Table of Contents

When viewing Sheriff Help as a PDF format file, please click on Bookmarks to see the Contents

1.1 Guide to Help Topics

TOPIC	CONTENTS	
Quick Start/Upgrading	How to get up and running with Sheriff as quickly as possible. Users of Sheriff 2 who are upgrading to 3.0 should read topics 3.4 and 6.11.	
Technical Overview	A more detailed look at the technical aspects of Sheriff.	
Developers' Guide	How to implement Sheriff via traditional DLL or ActiveX. Also explains Internet Software Registration (Sheriff ISR).	
Applications	SlsDemo	A simple example of a Sheriff-protected progam.
	SlsPsn	Generates unique Product IDs & Secret Codes with which to protect software applications.
	SlsGen	Generates licence keys for end users.
	SlsAdmin	Enables end users to manage their licence keys. Includes information about networking.
	SlsClock *Obsolete*	[Ensured that the Sheriff Clock was synchronised over a network. Not required in Sheriff 3.0 so not included in SDK.]
	SlsServer	Application that runs on the server in a networked environment.
	SlsPsn	To generate your unique Product ID and Secret Codes
FAQ	The answers to frequently asked questions.	
Licence Agreement	The End User Licence Agreement (EULA) for the Sheriff SDK and the provision of support & maintenance.	

1.2 Evaluating Sheriff

Pricing

Please click here to view the current price list at our web site.

Evaluation Restrictions

You can evaluate the full Sheriff SDK for as long as you like. There is no time limit to the evaluation period and there are no restrictions other than those that apply to all Sheriff users, as set out in the End User Licence Agreement .

Until you purchase your own unique Sheriff Serial Number you will need to use the Evaluation Product ID and Secret Codes. The only restriction with using these is that the licence keys you produce will expire after 30 days. In every other respect you will be using Sheriff exactly as if you are a registered user.

Obtaining a Product ID & Secret Codes

Purchasers of Sheriff receive a unique Serial Number with which to generate their Product ID and Secret Codes. To generate a unique Product ID and Secret Codes enter your Serial Number in the Sheriff Product ID and Secret Codes Generator (SlsPsn).

Each Serial Number can generate only one set of Product ID and Secret Codes, so if you want to protect more than one product you will need to purchase a Serial Number for each additional product. One set of Product ID and Secret Codes will protect an unlimited number of copies of a single product.

Demo Code and Applications

There is demo code for Visual C++ and Visual Basic. The demo applications are named SlsDemoLocal.exe and SlsDemoRemote.exe (we no longer provide demo apps for FoxPro or Delphi).

Note for Visual C++ Developers

IMPORTANT NOTE: The information below applies only to Sheriff 2. Static libraries are not supported in Sheriff Version 3.

1.3 Requirements

System

Client and server work with all versions since Windows 2000 including 64-bit versions. However, to use Sheriff on Windows 2000 you need to use the provided slsService_legacy.exe (rename to slsService.exe). This is only necessary on Windows 2000 installations.

Network Operating System: Windows

Language Support

Sheriff supports any Windows programming language that can call a Windows DLL. The Sheriff SDK provides the source codes of template classes and demo applications for Visual C++ and Visual Basic.

1.4 Contact Information

Web : www.sheriff-software.com

Email : support@sheriff-software.com

Address : Licensing Technologies Limited
22 Wycombe End
Beaconsfield
Buckinghamshire
HP9 1NB
England

2.1 What is Sheriff?

>> Sheriff is a software development kit (SDK) for protecting software against illegal copying.

>> Sheriff uses licence keys (strings of numbers) to provide secure protection without the need for dongles or key disks.

>> Sheriff ties licences to individual machines or to a network file server.

>> Sheriff enables you to distribute your software product as a demo, a trial application, a limited feature application or a full feature application.

>> Sheriff enables you to create licences, on a user by user basis, that feature one or more of the following schemes:

Day Metering	Determines how many days the licence is valid before it expires.
Unit Metering	Determines how many times an application can run before the licence expires.
Expiration Control	Forces the licence to expire on a given date.
Concurrency	Limits the maximum number of concurrent users on a network. Sheriff treats many invocations of an application by the same user on the same computer as a single user.
Feature Access Control	Lets you restrict which features of your application are enabled for a given licence.

2.2 How Does Sheriff Work?

Sheriff uses an encrypted Licence Key (a sequence of numbers) to authorise an application to run on a given machine.

1. When a user installs or runs your application for the first time on a particular machine, they are presented with a Reference Code that is unique to that machine.

2. The user passes the Reference Code on to you and you use the Sheriff Licence Generator application (SlsGen) to turn the Reference Code into a Licence Key.

3. You give the Licence Key to the user and they enter it into the dialog box that gave them the Reference Code.

4. Sheriff records the Licence Key in a securely encrypted Licence Database.

5. Whenever your application runs it calls Sheriff to check whether the application is licensed on that machine; it does this by passing the application's unique Product ID to Sheriff for checking against the Licence Database. If this validation is successful your application runs normally, if unsuccessful your application can either exit or continue running in demo/evaluation mode.

Your application is now protected from illegal copying - it is bound to the user's hardware.

Some points worth noting:

- Steps 2 & 3 can be automated via the Internet using Sheriff ISR

- Users can temporarily export their licence key(s) to another machine. They do this using the Sheriff Administrator application (SlsAdmin) that you can supply along with your application (or you can replicate its functions in your app via the Sheriff API). Users can also permanently export the original licence file to another machine. Either or both of these facilities can be disabled if you wish.

- The functions of the Sheriff Licence Generator for issuing licence keys etc. are made available to developers via the Extended API (SlsApiEx).

- You can create licences that restrict the use of your application to a given number of runs, a given number of days or to a given date.

- You can create licences that authorise a specified number of users to share your application over a network. While your application is running it periodically communicates with Sheriff to ensure the Licence Database knows your application is still running. Before an application exits it notifies Sheriff to free-up its resources, making them available for the next user. Should an application terminate abnormally Sheriff automatically reclaims the licence. Further details of running Sheriff-protected applications over a network can be found in the topics Sheriff Administrator and FAQ: Network Licences.

2.3 What is in the Sheriff SDK?

>> Code that you call from your application to copy protect it. There is code specific to Visual C++ and Visual Basic

Please see the API folder for the code for your development language.

>> Demonstration software that shows how to use the Sheriff SDK to protect your applications. There is also a demo application .

>> Product ID and Secret Codes Generator that you use to generate the unique identifying codes for your application (i.e. the Product ID and Secret Codes). An evaluation Product ID and Secret Codes are supplied.

>> The Licence Key Generator that you use to generate licence keys.

>> The Administrator application that your customers can use to administer their licences.

>> Instructions on how to implement Sheriff ISR (Internet Software Registration), which enables you to distribute your licence keys automatically from a web site.

>> The SlsServer application that enables you to manage licences on a network. Full instructions for running SlsServer are given in topic 3.4 Networking.

See the SDK Components section of this help file for a detailed list of the files in the kit.

2.4 Protecting an Application

Overview

How you integrate Sheriff with your application will depend upon your development environment:

Visual C++	Using the Windows Library, you can call the Sheriff DLL from your application.
All Others	You can use the Windows Libarary to call the Sheriff DLL from your application or you can use the ActiveX control.

ActiveX Control

The ActiveX control provides a quick and easy way of protecting applications written in environments that support this method, such as Visual Basic and Delphi. Further information is given in the Developers' Guide:

Introduction to ActiveX Control	Topic: Standard ActiveX Control - Introduction

Windows Library

The Windows library provides a flexible alternative to ActiveX. To protect a Windows application with Sheriff, access to the source code is required. However, it is very quick and easy to implement protection; a typical implementation requires four API function calls:

- SLS_Register
- SLS_Request
- SLS_Update
- SLS_Release

The Sheriff SDK provides ready-to-use template classes and demos for the popular Windows programming languages Visual C++ and Visual Basic. Further information is given in the Developers' Guide:

A quick guide to protecting an application using the Windows Standard Library	Topic: Standard Library - Quick Start
A step-by-step demonstration of protecting an application written in C++	Topic: Standard Library - Implementing the Demo
Sheriff SDK - Function Reference	Topic: Standard Library - API Reference
CSheriff Class Reference	Topic: Standard Library - Class Reference

3.1 Key Features

1. Machine Locking
2. Comprehensive Licence Policy
3. Portable Licence
4. Network Licence
5. Trial & Demo Licence
6. Administration Tools
7. Internet Software Registration

---

1. Machine Locking

Machine Locking ensures that an application will only execute on a PC or network against a valid licence. The licence cannot be copied to another machine nor can it be backed up and restored on the same machine. To achieve this, Sheriff employs an advanced proprietary algorithm for generating a unique 'fingerprint' of the machine where the licence database resides, as well as a unique "fingerprint" of the licence database itself.

2. Comprehensive Licence Policy

Sheriff offers the following licensing options:

- Day Metering
- Unit Metering
- Expiration Control
- Concurrency Control
- Feature Access Control
- Reusable Key

Day Metering enforces the lifespan of the application from the day of installation.

Unit Metering enables software publishers to decide what actions are chargeable and how they are charged.

Expiration Control forces an application to expire ('bomb out') on a specified date.

Concurrency limits the maximum number of concurrent users on a network. Sheriff treats many invocations of an application by the same user on the same computer as a single user.

Feature Access Control enables software publishers to categorise features in their products into many different levels and permits users to access only the levels that they have subscribed to.

Reusable Key enables a user to regenerate the licence database on the original machine without reference to the publisher in the event that the database file is damaged or overwritten. A Reusable Key is only issued when metering or expiration controls are not used.

3. Portable Licence

The portable licensing facility enables the user to export a licence from one machine to another, such as a portable PC (this facility can be disabled when issuing the licence key.

Normally, when a licence has been exported the protected application cannot execute on the original machine until the licence is re-imported, however the user can define the features of the exported licence within the terms of the original licence policy. So, for example, the exported licence may be set to expire after seven days and when it does the licence is automatically restored on the original machine.

4. Network Licence

Sheriff can be run on a network with a single licence database; typically this is installed on a file server. See *Networking* topic for further details.

5. Trial & Demo Licence

Typically, Sheriff-protected applications run in trial or demo mode if there is no valid licence present on the end user's machine. You can limit the number of days a trial is to last, specify a particular date when it will terminate or implement a demo version of your application with a limited feature set using *feature access control*. You can implement your trial application so that it is freely distributable but require a licence key once the trial period is over.

