," on page 423.

VLSucgAllowUpgradeCapacity

Syntax

```
int VLSucgAllowUpgradeCapacity(
VLSucg_HANDLE  iHandle,
ucodeT         *ucodeP);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* 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:

VLSucgAllowUpgradeCapacity Error Codes

Error Code	Description
1	VLSucgSetUpgradeCapacity is allowed
0	VLSucgSetUpgradeCapacity is not allowed

For a complete list of the error codes, see Appendix E, "Error and Result Codes for Upgrade License Functions," on page 423.

VLSucgSetUpgradeCapacityUnits

Syntax

```
int VLSucgSetUpgradeCapacityUnits(
VLSucg_HANDLE  iHandle,
ucodeT         *ucodeP,
char           *cap_units);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* struct.
cap_units	Capacity specification units: from 0 to 4. The values are: • If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022. • If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220. • If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200. • If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000. • If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000.

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 and Result Codes for Upgrade License Functions,” on page 423.

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 • If capacity_units is 0, capacity shall be multiple of 1(s), maximum 1022. • If capacity_units is 1, capacity shall be multiple of 10(s), maximum 10220. • If capacity_units is 2, capacity shall be multiple of 100(s), maximum 102200. • If capacity_units is 3, capacity shall be multiple of 1000(s), maximum 1022000. • If capacity_units is 4, capacity shall be multiple of 10000(s), maximum 10220000. **NOLIMITSTR or EMPTY("/0") String can be used to specify infinite capacity.**

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 and Result Codes for Upgrade License Functions," on page 423.

VLSucgGenerateLicense

Syntax

```
int VLSucgGenerateLicense(
VLSucg_HANDLE iHandle,
ucodeT        *ucodeP,
char          *upgrade_code,
char          **result);
```

Argument	Description
iHandle	Instance handle for this library.
ucodeP	The pointer to *ucodeT* struct.
upgrade_code	Upgrade code of base license
result	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 VLScg_SUCCESS is returned if successful. Otherwise, it will return the following error codes:

VLSucgGenerateLicense Error Codes

Error Code	Description
VLSucg_INVALID_INPUT	If the ucodeP is passed as NULL.
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 ucodeT struct.
VLSucg_LICMETER_ EXCEPTION	If error occur while accessing the dongle.
VLSucg_NO_NETWORK_ AUTHORIZATION	If not authorized to generate network licenses.
VLSucg_LICMETER_ COUNTER_TOOLOW	If license meter count is less than the expected decrement count.
VLSucg_NO_CAPACITY_ AUTHORIZATION	If not authorized to generate capacity licenses.
VLSucg_NO_UPGRADE_ AUTHORIZATION	If not authorized to generate upgrade licenses.
VLSucg_INTERNAL_ERROR	If any internal error occur while generating the license string.
VLSucg_INVALID_BASE_LIC _INFO	The information-feature name, version vendor code provided for base license is incorrect.
VLSucg_CAPACITY_UPD_ NOT_ALLOWED	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 and Result Codes for Upgrade License Functions," on page 423.

VLSucgGetLicenseMeterUnits

Syntax

```
int VLSucgGetLicenseMeterUnits(
VLSucg_HANDLE  iHandle,
long           *initialUnitsP,
long           *unitsLeftP,
int            ucodegen_version);
```

Argument	Description
iHandle	Instance handle for this library.
initialUnitsP	User provided license string to be decoded.
unitsLeftP	User allocated buffer to receive decoded license string.
ucodegen_version	Version of the ucodegen library

Description Returns the number of license generation units available in the attached dongle.

Returns The status code VLScg_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 *ucodeT* 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 and Result Codes for Upgrade License Functions," on page 423.

VLSgenerateUpgradeLockCode

Syntax

```
int VLSgenerateUpgradeLockCode(
unsigned char   *lic_string,
unsigned char   *upgrade_lock_code,
unsigned long   *buffer_size);
```

Argument	Description
lic_string	Base license string.
upgrade_lock _code	Buffer containing the generated upgrade lock code. The caller should allocate the memory space.
buffer_size	• Size of the allocated buffer. • If NULL is passed instead of buffer, then this will return buffer size required for the generated upgrade lock code.

Description

VLSgenerateUpgradeLockCode 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	• *buffer* parameter is NULL. • Size of upgrade lock code exceeds *buffer_size* parameter.

For a complete list of the error codes, see Appendix E, "Error and Result Codes for Upgrade License Functions," on page 423.

Upgrade License Decode API

The following table summarizes the upgrade license decode related functions:

Upgrade License Decode Related Functions

Function	Description
ulcCode	Stores information required to decode upgrade lock code.