6. Administration Tools

Applications are provided for the vendor and end user:

- Sheriff Licence Key Generator (SlsGen.exe)
- Licence Administrator (SlsAdmin.exe).

The Licence Key Generator is used by software vendors to create keys for their customers and logs all the licence keys issued together with user information.

The Sheriff Licence Administrator enables end users to manage the licences of their Sheriff-protected applications, which may come from different publishers. The Licence Administrator can also be used to monitor licence usage on a network, such as who is logged on and what licence is being used, as well as the current state of licences, the renewal of licences and the import/export of licences.

Note that the full functionality of these applications is replicated in the Sheriff API (via SlsApi and SlsApiEx) enabling a very high degree of customisation.

7. Internet Software Registration

Sheriff ISR enables you to automate your licence key distribution via the Internet using Microsoft Internet Information Server (IIS).

In an Internet Software Registration system, three main functions that can be automated:

1. Reference Code generation
2. Licence Key generation
3. Licence Key authorisation

For further details see topic *Sheriff ISR*

3.2 Architecture

Sheriff adopts a client-server architecture:

1. Client. This is the application you wish to protect. Client interacts with the licence server through API functions.

2. Server. This is the Sheriff DLL that provides licensing services through its API functions.

3. Registry. Sheriff utilises the Windows Registry to keep track of various licences and the location of licence databases on the user's machine.

4. Sheriff Licence Database. This maintains information such as licence policy, licence usage and user's activities.

When an application executes it provides Sheriff with information such as the Product ID and user name, which Sheriff validates against its licence database. If validation is successful the application runs normally, if unsuccessful an application can either exit or continue running in 'demo' mode.

While an application is running it periodically communicates with Sheriff to ensure the currency of the database; this is often referred to as a 'heartbeating'. Before an application exits it notifies Sheriff to free-up its resources, making them available for the next user. Should an application terminate abnormally Sheriff automatically reclaims the licence.

3.3 Security

1. Encrypted Licence Key
2. Encrypted Licence Database
3. Encrypted and Challenged Communication Channels
4. Anti-tampering Measures
5. Anti-debug Measures
6. Protected Licence Key Generator with Secret Codes

---

1. Encrypted Licence Key

The Sheriff Licence Key is dynamically encrypted, which means that the Licence Key changes every time it is generated even with the same Reference Code and licence policies. This is to prevent the Licence Key from being dissected. The Licence Key as well as the Reference Code are self-checked; any bit change to them causes them to become invalid.

2. Encrypted Licence database

The Sheriff licence database is encrypted and cannot be modified.

3. Encrypted and Challenged Communication Channels

The communication channel between an application and Sheriff is encrypted to prevent parameters being monitored or altered en-route. In addition, Sheriff employs a challenge/response protocol in calls between the client and the server. It provides a very reliable way for Sheriff and an application to verify that the other is a legitimate party.

4. Anti-tampering Measures

Comprehensive anti-tampering facilities are built into Sheriff, including:

1. System clock verification: If a licence is *time metered* or *expiry date controlled*, Sheriff has built-in measures to prevent the manipulation of the system clock being used to gain extra time (E.g. extra days cannot be gained if the clock is wound back).
2. If a licence is *time metered* or *unit metered*, Sheriff has built-in measures to prevent the licence from being backed up and restored in order to gain extra time or units.
3. Hardware parameters cannot be emulated to duplicate licence for unauthorised machines.

5. Anti-debug Measures

Built-in anti-debug facilities prevent Sheriff codes from being traced and dissected.

6. Protected Licence Key Generator with Secret Codes

The Sheriff Licence Key Generator cannot be used to generate unauthorised Licence Keys. To generate Licence Keys for a product, a set of Secret Codes is required and each product has its own set of Secret Codes.

3.4 Networking

Overview

Sheriff is designed to protect applications running on networks. In particular, the concurrency limit is used to control the maximum number of users that can simultaneously access a Sheriff-protected application. The network can be a local or wide area network, which can span time zones. The network protocol is of no importance.

Use SlsGen to Define the Licence

Whether or not a Sheriff-protected application can be run over a network will depend upon the type of licence policy that was defined using the Sheriff Licence Generator (SlsGen).

- A *standalone* licence is a licence for a single user. The protected application can only be run on the machine on which the licence is installed and cannot be shared by network users on different machines. A standalone licence cannot be shared by two operating systems running on a dual-boot PC.

- A *network* licence permits more than a single user and its licence database is usually located in a shared network file server. The maximum number of concurrent users is defined using SlsGen.

Installing Sheriff on a Network

To install Sheriff on a network, you need to have only one licence database installed; typically this will be on a file server that is accessible to all of the users who want to run the protected application.

Files

You need to install the following files on the server:

\\BIN\SlsLocal.dll
\\BIN\SlsServer.exe
\\BIN\SlsService.exe

You may install those files at any location on the server machine. A file "slsService_legacy.exe" is provided for versions of Windows prior to XP/Server 2003. Please rename to "slsService.exe".

SlsServer

The server process is called SlsServer. If you are only running a single Sheriff-protected product on the server you can invoke SlsServer from the command line:

[home]\SlsServer.exe /port:8080 /pid:9758-3050-1918-9292-6466 /pn:Sheriff Demo /pp:C:\Sheriff

[home] is the path where SlsServer resides (specify if executed from different folder)
"port" parameter set the TCP/IP port that Sheriff server will be listening to
"pid" is your Product ID
"pn" is your Product Name
"pp" is your Product Licence Path

All of the above parameters are optional. If "port" is not specified, Sheriff Server will listen to port 8080. If "pid", "pn" and "pp" are not specified then Sheriff Server will find product info from the registry.

Alternatively, instead of registering your product using "pid", "pn" and "pp" command line parameters, you can register your product by creating a file called "SlsServer.ini" and placing it in the same folder as SlsServer.exe. If you have more than one Sheriff-protected product, then creating the SlsServer.ini file is mandatory.

The following is an example of SlsServer.ini

```
[Settings]
Products=2
[Product0]
ID=9758-3050-1918-9292-6466
NAME=Demo 1
PATH=C:\Demo
[Product1]
ID=5359-8631-2629-7641-5701
NAME=Demo 2
PATH=C:\Demo
```

Where, "Products" in the "Settings" section specifies the total number of products and within each product section there are three variables, namely:

ID:	Product ID
NAME:	Product Name
PATH:	Product Licence Path

SlsService

It is usually convenient to run SlsServer as a service. SlsService can run any executable as a service -- if you just want to run SlsServer then leave Processes=1. SlsService starts the server via SlsServer.ini, where it contains the command line parameters:

CommandLine = [home]\SlsServer.exe /port:8080 /pid:9758-3050-1918-9292-6466 /pn:Sheriff Demo /pp:C:\Sheriff

[home] is the path where SlsServer resides (specify if executed from different folder)
"port" parameter set the TCP/IP port that Sheriff server will be listening to
"pid" is your Product ID
"pn" is your Product Name
"pp" is your Product Licence Path

To install the service, run the following command:

```
SlsService   -i
```

Licensing the Server

To install a licence on the server, you can either run SlsAdmin on the server machine, or you can call SLS_SetLicence from your own application running on any machine that is connected to the Sheriff Server

Upgrading Notes

1. SlsClock is now obsolete. It is no longer provided.
2. Like the standalone licence, a network licence is now locked to the server machine, not just the hard disk, making it more secure.
3. Connecting to a network licence via a shared drive/folder is no longer supported.

Once a licence is installed on a network drive, applications running on workstations should be *registered*. Registering a product creates the appropriate key in the Windows Registry. The key contains important information about the licence file i.e. its name and location. Registration can either be accomplished programmatically or else by using the Sheriff Administrator (SlsAdmin).

Networking environments can be highly complex and it is very desirable to allocate one machine - probably the server - for licence administration. Thus, if you intend to issue licence keys from a machine on the network you would install SlsGen on this *administrator* machine and also use the same machine to manage the licence keys with SlsAdmin - see below.

Use SlsAdmin to Manage Network Licences

Use the *SlsAdmin* application to monitor active programs and which active users have access to them. You can also suspend an active user at any time although normally you will only do this you when you have reached the limit of authorised concurrent licences and you wish to grant access to another user.

Licence Reclaim

If a networked application terminates abnormally without notifying Sheriff, its licence also dies.

Sheriff will automatically reclaim the dead licence after the expiration of a predefined 'reclaim time' (which must be longer than the 'heart-beat' time). Once a licence has been reclaimed by Sheriff it can be claimed by another network user, however should the terminated application request a licence within the 'reclaim time' Sheriff will revive the dead licence immediately for that application.

3.5 How to Generate a Licence

You generate licences for your customers using the Sheriff Licence Key Generator (SlsGen); full details of how to do this are given in the topic. Below is a brief guide to the principles involved.

Overview

To enable a Sheriff-protected application the end user will require a Licence Key from the publisher. To generate the Licence Key the publisher needs the user's unique Reference Code, which is usually displayed when the application is installed. The Reference Code and Licence Key can be exchanged by phone, fax, email. Alternatively, an Internet-based system can be implemented using Sheriff ISR. Reference Codes are also generated when licences are upgraded or when new licences are purchased.

The publisher enters the Reference Code into the Sheriff Licence Key Generator and chooses the appropriate features for the Licence Policy (E.g. the licence to expire on a certain date). The Sheriff Licence Generator then generates a unique one-time Licence Key for the user.

Points to Note

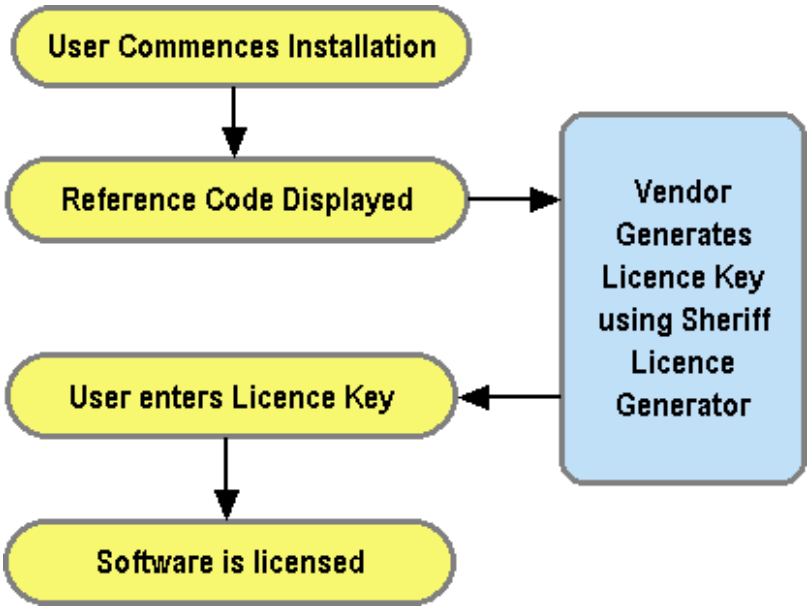
- Once the Reference Code has been generated it is valid until the corresponding Licence Key is received and entered, allowing publishers and end to users to work in different time zones. This also gives publishers the option to check the details of a new customer before releasing the Licence Key.
- Each pair of Reference Codes and Licence Keys is unique and machine-specific. The Reference Code cannot be regenerated; the Licence Key cannot be reused on the original machine or used on any other machine.
- When using Expiry Date metering it is possible to generate a 'Reusable Reference Code'. This removes the need for contact with the user since one Reference Code, within certain limitations, lasts indefinitely. Using this option you can generate new expiry-dated licence keys as needed.
- To uninstall Sheriff there are two possibilities:

    1. You can terminate the user's licence, using the *terminate* function. This can be implemented by the user without reference to the publisher but still prevents the user from re-installing a demo application, for example.

    2. You can remove the user's licence, using the *Remove* function. This completely uninstalls Sheriff from the user's machine. This function is normally password protected i.e. the user needs to obtain a password from the publisher, otherwise he/she might re-install a trial or demo application.

       Both functions can be implemented via the Sheriff API and are also available in SlsAdmin.

Summary



3.6 Glossary

Control - Concurrent User

Defines the maximum number of network users that can access the product simultaneously. The identity of those users can be controlled from the Sheriff Licence Administrator (SlsAdmin.exe)

Control - Expiry

Defines the date on which a licence expires.

Control - Feature Access

Defines which features of a product are enabled.

EULA

End User Licence Agreement. The agreement between software publishers and their customers.

ISR

Internet Software Registration. Enables licence keys to be distributed automatically from a web site

Licence Administrator (SlsAdmin.exe)

A Sheriff application that enables end users to manage their software licences (Eg. to *Import* a licence from another computer or to monitor *concurrent users*).

Licence Import

A means of importing a licence from another computer. See the Sheriff Licence Administrator help for details.

Licence Export

A means of exporting a licence to another computer. This feature can be disabled in Options i.e. you can disable the user's ability to export a licence key to another machine, so locking the licence to one machine only. See the Sheriff Licence Administrator help for details of how to export a licence key.

Licence Database

Collection of three files installed on the end user's machine, including the *Licence File*.

Licence File

File created on the end user's machine that contains the *Licence Policy.* The file must be present on the end user's machine for a Sheriff-protected application to execute.

Licence Key

The code issued to the end user by the publisher, and embedded in the*Licence File,* that enables a Sheriff-protected application to execute in accordance with the terms of the *Licence Policy.*

Licence Key Generator (SlsGen.exe)

Sheriff application used by publishers to generate *Licence Keys.*

Licence Policy

The features and limitations of the end user's licence as defined by the publisher and passed to the user in the *Licence Key.*

Log File

Details of all licences issued are appended to the log (SlsGen.log). Comments may be added to the log directly from the Licence Generator.

Metering - Unit

Method of limiting the number of times a Sheriff-protected application is executed by reference to the consumption of units that are defined by the publisher.

Metering - Day

Method of limiting the number of days in which a Sheriff-protected application is executed.

Portable Licences

A licence can be imported to another machine using the Sheriff Licence Administrator's *Import* facility.

Product ID

A unique code used by the publisher to identify a Sheriff-protected product and distinguish it from other Sheriff-protected products. A unique Product ID is generated from a *Serial Number*.

Product ID & Secret Codes Generator (SlsPsn.exe)

Enables the publisher to generate a unique *Product ID* and *Secret Codes* from a *Serial Number*.

Publisher Data

Supplements the Licence Key. The Licence Key is compulsory whereas Publisher Data is optional. If a publisher feels that the Licence Key does not provide enough features, the Licence Key can be supplemented by Publisher Data. Publisher data is saved with the Licence Key and is in effect an an extension to the Feature Access Key facility. Publisher data occupies a space up to 32 bytes; the format and meaning of the data is up to the publisher.

Publisher Data can be entered or displayed via SlsAdmin, where this has been used as the licensing tool. The Publisher Data may be saved or retrieved by using the appropriate Sheriff API functions.

Reference Code

Code provided by the end user when enabling a Sheriff-protected application. Publisher requires Reference Code in order to generate a *Licence Key*. The Code consists of 24 digits. The first 12 digits are the user's machine signature, the next 4 digits are the product signature and the remaining 8 digits are the run-time signature. It is possible to tell whether or not two Reference Codes are generated from the same machine by comparing the machine signatures.

Reusable Key

A *Licence Key* that can be used on the same machine an unlimited number of times. With a Reusable Key the end user can restore the original licence to the same machine if the *LicenceFile* is overwritten or corrupted (not used when day or unit *metering* is implemented). NB. A new key will be required if the hard disk is reformatted.

Reusable Reference Code

This option is only available when Expiry Date metering has been selected. It is used when you want to be able to extend the date on which the licence will expire without first contacting the user for a new Reference Code.

Secret Codes - Standard

Secret Codes are required to generate unique *Licence Keys* for a product. The Secret Codes are entered into the *Sheriff Licence Key Generator* and enable a publisher to generate *Licence Keys* for his product alone.

Secret Codes - Encrypted

As additional protection against hacking the publisher may prefer to embed encrypted Secret Codes in his product (the *Standard Secret Codes* are entered into the *Sheriff Licence Key Generator*).

Serial Number

A unique number issued to the publisher to enable the generation of a *Product ID* and *Secret Codes* (using the *Product ID & Secret Codes Generator)*.

Status Code

The Verify Status Code feature enables a publisher to check the current status of a user's licence. In the Administrator, when a licence is being registered/authorised, a message box underneath the Reference Code shows the current (or, if not available, previous) licence status. To verify the Status Code, the publisher requests the user to provide that code and then, in the Licence Generator, selects Tools|Verify Status Code and keys in the code. The licence status is displayed.

System Clock Verification

A feature of Sheriff's anti-tampering facilities. Ensures that *day metering* & *expiry date control* cannot be bypassed.

Termination Code

When a user manually terminates a licence using the *Sheriff Licence Administrator* a Termination Code is displayed. The publisher can verify this code using the *Sheriff Licence Key Generator.*

4.1.1 Introduction

>> The Sheriff API is provided as Windows Dynamic Link Libraries; Sheriff can also be implemented as an ActiveX control. For VC++ applications.

>> Any Window programming language capable of calling a standard Windows DLL can work with the Sheriff API.

>> The Sheriff SDK provides direct support for Visual C++ and Visual Basic with API interfaces and API classes.

>> For simplicity, API functions for C++ are used throughout, unless it is explicitly stated otherwise. However API interfaces and classes in Visual Basic are very similar to their C++ counterparts.

>> The API provides both Standard and Extended Libraries:

- The Standard Library offers all the functionality you need to protect your application.
- The Extended Library provides the functionality of the Sheriff Licence Generator (SlsGen) application. This enables you to create your own customised application for issuing licence keys E.g. if you want to distribute licence keys via a distributor, you can create your own application for doing this.

>> A detailed list of API functions is given in the Programming Reference topic for each library. The `next topic` provides a detailed list of all the files in the SDK.

4.1.2 SDK Components

Reference	
SDK Documentation	\
Release notes	Readme.txt
Windows HTML Help File	Sheriff_3.chm
Help in PDF format	Sheriff_3.pdf
Sheriff API	
Sheriff ActiveX API	**\SDK\API\ActiveX**
Local ActiveX Component (for local application)	SlsLocalCom.dll
Remote ActiveX Component (for client of a remote application)	SlsRemoteCom.dll
Sheriff DLL	**SDK\API\SDK\DLL**
Windows DLL (for local application)	SlsLocal.dll
Link library for DLL	SlsLocal.lib
Windows DLL (for remote application)	SlsRemote.dll
Link library for DLL	SlsRemote.lib
VB Includes: Local	**SDK\API\INC**
Sheriff VB class	SCSheriff.cls
Sheriff VB header	SlsApi.bas
VC Includes: Local	**SDK\API\INC**
API Constant Definition	Licdefs.h
Header for Other C++ Compilers	Licnonvc.h
API Error Codes	Licscode.h
API Data Structure	Lictype.h
C++ API Class Implementation	Sheriff.cpp
C++ API Class Header	Sheriff.h
C++ API Interface	SlsApi.h
VB Includes: Remote	**SDK\API\INC**
Sheriff VB class	SCSheriff.cls
Sheriff VB header	SlsApi.bas
VC Includes: Remote	**SDK\API\INC**
API Constant Definition	Licdefs.h
Header for Other C++ Compilers	Licnonvc.h
API Error Codes	Licscode.h
API Data Structure	Lictype.h

C++ API Class Implementation	Sheriff.cpp
C++ API Class Header	Sheriff.h
C++ API Interface	SlsApi.h
Sheriff Extended API	
Sheriff Extended DLL	**SDK\API (Extended)**
Sheriff Extended ActiveX Control	SlsLocalComEx.dll
Sheriff Extended DLL	SlsLocalEx.dll
VB Extended API Interface	SlsApiEx.bas
Header file for Extended Library	SlsApiEx.h
Sheriff Demo	
Sheriff Demo	**SDK\DEMO**
Sheriff Demo: ActiveX	DEMO\ActiveX
Sheriff Demo: Extended API	DEMO\API (Extended)
Sheriff Demo: DLL	DEMO\DLL
Sheriff ISR (Internet Software Registration)	
Sheriff ISR	**SDK\ISR**
Sample Page	Register2.asp
Sample Page	RegisterForm2.asp
Client Side ActiveX Control for Sheriff ISR	SlsLocalComNet.dll
Sheriff Server	
Sheriff Server	**SDK\SERVER**
Sheriff Server	SlsServer.exe
Service for running Sheriff Server	SlsService.exe
Configuration file for SlsService	SlsService.ini
Tools	
Tools	**SDK\TOOLS**
Administrator	SlsAdmin.exe
Administrator Help	SlsAdmin.chm
Licence Key Generator	SlsGen.exe
Licence Key Generator Help	SlsGen.chm
Product ID & Secret Codes Generator	SlsPsn.exe
Product ID & Secret Codes Generator Help	SlsPsn.chm

4.2.1.1 Quick Start

Protecting an application with Sheriff requires four steps:

1. Product Identification
2. Product Registration
3. Licence Authorisation
4. Runtime Implementation

These steps are explained in detail in topics that follow.