Upgrade License Decode Related Functions (Continued)

Function	Description
VLSdecodeUpgrade lockCode	Decodes the upgrade lock code.
VLSucgDecodeLicense	Decodes the encrypted license string generated by 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
version_num	Number maintaing the version of the structure
hash_vendor_ string	A numeric value representing the feature and version of the license.

ulcCode Struct

Member	Description
capacity_flag	The value of capacity flag.
standalone_flag	The value of standalone flag.
num_keys	The number of keys
birth_day	The starting day of the license.
birth_month	The starting month of the license.
birth_year	The starting year of the license.
death_day	The expiration day of the license.
death_month	The expiration month of the license.
death_year	The expiration year of the license.
client_server_lock_mode	The locking mode
base_lock_code	Base lock code
base_feature_name	Base feature name
base_feature_version	Base feature version
capacity	Capacity of the license.

VLSdecodeUpgradelockCode

Syntax

```
int VLSdecodeUpgradelockCode(
  char        *upgrade_lock_code,
  char        *compacted_upd_lock_code,
  int         length,
  ulcCode     **ulcCodePP );
```

Argument	Description
upgrade_lock_code	Upgrade lock code to be decoded.

Argument	Description
compacted_upd_lock_code	Upgrade lock code string after removing comment chars and white spaces . This can also be set as null.
length	Length of compacted_upd_lock_code in case it is not null.
ulcCodePP	Pointer to pointer to ulcCode structure.

Description VLSdecodeUpgardelockCode 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 *ucodeT* struct

For a complete list of the error codes, see Appendix C, “Sentinel LM Error and Result Codes,” on page 397.

VLSucgDecodeLicense

Syntax

```
int VLSucgDecodeLicense(
VLSucg_HANDLE iHandle,
char          *AnyLicenseString,
char          *lic_string,
int           lic_string_length,
ucodeT        **ucodePP);
```

Argument	Description
iHandle	Instance handle for this library.
AnyLicenseString	User provided license string to be decoded.
Lic_string	User allocated buffer to receive decoded license string.

Argument	Description
Lic_string_length	Length of decoded license string returned.
ucodePP	• Address of pointer to *ucodeT* struct • Contains input license string.

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 *ucodeT* struct
VLSucg_FAIL	On Failure.

For a complete list of the error codes, see Appendix E, “Error and Result Codes for Upgrade License Functions,” on page 423.

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
hostName	The host name of the computer containing the license server that is using the log file.
newFileName	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 397.

VLSgetUsageLogFileName

Syntax

```
int VLSgetUsageLogFileName(
char  *hostName,
char  *fileName);
```

Argument	Description
hostName	The host name of the computer containing the license server that is using the log file.
fileName	The name of the existing usage log file is returned in this argument.

Description Determines the name of the existing usage log file.

Returns The status code LS_SUCCESS is returned if successful. Otherwise, a specific error code is returned indicating the reason for failure. For a complete list of the error codes, Appendix C, "Sentinel LM Error and Result Codes," on page 397.

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
seconds	Time interval in *seconds*.
eventHandler	Signal handler.
repeat_event	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 397.

VLSdisableEvents

Syntax

```
int VLSdisableEvents(
void     *eventHandler);
```

Argument	Description
eventHandler	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 397.

VLSeventSleep

Syntax

```
int  VLSeventSleep(
void VLSeventSleep (unsigned int seconds));
```

Argument	Description
seconds	Time in *seconds* to sleep.

Description This function is called for disabling the license operations for an allotted time and interferes with the system alarms.

VLSeventSleep must be used in conjunction with VLSdisableAutoTimer.

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 397.

Appendix A
Sample Applications

Sentinel LM installs a few sample applications on your computer. These sample applications can be located at *\Rainbow Technologies\Sentinel LM\7.X.X\English\MsvcDev\Sample.*

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

Customization Sample Files on UNIX

Component	File(s)
linking	*Makefile*
the license manager	*server.o*
lsdecode	*lsde.o*
lslic	*lslic.c*
lsmon	*lsmon.c*
lswhere	*lswhere.c*
Challenge-response	*crexamp.c, chalresp.[c h], md4.[c h]*

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

Customization Sample Files on Windows

Component	File(s)
the license server	*lservdown.[c dsp], lserv.h*
licence generator	*echoid32.dsp, echomain.c*
lslic	*lslic.[c dsp]*
lsmon	*lsmon.[c dsp]*
lswhere	*lswhere.[c dsp dsw ncb opt]*
Challenge-response	*crexamp.c, chalresp.[c h], md4.[c h]*
Sample function macros	*dots, bounce*