4.2.1.2 Identifying a Product

Overview

A Product ID and set of Secret Codes will be used in the Sheriff Licence Key Generator application (SlsGen) and also in your code.

Please note that:

- Sheriff identifies each application by its unique Product ID.
- Secret Codes ensure the one-one else can issue licence keys for your product.
- You should not use the evaluation Product ID and Secret Codes with your released product - they are used by everyone who is evaluating Sheriff.
- When you purchase Sheriff you will be sent a unique Serial Number with which to generate your own Product ID and Secret Codes, using the Sheriff Product ID & Serial Number Generator (SlsPsn).
- When you have obtained a unique Product Serial Number you should keep it secret.

Evaluation Product ID and Secret Codes

If you are using Sheriff for evaluation purposes, you can use the following evaluation Product ID and either the plain or encrypted Secret Codes:

Product ID

```
9758-3050-1918-9292-6466
```

Plain Secret Codes

```
0763-1985-3207-6621
1854-2076-4198-7532
2482-1593-2604-4927
3739-0628-9517-9618
```

Encrypted Secret Codes

```
SLS_SECRET   g_arySecrets[]=
{
    {0x76,0x3E,0x6D,0x36,0x14,0x14,0x6D,0xB8,0x1D,0xCA,0xF3,0xD5,0x34,0x2D,0x08,0xBC,0x14,0x31},
    {0x00,0xB7,0x17,0x00,0x43,0xFC,0xC0,0x7F,0x8F,0xA2,0x22,0xD2,0x1B,0x62,0xB3,0xDE,0x95,0x3F},
    {0xAC,0x2E,0x39,0x6F,0x45,0xD5,0x41,0x11,0x43,0x41,0x2B,0x75,0x9C,0xB2,0x67,0x6E,0x14,0x02},
    {0x79,0x52,0x9A,0x49,0x97,0x8B,0x1C,0x15,0x64,0xB7,0xBE,0x6C,0x7F,0x18,0xC3,0x4E,0x78,0x87},

};
```

4.2.1.3 Registering a Product

First of all, you need to register your application on the end user's machine. Product registration requires the Product ID, Product Name and Licence Path. Products are distinguished by their Product IDs - each product has a unique ID. The product name is only used for administration purposes, for example the Sheriff Administrator application (SlsAdmin.exe) displays the names of products currently running in its 'monitor' window. The licence path tells Sheriff where the licence files reside.

Once a product is registered this information will be kept in Windows Registry in the following location:

HKEY_LOCAL_MACHINE\SOFTWARE\ACUDATA\SHERIFF\PRODUCTID

Sheriff uses the Windows Registry to keep track of which products are registered on the machine and where their licences reside.

There are two ways of registering a product, manually or programmatically:

1. To register your product manually, end users will have to run the Sheriff Administrator (SlsAdmin.exe) to register your product. Please refer to SlsAdmin online help for details.

2. To register your product programmatically, call **`SLS_Register`**. This step is typically carried out by your installation program, although you can register your product from your main application before you call other API functions.

Once your product is registered on the machine you do not need to repeat it, although there is no harm in repeating a registration.

4.2.1.4 Licensing a Product

After your product is registered on the end user's machine you need to license the product by issuing the user with a Licence Key.

The end user exchanges a Reference Code for the Licence Key. The Reference Code is unique to your product and the machine it is installed on and consists of 24 digits, with the first 12 digits being the machine signature, the next 4 digits being the product signature and the last 8 digits being the run-time signature.

There are two ways of authorising a licence: by using the Sheriff Administrator or by using your application:

1. The end user will run Sheriff Administrator to generate a one-time user Reference Code and give it to you in exchange for the Licence Key. You use the Sheriff Licence Key Generator (SlsGen) to generate a Licence Key for the user, who enters the Licence Key in the Sheriff Administrator (SlsAdmin). Details can be found in the help topics for SlsAdmin and SlsGen.

2. If you wish to present your own authorisation dialog boxes in order to keep a consistent 'look and feel' - and to make the authorisation procedure more convenient for your users - you will need to call two API functions, **`SLS_GetReference`** and **`SLS_SetLicence`**.

   `SLS_GetReference` is called to generate the user's Reference Code. Once the Reference Code has been exchanged for a Licence Key then `SLS_Licence` is called to authorise the Licence Key.

   Authorisation should take place the first time the user installs your application or any time your application detects that there is either no licence on the machine or an existing licence has become invalid (E.g. because it has expired). When your application calls **`SLS_Request`** to request a valid licence, `SLS_Request` returns an error if the licence requested is invalid or undefined.

   You can also call **`SLS_QueryLicenceInfo`** to obtain the current state of a licence including detailed licence policies and licence usage such as the number of days left or the number of users using the licence.

4.2.1.5 Runtime Implementation

To protect your application with Sheriff you need to implement three functions: Requesting, Updating and Releasing. Each function only involves one API call. NB. For detailed information about all of the API functions click here .

Requesting

In your application, the first thing you want to do is to check that your licence is still valid. In other words, you need to request Sheriff to grant permission to run your application in the appropriate mode. **`SLS_Request`** is the function to call, it will return with various error codes if Sheriff detects that there is no valid licence on the machine. It is up to your application to take appropriate action should this happen. The demo application provides some examples.

Updating

While your application is running you need to periodically call **`SLS_Update`** to update Sheriff with the latest licence usage information, Sheriff will also update your application with the latest licence state. Updating performs two important tasks:

- Sheriff needs to check whether the application is still up and running in order to decide whether an allocated licence resource should be reclaimed.
- Updating ensures that the licence is valid throughout the time the application is running. This is particularly important if unit metering is being used.

Updating is often referred as 'heartbeating', to make client and server heatbeat at the same pace. As explained, updating is related to licence reclaiming. If a running application fails to update Sheriff within the predefined reclaim time, Sheriff will decide that the application is dead (E.g. crashed or deadlocked). By default, the reclaim time is set to 15 minutes; you can set your own reclaim time by calling **`SLS_SetOptions`**.

Therefore the updating interval should be shorter than the reclaim time. Typically you will set up a timer in your application and call SLS_Update in the timer message handler.

Releasing

Before your application exits, you need to call **`SLS_Release`** to notify Sheriff to release the licence resource so that it will be ready for the next application. Failing to call SLS_Release will result in a dead licence and there may be some delay before it can be automatically reclaimed by Sheriff.

4.2.1.6 Demo Application

The demo is a small sample application to show how a program may be licensed.

- If it is not already present, you will need to copy SlsApi.dll from the API folder into the **\DEMO APP** folder or to a folder in the system path.

- Run SlsDemo and, when prompted for a licence key, generate one with the Sheriff Licence Generator (SlsGen) located in the **\SYSTEM APPS** folder.

  You will need to register the demo app in the Licence Generator using the following:

  Name: Sheriff Demo (or anything else you like)

  Product ID: 9758-3050-1918-9292-6466

  Secret Code 1: 0763-1985-3207-6621
  Secret Code 2: 1854-2076-4198-7532
  Secret Code 3: 2482-1593-2604-4927
  Secret Code 4: 3739-0628-9517-9618

The Visual C++ source code can be found in the **API\API\DEMO SOURCE\VC** folder; there are also projects for VB, VFP and Delphi.

See topic 4.2.2.5 for a detailed description of implementing the C++ demo.

4.2.2.1 API Challenge

Sheriff employs a so-called *challenge* mechanism to provide extra security. Your application can challenge Sheriff to verify that the it is original and has not been tampered with; Sheriff can also challenge back to verify that the calling application is original and has not been tampered with. Challenge also ensures that the parameters cannot be altered en-route.

Challenge can be enabled to add an extra level of security to the following functions: SLS_License, SLS_Request, SLS_Update and SLS_Release. To enable the challenge mode, you need to set the challenge protocol to **SLS_BASIC_PROTOCOL**. For example:

```
SLS_CHALLENGE Challenge;

Challenge.Protocol=SLS_BASIC_PROTOCOL;

Challenge.Size=sizeof(SLS_CHALLDATA);

Challenge.ChallengeData.SecretIndex=nSecretIndex;

Challenge.ChallengeData.Random=nRandom;
```

If you prefer not to enable challenge mode, all you need to do is to set the challenge protocol to **SLS_NO_PROTOCOL**. But please note that challenge **must** be implemented in calling function SLS_License.

To create a challenge, call **`SLS_CreateChallenge`** with your product Secret Codes and the input data stream; to verify a challenge, call **`SLS_VerifyChallenge`**.

If CSheriff class is used, calling SetSecrets method with your product Secret Codes will automatically enable the challenge mode. Otherwise, it is not enabled.

4.2.2.2 Error Handling

Should any API function call fail, it returns an "error" (or "status") code other than SLS_SUCCESS. A full list of these codes can be found in the topic SlsApi Status Codes. To obtain the error message you call SLS_GetErrorMessage with the code.

4.2.2.3 Customising a Licence Policy

For most applications, the standard licensing options that Sheriff provides should be sufficient. However, some applications might require extended licensing features, such as version control, access level control etc. There are two ways of customising a Sheriff licence policy:

1. Use *Feature Access Control*
2. Use *Publisher Data*

1. Feature Access Control

You may classify your application into various levels of feature and your users only need to pay for the level of feature they want. The feature access level can be set from 0 to 99999.

Apart from using the access level to control features, it is possible to use it for controlling other aspects of your application; for instance, you may use it to control your application's version. Alternatively, you may use it to control a combination of features, using the first two digits for controlling your version number and the remaining three digits for controlling access level, for instance.

Feature access level information is embedded in the standard licence key. There is one API function that is related to this, namely SLS_QueryLicence, which is used to retrieve the access key

2. Publisher Data

Publisher Data is designed for customising your licence policy and is kept together with the product's licence key in the licence file. Publisher Data occupies a space up to 32 bytes; the format and meaning of the data is up to the publisher. Publisher Data is stored in an encrypted format.

Publisher Data may be set or retrieved by using the appropriate Sheriff API functions: Publisher Data is set by calling SLS_SetPublisherData and retrieved by calling SLS_GetPublisherData. Therefore, although Publisher Data is stored with the Licence Key it does not form part of Licence Key string and is not passed back to the client with the Licence Key. Publisher Data has to be keyed in by the user either via SlsAdmin or the publisher's application i.e. Publisher Data can be entered or displayed via SlsAdmin, where this has been used as the licensing tool.

4.2.2.4 Trials & Demos

Overview

In this topic we refer to *trials* and *demos*: a *trial* is limited by time while a *demo* is limited in features; these can be combined to give a *time-limited demo*.

Typically, Sheriff-protected applications run in trial or demo mode if there is no valid licence present on the end user's machine.

Trials

There are two kinds of time limitation, one is to limit the number of days starting from the date of installation, another is to set an expiry date. With Sheriff you can use Day Metering and/or Expiry Control to make a trial application. Once the trial period has expired and an end user wishes to continue using your application, they will need to extend the demo or obtain a full licence.

Demos

Demo applications offer limited features to end users. With Sheriff, a demo application can be a full-featured application yet only allow limited access. With *feature access control* you can make your demo application more realistic by allowing different users to have access to different levels of features in your application.

Distribution

Initially, there is the issue of distributing a trial Licence Key. As discussed elsewhere, distribution involves the exchange of a user's Reference Code and the publisher's Licence Key, which can be done via phone calls, faxes, emails or the Web. However, if you do not want to get involved in the distribution of trial Licence Keys you can automate this procedure by calling a couple of Sheriff API functions. The function to issue a licence key without intervention between users and publishers is **`SLS_License`**. Challenge is required to call this function otherwise it would be subject to abuse.

Another issue is how to prevent the application from being re-installed after the trial period in the attempt to gain extra time, which can be done by calling **`SLS_IsProductInstalled`**. In your application, you need to call SLS_IsProductInstalled before you call SLS_License.

SLS_IsProductInstalled detects whether a specified product has been previously installed, however this does not prevent a product from being installed on another machine.

To prevent users from installing a trial on more than one machine, yet at the same time without the need to manually issue a trial Licence Key, the ideal solution is to set up a Web page to distribute the trial Licence Keys (see topic *Sheriff ISR*).

4.2.3.1 About the SlsAPI

The Sheriff Licence System provides a management layer that can track the rights to use purchased software. You can incorporate licence enforcement and verification in your product by using the Sheriff Licence System Application Programming Interface (SLSAPI), a set of functions that provides licensing services within applications. SLSAPI-enabled applications can gain access to the SLSAPI interface through the protected dynamic-link library SLSAPI.DLL.

The SLSAPI-compliant product includes the following basic components:

- An SLSAPI-enabled desktop application
- The dynamic-link library SLSAPI.DLL
- A Sheriff licence file that serves digital licence certificates

Alphabetical List of SLSAPI Functions

SLS_Connect	Connects to a Sheriff network server
SLS_CreateChallenge	Creates Challenge
SLS_ExportLicence	Export licence
SLS_GetErrorMessage	Returns the message string associated with a licensing service function status code.
SLS_GetPublisherData	Enables publishers to retrieve their own data that is stored with the Licence Key
SLS_GetReference	Returns licence reference code
SLS_GetStatusCode	Get current licence status
SLS_GetUserCount	Returns the total number of users who are using the product
SLS_GetVersion	Get the current version numbers of the API libraries
SLS_ImportLicence	Import licence
SLS_IsAdminAccount	Checks whether the current user account is an admin account.
SLS_IsLimitedAccount	Checks whether the current user account is a limited account.
SLS_IsProductInstalled	Checks if the specified product has been installed on the machine
SLS_IsProductLicensed	Checks if the specified product has been licensed on the machine
SLS_License	Issues licensing policy programmatically
SLS_MoveLicence	Move licence
SLS_QueryLicenceInfo	Returns information about licence policy and licence usage
SLS_QueryUserInfo	Returns information about the users who are using the specified product
SLS_Register	Registers licence system in the user's PC
SLS_Release	Requests that the licence system release the licensing resources associated with a specific licence context (compulsory).
SLS_Remove	Removes the specified licence
SLS_RemoveEx	Like SLS_Remove but with added security
SLS_Request	Requests that the licence system grants the licensing resources so the calling application can execute (compulsory).
SLS_RequestEx	Extended version of SLS_Request with extra parameter dwUserID.
SLS_SetLicence	Issues licence with licence key
SLS_SetOptions	Sets various options such as reclaim times

SLS_SetOptions	Sets various options such as reclaim times
SLS_SetPermissions	Sets write access permission to all relevant registry keys and files.
SLS_SetPublisherData	Enables publishers to store their own data with the Licence Key
SLS_SetSessionOptions	Enables publishers to relax the Sheriff machine locking when desired.
SLS_Terminate	Terminates the specified licence
SLS_Update	Updates the synchronisation between the licensed application and the licence system (compulsory).
SLS_VerifyChallenge	Verifies Challenge

4.2.3.2 Licence Security

Sheriff is the most secure software-based system available. If you incorporate the Sheriff Licence System Application Programming Interface (SLSAPI) functions in your application, you can deter licence system tampering and reveal when tampering occurs.

The security model employed by Sheriff aims to stop an intruder's attack in all areas. The model comprises of three components: encrypted licence key, encrypted licence file and encrypted Application-SLSAPI communication channel.

Encrypted Application-SLSAPI communication channel is accomplished by including a challenge/response protocol in the calls to the SLSAPI functions. It provides a very reliable way for both the licence system and the application to verify that the other is a legitimate party. It also provides a reliable encryption system to prevent the communication parameter between the application and the API from being modified en-route.

The SLSAPI functions use the MD5 Message-Digest Algorithm to implement the challenge/response protocol. This algorithm does not require patented cryptographic techniques. Knowledge of the algorithm neither compromises the secrets the application passes through it nor the level of security the SLSAPI offers.

4.2.3.3 Licence Policy

With Sheriff you can combine the following features in your licensing policy:

1. Concurrency
2. Expiration
3. Day Metering
4. Unit Metering
5. Feature Access Control

Concurrency Limiting

This implementation limits the maximum number of concurrent users to that permitted by the licence agreement (EULA).

Licence Metering

Sheriff provides two types of metering:

1. Day Metering enables the software to run to a specified date (E.g. until 31/8/97) or for a specified number of days (E.g. 21 days).
2. Unit Metering provides a flexible method for the software publisher to determine which events will be chargeable. Each chargeable event is known as a 'unit' (E.g. an execution of the software, viewing/printing a document, or any specified period of time). So the publisher might permit the software to be executed ten times only, this would be the equivalent of providing ten metered units.

Feature Control

Software publishers can distribute full versions of their software, but limit end users' access to software features. End users can then buy what they need. Using 'feature control', software publishers can license their software as shareware, demo software or buy-as-you-need.

4.2.3.4 Licensing Strategies

Use the SLSAPI functions to obtain authorisation from the licence system for your application to run. If a valid licence is not available, your application can be customised to take appropriate action:

- *Demonstration Mode.* If there is no licence or licence system, the software can display a warning message but continue to run in the demonstration mode E.g. with some features disabled or for a limited period of time.
- *Authorisation Desired.* If there is no licence or licence system, the software can display a warning message but continue to run.
- *Authorisation Required.* If no valid licence is available, the software will not run.

SlsDemo.exe demonstrates how to implement these basic licensing strategies.

4.2.3.5 SLSAPI Integration

This section provides an example of how to incorporate the Sheriff Licence System Application Programming Interface (SLSAPI) functions in your application, with and without challenge protocol.

Before you begin coding your application, you will need to obtain your Product ID and Secret Codes, either by purchasing a Serial Number or using the evaluation versions. You also need the Sheriff SDK which includes the dynamic link library SLSAPI.DLL and source code for implementing the challenge/response in your application.

To code the application

1. Choose and implement your licensing strategy. For examples, see "Licensing Strategies".
2. Incorporate the source codes provided in SDK to prepare the challenge.
3. Add calls to the **SLS_Request** functions in your application to grant licence resources.
4. Include periodic calls to the **SLS_Update** function to notify the licence system that the current licence is still being used.
5. Incorporate code to handle errors. Call the **SLS_GetErrorMessage** function to return a message string that describes the error.
6. Call **SLS_Release** to release licence resources before exiting from the application.

4.2.3.6 SlsAPI Data Types & Structures

SLSAPI Data Types

```
typedef DWORD SLS_HANDLE;
```

Defines the handle to the licensing context used by the licence service function calls.

SLSAPI Data Structures

Licence Date

```
typedef struct _SLS_DATE

{
     DWORD Year,
     DWORD Month,
     DWORD Day;
} SLS_DATE;
```

The SLS_DATE defines a date structure that contains calendar year, month and day. It is part of the SLS_LICENCE structure.

Licence

```
typedef struct _SLS_LICENCE
{
     DWORD Type; //type of licence
     DWORD Meter; //for metering, e.g. Unit, Time metering
     SLS_DATE EndDate; //for expiration date
     DWORD CoUsers; //Concurrent users
     DWORD AccessKey; //Access control key
} SLS_LICENCE;
```

The SLS_LICENCE defines Sheriff's licence policy structure. It is used by SLS_License to issue licence programmatically.

Type

Specifies the type of the licence policy. It can be a combination of the following values:

```
//type of licence
#define SLS_TYPE_UNDEFINED 0x0000 //Undefined
#define SLS_TYPE_UNIT_METER 0x0001 //Unit Metering
#define SLS_TYPE_TIME_METER 0x0002 //Time Metering
#define SLS_TYPE_EXPIRATION 0x0004 //Expiry Control
#define SLS_TYPE_CONCURRENCY 0x0008 //Concurrency Control

#define SLS_TYPE_REUSABLE_KEY 0x0100 //Reusable key
#define SLS_TYPE_REUSABLE_REF 0x0200 //Reusable reference
#define SLS_TYPE_UNEXPORTABLE 0x0400 //Export Disabled
#define SLS_TYPE_STANALONE 0x0800 //Standalone
```

Meter

Used only for metering, i.e when **type** is either `SLS_TYPE_UNIT_METER` or `SLS_TYPE_TIME_METER`, to specify the total amount of units or total number of days allowed in the licence policy. Maximum value is 99999.

EndDate

Used only for expiry date licensing, i.e. when **type** is `SLS_TYPE_EXPIRATION`, to specify the expiration date.

CoUsers

Specifies the maximum concurrent users. Maximum value is 99999.