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 xxv) 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\Rainbow Technologies\Sentinel LM\MsvcDev\custom*. In this directory you will find the Makefile *custom32.mak*. Make a copy of this file and name it **MAKEFILE**. Edit this file. Add your customized object files in the following section:

```
# For now, use the default functions from the 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.

VLSserverVendorInitialize

Client	Server	Static Library	DLL
	✔	✔	

Description Initializes the server.

Syntax

```
LSERV_STATUS VLSserverVendorInitialize (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. ■ 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: r*eqprhk1.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 Rainbow Technolgies, Inc.   */
/*              All Rights Reserved             */
/*                                              */
/************************************************/

#include "lservcst.h"
extern int LSReqPreHook1(VLShandlerStruct
*handleStruct, char *inBuf, char *outBuf, int outBufSz);
extern int LSReqPostHook1(VLShandlerStruct *handleStruct,
char *inBuf, char *outBuf, int outBufSz);
extern int LSRelPreHook1(VLShandlerStruct *handleStruct, char
*inBuf, char *outBuf, int outBufSz);
extern int LSRelPostHook1(VLShandlerStruct *handleStruct,
char *inBuf, char *outBuf, int outBufSz);
extern int LSReqPreHook2(VLShandlerStruct *handleStruct, char
*inBuf, char *outBuf, int outBufSz);
extern int LSReqPostHook2(VLShandlerStruct *handleStruct,
char *inBuf, char *outBuf, int outBufSz);
extern int LSRelPreHook2(VLShandlerStruct *handleStruct, char
*inBuf, char
*outBuf, int outBufSz);
extern int LSRelPostHook2(VLShandlerStruct *handleStruct,
char *inBuf, char
*outBuf, int outBufSz);
LSERV_STATUS VLSserverVendorInitialize(void) {
#ifndef _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
actionOnTmTamper	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]
tmTamperMethod	Whether to use the Sentinel LM built-in system clock tamper checking function, or use one provided by you. [VLS_ENABLE_DEFAULT_TM_TAMPER]
gracePeriod	Useful only in case *tmTamperMethod* is VLS_ENABLE_DEFAULT_TM_TAMPER. If Sentinel LM finds the system clock has been set back by less than *gracePeriod* seconds, it will not count the offending system file as a violation.
percentViolations	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.
numViolationsForError	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, **lsdecode**. All three programs must be re-linked. Licensed applications do not encrypt or decrypt license codes. Client applications need not be re-linked.

Note: Encryption is not available for stand-alone licenses.

VLSencryptLicense

Client	Server	Static Library	DLL
	✔	✔	

Encrypts license codes.

Syntax

```
int VLSencryptLicense(
char   *origText;
char    *encryptedTextBuffer;
int     buffSize);
```

Argument	Description
origText	The original license code.
encryptedTextBuffer (OUT)	The encrypted license code to be returned.
buffSize	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 Rainbow Technologies, Inc.  */
/*                All Rights Reserved               */
/*                                                 */
/*************************************************/
/* Usage of VLSencryptLicense() */
#include <stdio.h>
#include <string.h>
#include "lstest.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
origText	The original license code.
decryptedTextBuffer (OUT)	The decrypted license code to be returned.
buffSize	Size of the decrypted text buffer.

Description See VLSencryptLicense above.

Example file:

```
/************************************************/
/*                                              */
/*  Copyright (C) 2004 Rainbow Technologies, 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
origText	The original message text.
encryptedTextBuffer (OUT)	The encrypted message text.
buffSize	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 Rainbow Technologies, Inc. */
/*               All Rights Reserved             */
/*                                              */
/************************************************/
/* Usage of VLSencryptMsg() */
#include <stdio.h>
#include <string.h>
#include "lstest.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
origText	The original message text.
decryptedTextBuffer (OUT)	The decrypted message text.
buffSize	Size of the decrypted text buffer.

Description See VLSencryptMsg on the previous page.

Example file:

```
/************************************************/
/*                                              */
/* Copyright (C) 2004 Rainbow Technologies, Inc.  */
/*              All Rights Reserved               */
/*                                              */
/************************************************/
/* Usage of VLSdecryptMsg() */
#include <stdio.h>
#include <string.h>
#include "lstest.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
currentPort	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 Rainbow Technologies, Inc. */
/*                 All Rights Reserved          */
/*                                              */
/************************************************/
#include "lservcst.h"
#include "lserv.h"
#include <stdio.h>
#ifdef __STDC__
int VLSchangePortNumber(int newPort)
#else
int VLSchangePortNumber(newPort)
int newPort;
#endif
{
  newPort=6000;
  return(newPort);
}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_DI
SK_ID|VLS_LOCK_HOSTNAME|VLS_LOCK_ETHERNET|VLS_LOCK_NW_IPX|VLS
_LOCK_NW_SERIAL|VLS_LOCK_PORTABLE_SERV,&machineID,&lock_selec
tor_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%lX",customid);
}
```

Using a Customized Host ID

The sequence of events for an application using a custom ID is as follows:

1. Generate client node lock and/or server node locked licenses with the custom host ID as returned by *myechoid.exe.*

2. Rebuild and execute the customized **lserv**.

3. In the client application set the host ID function to getCustomHostId.

   Now the client side host ID has been changed.

4. Add the client node lock license to the server.

   When an application tries to request a key for a client node-locked license, the server then verifies the client host ID as sent in the request message and compares it with the host ID in the license.

5. In the case of server locking to a customized host ID, when a server-locked license is added to the server, it executes the VLSserverVendorInitialize 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
original_text	IN	The string to be encrypted
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

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 125.

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.
- *lservrccnf* - 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 397.

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 Rainbow 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
ServerInit.cpp	C++ source file containing re-definition of VLSserverVendorInitialize function.
CustomHostID.cpp	C++ source file containing custom locking code definition.
CustomHostID.h	C++ include file containing custom locking code prototype.
lsmainwa.c	C source file containing entry point to license server application.
lserv.h	Sentinel LM include file installed during Sentinel LM installation.
lserv95.res	Resource file for license server application.
lserv9x.lib	Sentinel LM static library installed during Sentinel LM installation.
lserv9x.dsp	Microsoft Visual C++ 6.0 project file.
lserv9x.dsw	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 VLSserverVendorInitialize is redefined so that it will call VLSsetHostIdFunc. VLSserverVendorInitialize is called during server startup for both default license servers and custom license servers, but the default version does not call VLSsetHostIdFunc.

VLSsetHostIdFunc 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.

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

Steps to Rebuilding the License Server

1. Obtain a zip file from Rainbow Technologies that contain all the necessary files. Please see "Files Required" on page 390. 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 393.

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 Rainbow 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
CustomHostID.c	C source file containing custom locking code definition.
CustomHostID.h	C include file containing custom locking code prototype.
echoid.c	C source file containing logic for generating fingerprints. Notice, this file will be modified to call the custom locking code function.
lscgen.h	Sentinel LM include file installed during Sentinel LM installation.
lserv.h	Sentinel LM include file installed during Sentinel LM installation.
lsapiw32.lib	Import library for Win32 run-time *DLL* that is installed during Sentinel LM installation.
echoid.dsp	Microsoft Visual C++ 6.0 project file
echoid.dsw	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 Rainbow Technologies that contain all the necessary files. Please see "Files Required for echoid.exe" on page 392. 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 393.

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 393.

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 390.
3. Rebuild *echoid.exe*. See "Step 2 - Rebuilding echoid.exe" on page 392.
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 *lservrc*.
8. Modify the client application and rebuild it. See "Step 3 - Modifying Client Application" on page 393.
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.
0xC80010009	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-sever 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. Dbfault: 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 *feature, version*, 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_SERVE R_FOUND) is returned when one trys 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	*lshandle* is active handle.	*lshandle* 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.	*hostName* 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.	*IP_address* is valid, but could not be resolved.
57		VLS_UNRESOLVED_HOSTNAME	Hostname given is unresolved.	*IP_address* is valid, but could not be resolved.
58		VLS_INVALID_IP_ADDRESS	Invalid IP address format.	*IP_address* 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_COMMUTE R_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. VLSuninstallAndReturn CommuterCode 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: • 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.b
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.

Appendix D

Error and Result 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.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
6	VLScg_EXCEEDS_MAX_ STRLEN	Length of <*value*> is greater than <*value*>.	Length exceeds the maximum string length.
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.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
20	VLScg_DECRYPT_FAIL	Decryption failed for license code.	Decryption failed for license code.
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 *codeP* or *flag* 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.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
35	VLScg_INVALID_ VENDOR_CODE	Invalid Vendor Code. Please contact your Sentinel LM distributor.	If vendor identification is illegal.
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_TOOLOW	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_F AIL	Error in updating locale.	Error in updating locale.

Sentinel LM License Code Generation Return Codes (Continued)

Error No	Return Code	Default Message	Description
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.
70	VLScg_LICMETER_NOT_ SUPPORTED	Your Sentinel LM License Meteris not supported.	Your Sentinel LM License Meteris not supported.

Appendix E

Error and Result 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*>.	Lenght of 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_VERS ION_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 *upgrade_code* 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_TOOLOW	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 Meteris not supported.	Your Sentinel LM License Meteris 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, LSERVRCCNF, 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 *Setinel 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 Capa-city	User Capa-city	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. • 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 *log_comment* 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

D

E

F

T

U

W