AccessKey

Specifies a key to access application features. A software application can offer many features and these features can be managed in many levels. The software publisher may want to license the software in levels. The **AccessKey** is used by software publishers to control the accessibility of features. The **AccessKey** is defined by software publishers and its meaning is also only interpreted by the publishers. Maximum value is 99999.

Request

```
typedef struct _SLS_REQUEST
{
     DWORD UnitsReserved;
} SLS_REQUEST
```

The SLS_REQUEST is used for requesting a licence to run the application. It is used by **SLS_Request**. When the licensing policy requires unit metering, the data member **UnitsReserved** specifies the number of units to run the application; when the licensing policy does not requires unit metering, the **UnitsReserved** should be set to zero.

```
typedef struct _SLS_PERMIT
{
     DWORD UnitsGranted;
     DWORD AccessKey;
} SLS_PERMIT;
```

The SLS_PERMIT is used for receiving a licensing permit, when **SLS_Request** is called to request a licence to run the application, or when **SLS_Update** is called to update the licence status.

UnitsGranted

Returns the total number of units granted and reserved.

AccessKey

Returns the publisher-defined feature access key.

Update

```
typedef struct _SLS_UPDATE
{
     DWORD UnitsReserved;
     DWORD UnitsConsumed;
} SLS_UPDATE;
```

The SLS_UPDATE is used for updating licence status. It is used by **SLS_Update**.

UnitsReserved

Specifies the total number of units to be reserved. If no additional units are required since the initial call to the **SLS_Request** function or the last call to the **SLS_Update** function, then this parameter should be the current total returned in the **SLS_PERMIT.UnitsGranted** parameter. The total reserved includes the units consumed.

total returned in the parameter. The total reserved includes the units consumed. That is, if an application requests 100 units, and then consumes 20 units, there are still 100 units reserved but only 80 available for consumption.

If additional units are required, the application must calculate a new total for **UnitsReserved**.

The licence system verifies that the requested number of units exist, and it may reserve those units, but these units are not consumed at this time.

This value may be smaller than the original number requested by **SLS_Request** to indicate that fewer units are needed than originally anticipated.

UnitsConsumed

Specifies the total number of units consumed since the initial call to the **SLS_Request** function. If the *UnitsConsumed* exceeds the number of units reserved, the error SLS_E_UNIT_EXCEEDED is returned and the remaining units are consumed.

Release

```
typedef struct SLS_RELEASE
{
     DWORD UnitsConsumed;
} SLS_RELEASE;
```

The SLS_RELEASE is used by **SLS_Release** to report licence usage.

UnitsConsumed

Specifies the total number of units consumed since the initial call to the **SLS_Request** function.

Licence Info

```
typedef struct _SLS_LICENCE_INFO
{
     //Licence Part
     DWORD Type; //type of licence
     DWORD Meter; //Units or Days limit
     SLS_DATE EndDate; //Expiration date
     DWORD CoUsers; //Concurrent users
     DWORD AccessKey; //Access control key
     //Usage Part
     DWORD State; //State of licence
     DWORD MeterUsage; //Units used, days used
     SLS_DATE StartDate; //Start date
     DWORD ActiveUsers; //Number of active users
} SLS_LICENCE_INFO;
```

The SLS_LICENCE_INFO structure is used by **SLS_QueryLicenceInfo** to retrieve the licence policy and its current usage.

Type, Meter, EndDate, CoUsers, AccessKey as explained in SLS_LICENCE.

State

Returns the state of the licence being requested. Its value can be one of the following:

```
#define SLS_STATE_UNDEFINED 0x0000 //Licence is undefined
#define SLS_STATE_BAD 0x0001 //Licence is invalid
#define SLS_STATE_EXPIRED 0x0002 //Licence has expired
#define SLS_STATE_EXHAUSTED 0x0003 //Licence units have run out
```

```
#define SLS_STATE_OK 0x0008 //Licence is valid
```

MeterUsage

Specifies the total number of units or days that have been used since the licence was purchased.

StartDate

Specifies the purchase date of the licence policy.

ActiveUsers

Specifies the number of active users at the time this information is being requested.

Product Info

```
typedef struct _SLS_PRODUCT_INFO
{
     char ProductID[32];
     TCHAR ProductName[128];
     TCHAR LicencePath[MAX_PATH];
} SLS_PRODUCT_INFO;
```

The **SLS_PRODUCT_INFO** structure is used by SLS_QueryProductInfo to retrieve the product information including ProductID, Product Name and its Licence Path.

Challenge

```
typedef struct _SLS_CHALLENGE
{
     DWORD Protocol;
     DWORD Size;
     SLS_CHALLDATA ChallengeData;
} SLS_CHALLENGE;
```

The **SLS_CHALLENGE** structure is used for both the challenge and the response of the **SLS_License, SLS_Request** and **SLS_Update** licence service functions. It is the main structure in the challenge/response mechanism, and it is supported by all challenge/response protocols.

Protocol

Specifies the protocol setting for licence authentication. Currently only the basic protocol SLS_BASIC_PROTOCOL is supported. If challenge is not desired, **Protocol** should be set to SLS_NO_PROTOCOL.

Size

Specifies the size, in bytes, of the **ChallengeData** (**SLS_CHALLDATA)** structure.

ChallengeData

Structure that contains the challenge that the application passes to the licence system and the response the licence system returns to the application.

Remarks

Use the **SLS_CHALLDATA** structure to pass the challenge to the licence system. The licence system also returns the challenge response in the **SLS_CHALLDATA** structure.

Because the **SLS_CHALLDATA** structure can vary depending on the protocol specified in the **Protocol** member, this structure must be a single contiguous entity in memory and must not exceed the number of bytes specified in the **Size** member. It cannot contain any pointers.

SLSAPI passes the **Protocol**, the **Size** of the **SLS_CHALLDATA** structure, and the actual data contained in the structure to the licence system. The licence system, in turn, casts the byte sequence into the appropriate structure based on the **Protocol** specified.

The constant value SLS_BASIC_PROTOCOL specifies a standard basic challenge protocol that is supported by all SLSAPI. When the **Protocol** specified is SLS_NO_PROTOCOL, there is no challenge and no response.

Challenge Data

```
typedef struct _SLS_CHALLDATA
{
     DWORD SecretIndex;
     DWORD Random;
     SLS_MSG_DIGEST MsgDigest;
} SLS_CHALLDATA;
```

The **SLS_CHALLDATA** structure is passed in the **SLS_CHALLENGE** structure. The **SLS_CHALLDATA** structure passes the challenge from the application to the licence system, and passes the response from the licence system back to the application.

SecretIndex

Specifies the index of the secret value to be challenged. The secret index is 1-based.

Random

Specifies a random 32-bit value.

MsgDigest

Structure that contains the message digest that is computed by the MD5 Message-Digest Algorithm from RSA Data Security, Inc.

Remarks

In the basic challenge protocol, the application must choose the index of the secret to be challenged and it must generate a random number. It must then compute a message digest using the MD5 Message-Digest Algorithm. The input to the algorithm is formed by concatenating the input parameters to the function being called, the random number, the index of the secret to be challenged, and the actual secret value. The input parameters are described in the SLSAPI functions. The application then passes the algorithm output to the licence system.

The licence system authenticates the message digest and computes a new message digest consisting of the output parameters, a new random number, a new index of the secret to be challenged, the actual secret value and the returned status. This new message digest is passed back to the application, which, in turn, authenticates it. Note that the actual secret value never passes between the application and the licence system in plain text.

If the function $h(x)$ is the algorithm that, given input x, returns the output of the MD5 Message-Digest Algorithm, then the following briefly illustrates the basic protocol.

The application passes the **SLS_MSG_DIGEST** structure to the licence system:

$h(in + R_{in} + X_{in} + S_{(X)})$

The licence system passes a new **SLS_MSG_DIGEST** to the application:

$h(out + R + X + S_{(X)})$

out out

where:

- in is a byte stream that encodes the input parameters
- R_{in} is the random number generated by the application
- X_{in} is the index of the secret to be challenged chosen by the application
- S indicates a secret
- $S_{(X)}$ is the actual secret value
- + denotes concatenation.

Similarly:

- out is a byte stream that encodes the output parameters
- R_{out} is the random number generated by the licence system
- X_{out} is the index of the secret to be challenged chosen by the licence system

This data format can be invalid if the **Protocol** specified in the **SLS_CHALLENGE** structure is not SLS_BASIC_PROTOCOL. Other protocols may define their own **SLS_CHALLDATA** format. Particularly if the **Protocol** is specified as SLS_NO_PROTOCOL then there will be challenge implemented between the application and the licence system.

Message Digest

```
typedef struct _SLS_MSG_DIGEST
{
     char MessageDigest[16]; //binary data
} SLS_MSG_DIGEST;
```

The **SLS_MSG_DIGEST** structure is passed in the **SLS_CHALLDATA** structure. The MD5 Message-Digest Algorithm from RSA Data Security, Inc., computes the **SLS_MSG_DIGEST** structure by using the following elements as input: the input and output parameters to the **LSRequest** or **LSUpdate** licence service functions; a random number; the index of an application secret; and the actual application secret.

MessageDigest[16]

Specifies a 128-bit message digest that is the output of the MD5 Message-Digest Algorithm.

Licence Options

```
typedef struct _SLS_OPTIONS
{
     DWORD HeartbeatTime; //heartbeat time
     DWORD ReclaimTime; //reclaim time
     DWORD Reserved1; //reserved, should be set to 0
     DWORD Reserved2; //reserved, should be set to 0
} SLS_OPTIONS;
```

The SLS_OPTIONS structure is used by **SLS_SetOptions** to set various licence options at run time. There are two options supported in the current version, MinReclaimTime and MaxReclaimTime.

HeartbeatTime

If the user has not updated its licence status within the time specified by HeartbeatTime, then Sheriff decides that the user has become inactive and its licence is ready to be reclaimed.

ReclaimTime

If the user has not updated its licence status within the time specified by MaxReclaimTime, then Sheriff decides that the user has become inaccessible and its licence will be reclaimed immediately.

User Info

```
typedef   struct   _SLS_USER_INFO
{
          char   UserName[32];
       DWORD  UserID;
        SLS_DATE_TIME  LoginTime;
         SLS_DATE_TIME  LastUpdateTime;
} SLS_USER_INFO;
```

The SLS_USER_INFO structure is used by **SLS_QueryUserInfo** to retrieve user's information

UserName

The name of the user as provided when call SLS_Request or SLS_RequestEx

UserID

The ID of the user as provided when call SLS_RequestEx

LoginTime

The time when SLS_Request or SLS_RequestEx is called.

LastUpdateTime

The time the last SLS_Update is called.

4.2.3.7 Status Codes

Status Code	Meaning
SLS_E_PRODUCT_CODE	Invalid Product Code (Product ID)
SLS_E_REFERENCE_CODE	Invalid Reference Code
SLS_E_MACHINE_CODE	Invalid Machine Signature
SLS_E_SYSTEM_TIME	Invalid System Time
SLS_E_OUT_OF_MEMORY	Not Enough Memory
SLS_E_OUT_OF_HANDLE	Not Enough Handles
SLS_E_CHALLENGE_ILLEGAL	Illegal Challenge
SLS_E_CHALLENGE_UNSUPPORTED	Unsupported challenge
SLS_E_ARGUMENT_INVALID	Invalid argument passed to function
SLS_E_LICENCE_HANDLE	Invalid licence handle
SLS_E_LICENCE_UNREGISTERED	Unregistered licence
SLS_E_LICENCE_UNDEFINED	Licence is undefined
SLS_E_LICENCE_EXCEEDED	Licence concurrency exceeded
SLS_E_LICENCE_EXHAUSTED	Licence meter exhausted
SLS_E_LICENCE_EXPIRED	Licence has expired
SLS_E_LICENCE_INVALID	Licence is invalid, corrupted
SLS_E_LICENCE_SUSPENDED	Licence is suspended
SLS_E_LICENCE_TERMINATED	Licence is terminated
SLS_E_LICENCE_UNAVAILABLE	Licence is unavailable
SLS_E_UNIT_UNDEFINED	Licence unit not defined
SLS_E_UNIT_EXCEEDED	Licence unit shortage (all units has been reserved)
SLS_E_BAD_SECRET	Incorrect Secret Code
SLS_E_BAD_MAC	Incorrect Message Authentication Code
SLS_E_FILE_NOT_FOUND	The file could not be located.
SLS_E_FILE_BAD_PATH	All or part of the path is invalid.
SLS_E_FILE_TOO_MANY	The permitted number of open files was exceeded.
SLS_E_FILE_ACCESS_DENIED	The file could not be accessed
SLS_E_FILE_BAD_HANDLE	There was an attempt to use an invalid file handle.
SLS_E_FILE_BAD_SEEK	There was an error trying to set the file pointer.
SLS_E_FILE_HARDIO	There was a hardware I/O error.
SLS_E_FILE_SHARING	There was a file sharing error
SLS_E_FILE_LOCKING	There was a file locking error
SLS_E_FILE_DISK_FULL	The disk is full.
SLS_E_FILE_EOF	The end of the file was reached.
SLS_E_FILE_BAD_CRC	Licence file CRC error
SLS_E_FILE_BAD_SIGNATURE	Signature error in the file

SLS_E_FILE_BAD_DATA	Data error in the file
SLS_E_PATH_INVALID	Invalid Licence Path
SLS_E_PATH_CREATE	Failed to create licence path
SLS_E_PATH_ACCESS_DENIED	Access (Read/Write) to licence path was denied
SLS_E_REG_FAILURE	Registry operation failure
SLS_E_BAD_LICENCE	Licence File been tampered
SLS_E_BAD_REGISTRY	Registry has been tampered
SLS_E_BAD_MACHINE	Machine Signature is wrong
SLS_E_OP_MACHINE_CODE	Failed to get machine code
SLS_E_OP_MACHINE_SIGN	Failed to get machine signature
SLS_E_OP_FILE_SIGN	Failed to get file signature
SLS_E_OP_DISK_SIGN	Failed to get disk signature
SLS_E_OP_TERMINATION	Failed to terminate licence
SLS_E_LICENCE_BAD_TIME	Incorrect time meter in licence
SLS_E_LICENCE_BAD_UNIT	Incorrect unit meter in licence
SLS_E_LICENCE_BAD_DATE	Incorrect expiration date in licence
SLS_E_LICENCE_BAD_COUSERS	Incorrect concurrent user limit in licence
SLS_E_LICENCE_BAD_ACCESSKEY	Incorrect access key in licence
SLS_E_LICENCE_BAD_LICKEY	Incorrect Licence key
SLS_E_IMPORT_BAD_LICKEY	Incorrect import licence key
SLS_E_IMPORT_BAD_SOURCE	Incorrect import source signature
SLS_E_REINSTATE_BAD_SOURCE	Cannot reinstate licence due to incorrect source licence
SLS_E_EXPORT_NOT_ALLOWED	Licence export not allowed
SLS_E_BAD_USER_INDEX	User Index out of range
SLS_E_UPID_SIGNATURE_REF	Machine Signatures in reference do not match
SLS_E_UMID_SIGNATURE_REF	Product Signatures in reference do not match
SLS_E_UPID_SIGNATURE_KEY	Machine Signatures in licence key do not match
SLS_E_REF_SIGNATURE_KEY	Reference Signatures in licence key do not match

4.2.3.8 Connect

HRESULT SLS_Connect(
LPCSTR lpszServer,
int nPort)

The **`SLS_Connect`** establishes the connection with the Sheriff server.

Parameters

lpszServer

Points to a string that contains the address of the server. This can be in the format of an IP address or a URL, such as 210.88.543.98, http://licensing.yourdomain.com

nPort

The port number that Sheriff Server listens to. By default, Sheriff Server listens to port 8080.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

SLS_Connect is only available in the network version of Sheriff.

4.2.3.9 Create Challenge

HRESULT SLS_CreateChallenge(
SLS_SECRET *pSecretArray,
int nSecrectSize,
const BYTE *pbStream,
int nStreamSize,
SLS_CHALLENGE *pChallenge)

The **`SLS_CreateChallenge`** function creates a challenge from the input data stream.

Parameters

pSecretArray

Points to an array of product secrets.

nSecretSize

Size of the secret array.

pbStream

Points to the buffer of input data stream.

nStreamSize

Size in byte of the input data stream buffer.

Challenge

Points to a challenge structure in which the challenge will be created.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

4.2.3.10 Export Licence

HRESULT WINAPI SLS_ExportLicence(
LPCSTR pszProductID,
LPCSTR pszReference,
SLS_LICENCE *pLicence,
LPSTR pszLicenceKey)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszReference

Points to a string that contains the reference code that is obtained from the PC to which the licence is being exported.

pLicence

Points to a licence policy structure.

pszLicenceKey

Points to a buffer in which the Licence Key is to be placed. The minimum size of the buffer is 64 bytes.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

4.2.3.11 Get Error Message

HRESULT SLS_GetErrorMessage(
HRESULT hr,
LPTSTR pszErrorMessage)

The **`SLS_GetErrorMessage`** returns an error message associated with an error code.

Parameters

hr

Specifies an error status code returned from any SLSAPI function

pszErrorMessage

Points to a buffer allocated by caller to receive the error message specified in the error code *hr*. The minimum length of the buffer is 256 bytes.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

4.2.3.12 Get Publisher Data

HRESULT WINAPI SLS_GetPublisherData(
LPCSTR pszProductID,
LPTSTR lpszPublisherData);

Although Sheriff provides a comprehensive set of licensing features including time/unit metering, expiration control and feature access control, sometimes publishers might want to control their licences in special ways that would be difficult to implement with the standard features - Set/Get Publisher Data is designed for this purpose. Publisher data is simply a space up to 32 bytes that Sheriff uses to keep any data that publishers might specify. The format and meaning of the data is up to the publisher.

Publisher data is kept together with the product's Licence Key in the licence file. It is set by calling **`SLS_SetPublisherData`** and retrieved by calling **`SLS_GetPublisherData`.**

4.2.3.13 Get Reference

HRESULT SLS_GetReference(
LPCSTR pszProductID,
LPSTR pszReference)

The SLS_GetReference queries SLSAPI for the reference code associated with the product.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszReference

Points to a buffer in which the Reference Code is to be placed. The minimum size of the buffer is 32 bytes.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

The application calls the **`SLS_GetReference`** to obtain the reference code of the product installed on the user's PC. With the reference code the user can obtain the licence key from the application publisher. **`SLS_Reference`** is often called by the application that wants to programmatically issue a licence policy from the application, in which case **`SLS_SetLicence`** is called upon when the application has received the licence key from the publisher.

4.2.3.14 Get Licence Status

HRESULT WINAPI SLS_GetStatusCode(
LPCSTR pszProductID,
LPSTR pszStatusCode)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID

pszStatusCode

Points to a buffer in which the Status Code is to be placed. The minimum size of the buffer is 32 bytes.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

Use this function to retrieve the current status of the licence.

An end user can modify a licence policy by exporting keys; this function provides the publisher with a way of checking the current licence status and verifying it. Licence status is returned in the form of 40-digit code.

4.2.3.15 Get User Count

HRESULT WINAPI SLS_GetUserCount(
LPCSTR pszProductID,
DWORD *pdwUserCount);

`SLS_GetUserCount` returns the total number of users who are using the product specified by the **pszProductID**. The user count is returned to the buffer pointed to by **pdwUserCount**.

4.2.3.16 Get Version Numbers

HRESULT WINAPI SLS_GetVersion(
int *pnMajorVersion
int *pnMinorVersion)

This function returns the current version numbers of the API libraries.

The major version number is returned in the pnMajorVersion variable and the minor version number in the pnMinorVersion variable.

Remarks

SLS_Version used to return the Major and Minor version number, such as MajorVersion=2 and MinorVersion=8. From version 2.8.7, the MinorVersion includes the Build version. For instance, version 2.8.7 will be represented as MajorVersion=2, MinorVersion=87

4.2.3.17 Import Licence

HRESULT WINAPI SLS_ImportLicence(
LPCSTR pszProductID,
LPCSTR pszReference,
LPCSTR pszLicenceKey)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszReference

Points to a string that contains the reference code that is obtained from the PC to which the licence is being exported.

pszLicenceKey

Points to a string that contains a exported licence key returned by calling SLS_ExportLicence.

4.2.3.18 Is Admin Account

HRESULT WINAPI SLS_IsAdminAccount()

Return Value

SLS_SUCCESS if the current user account is an administrative account. Otherwise an error status code is returned indicating the cause of the error.

Remarks

See also `IsLimitedAccount` and `SetPermissions`

4.2.3.19 Is Limited Account

HRESULT WINAPI SLS_IsLimitedAccount()

Return Value

SLS_SUCCESS if the current user account is a limited account. Otherwise an error status code is returned indicating the cause of the error.

Remarks

See also `IsAdminAccount` and `SetPermissions`

4.2.3.20 Is Product Installed

HRESULT WINAPI SLS_IsProductInstalled(LPCSTR pszProductID);

IsProductInstalled returns SLS_SUCCESS if Sheriff detects that the specified product has been installed on the PC.

See also **`IsProductLicensed`**. Note that both functions are designed to detect the *history* of the product and licence. A success return does not necessarily mean the current state of the product or licence on the PC. In other words, if the specified product was installed or licensed once on the PC and then terminated or deleted, **`SLS_IsProductInstalled / SLS_IsProductLicensed`** will still succeed.

These functions are typically used in applications to prevent users from re-installing a trial licence.

4.2.3.21 Is Product Licensed

HRESULT WINAPI SLS_IsProductLicensed(LPCSTR pszProductID);

IsProductLicensed returns SLS_SUCCESS if Sheriff detects that the specified product has been licensed on the PC.

See also **`IsProductInstalled`**. Note that both functions are designed to detect the *history* of the product and licence. A success return does not necessarily mean the current state of the product or licence on the PC. In other words, if the specified product was installed or licensed once on the PC and then terminated or deleted, **`SLS_IsProductInstalled`** or **`SLS_IsProductLicensed`** will still succeed.

These functions are typically used in applications to prevent users from re-installing a trial licence.

4.2.3.22 License

HRESULT SLS_License(
LPCSTR pszProductID,
SLS_LICENCE *pLicence,
SLS_CHALLENGE *pChallenge)

The **`SLS_License`** function asks SLSAPI to authorise a licence policy.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pLicence

Points to a licence policy structure.

pChallenge

Points to a challenge structure. The licence system does not challenge back.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

If an application wants to issue a licence policy programmatically without the user's involvement, this is the function to call. This function is typically used by an application's installation program to issue an evaluation licence with a limited licence policy.

The input parameter for computing challenge is the *Licence* structure.

4.2.3.23 Move Licence

HRESULT WINAPI SLS_MoveLicence(
LPCSTR pszProductID,
LPCSTR pszReference,
SLS_LICENCE *pLicence,
LPSTR pszLicenceKey)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszReference

Points to a string that contains the reference code that is obtained from the PC to which the licence is to be moved.

pLicence

Points to a SLS_LICENCE structure to receive licence policies.

pszLicenceKey

Points to a buffer in which the Licence Key is to be placed. The minimum size of the buffer is 64 bytes.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

4.2.3.24 Query Licence Info

HRESULT SLS_QueryLicenceInfo(
LPCSTR pszProductID,
SLS_LICENCE_INFO *pLicenceInfo)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pLicenceInfo

Points to a buffer to receive information about licence policy and usage.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

SLS_E_LICENCE_UNREGISTERED if the licence is not yet registered on the user's PC.

Remarks

The SLS_QueryLicenceInfo returns an SLS_LICENCE_INFO structure data containing licence policy and its current usage. If the functions returns successfully, the state of the licence, i.e. `pLicenceInfo->state` should be further checked.

4.2.3.25 Query Product Info

HRESULT WINAPI SLS_QueryProductInfo(
LPCSTR pszProductID,
SLS_PRODUCT_INFO *pProductInfo);

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pProductInfo

A data structure provided by the caller to receive the Product Info.

Remarks

Use this function to retrieve the licence path of a registered product.

4.2.3.26 Query User Info

HRESULT WINAPI SLS_QueryUserInfo(
LPCSTR pszProductID,
DWORD dwUserIndex,
SLS_USER_INFO *pUserInfo);

`SLS_QueryUserInfo` is called to retrieve information about the users who are using the specified product. **dwUserIndex** is the index number of the user, its value is between 1 and the user count returned by SLS_GetUserCount. ***pUserInfo** points to a SLS_USER_INFO data structure to receive the information.

4.2.3.27 Register Product

HRESULT SLS_Register(
LPCSTR pszProductID,
LPCSTR pszProductName,
LPCSTR pszLicencePath)

The **`SLS_Register`** function registers a product's licence on user's PC.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszProductName

Points to a string that identifies the application's product name.

pszLicencePath

Points to a full path where licence files will be stored.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

The SLS_Register is the first function that needs to be called by the application before any other SLSAPI function can be called. Once a licence is registered successfully on the user's PC, there is no need to call SLS_Register. This function is typically used by an application's installation program to register the application's licence on the user's PC.

Additional Functions

1. SLS_SetPermissions
As Sheriff requires that all users have write access to Sheriff related registry keys and files. This function therefore sets write access for all users on all Sheriff related registry keys and files. Typically, the application calls SLS_SetPermissions after a successful call to SLS_Register.

NB. SLS_SetPermissions is not about setting user account privileges, it is to set all Sheriff related registry keys and file access privileges for other users. In other words, SLS_SetPermissions makes these registry keys and files writable for other users. Only an Admin account can make such changes, so the application first calls SLS_IsAdminAccount to make sure that the current account is an Admin account before calling SLS_SetPermissions.

2. SLS_IsLimitedAccount
This function returns SLS_SUCCESS if the user account is a limited account.

3. SLS_IsAdminAccount
This function returns SLS_SUCCESS if the user account is an admin account.

NB. These three functions only work on Windows NT/2000/XP

4.2.3.28 Release Licence

HRESULT SLS_Release(
LPCSTR pszProductID,
SLS_HANDLE lLicenceHandle,
SLS_RELEASE pRelease,
SLS_CHALLENGE pChallenge)

The **SLS_Release** function requests that the licence system release the licensing resources associated with the licensing context identified by the *lLicenseHandle* parameter.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

lLicenseHandle

Specifies the handle to the licensing context. This parameter must be a handle created with the **SLS_Request** function.

pRelease

Specifies the total licence usage. This parameter is used only if a licence policy is in effect that comprises of unit metering (and if you choose to implement such a licence policy in the application). The *UnitsConsumed* parameter in the *Release* structure specifies the total number of units consumed in this handle context since the initial call to the **SLS_Request** function.

pChallenge

Points to a challenge structure. The challenge response will also be returned in this structure.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

Use the **SLS_Release** function to release licensing resources associated with the licence context identified by the *lLicenseHandle* parameter.

4.2.3.29 Remove Licence

HRESULT WINAPI SLS_Remove(
LPCSTR pszProductID,
DWORD dwOptions,
SLS_CHALLENGE *pChallenge)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

dwOptions

Instructs how which part of the licence system should be removed. It is a combination of the following values:

```
#define SLS_REMOVE_FILES 0x0001 //remove licence files
#define SLS_REMOVE_REGISTRY 0x0002 //remove licence registry
#define SLS_REMOVE_HISTORY 0x0004 //remove licence history
#define SLS_REMOVE_ALL 0x0007 //remove all of above
```

pChallenge

Points to a challenge structure. The licence system does not challenge back.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

Unlike **`SLS_Terminate`** which is designed to enable end users to terminate the licence, **`SLS_Remove`** is designed to enable the software publisher to remove the licence.

In other words, the difference between "Terminate" and "Remove" is that:

1. **`SLS_Terminate`** generates a termination code that the software publisher uses to verify whether the licence has been terminated from the PC, whereas **`SLS_Remove`** does not.

2. **`SLS_Terminate`** only removes the licence files, whereas **`SLS_Remove`** can completely remove all of the licence components including licence files, registry setting and history marks on the PC.

See also **`SLS_GenerateRemovePassword`** in the Extended API.

4.2.3.31 Remove Licence (Extended)

HRESULT WINAPI SLS_RemoveEx(
LPCSTR pszProductID,
DWORD dwOptions,
LPCSTR pszRemoveReference,
LPCSTR pszRemovePassword)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

dwRemoveOption

Specify one or a combination of the following options:

```
#define SLS_REMOVE_FILES 0x0001 //remove licence files
#define SLS_REMOVE_REGISTRY 0x0002 //remove licence registry
#define SLS_REMOVE_HISTORY 0x0004 //remove licence history
#define SLS_REMOVE_ALL 0x0007 //remove all of above
```

pszRemoveReference

Specify the reference code for the removal. The Remove Reference code is obtained by calling SLS_GenerateRemoveReference in SlsLocalEx.dll.

pszRemovePassword

Specify the password for the removal. The Remove Password is obtained by calling SLS_GenerateRemovePassword in SlsLocalEx.dll.

Return Value

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned indicating the cause of the error.

Remarks

SLS_RemoveEx is an extended version of SLS_Remove. It works in the same way as the "Remove" function in the SlsAdmin application i.e. the user must generate a verifiable reference code prior to removal.

4.2.3.30 Request Licence

HRESULT SLS_Request(
LPCSTR pszProductID,
LPCTSTR pszUserName,
SLS_REQUEST *pRequest,
SLS_PERMIT *pPermit,
SLS_HANDLE *plLicenceHandle,
SLS_CHALLENGE *pChallenge)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszUserName

Points to a string that uniquely identify the user's name.

NB. The user name passed to the CSheriff constructor, or the SLS_Request function cannot be longer than 31 characters (32 including the null-terminator).

pRequest

Points to a request structure.

pPermit

Points to a permit structure to receive permission granted by the licence system.

plLicenseHandle

Points to an SLS_HANDLE in which a handle to the licensing context is returned.

Challenge

Points to a challenge structure. The challenge response will also be returned in this structure.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

The typical error status codes are:

```
SLS_E_LICENCE_UNREGISTERED
SLS_E_LICENCE_UNDEFINED
SLS_E_LICENCE_EXCEEDED
SLS_E_LICENCE_EXHAUSTED
SLS_E_LICENCE_EXPIRED
SLS_E_LICENCE_INVALID
```

Remarks

Use the **SLS_Request** function to request the licensing system to grant permission to the identified product to execute. If a valid licence is found, the challenge response is computed and SLS_SUCCESS is returned. A licence handle is also returned by the licence system to identify licensing resources allocated to the user. The

application must call **SLS_Release** to free the licensing resources.

A challenge response is *not* returned unless the licensing request completes successfully. If **SLS_Request** fails, the licence handle is set to 0, therefore there is no need to call **SLS_Release**.

The input parameter for computing challenge is the *Request* structure; the output parameter for computing challenge is the *pPermit* structure.

4.2.3.32 Request Licence (Extended)

HRESULT SLS_RequestEx(
LPCSTR pszProductID,
LPCTSTR pszUserName,
DWORD dwUserID,
SLS_REQUEST *pRequest,
SLS_PERMIT *pPermit,
SLS_HANDLE *plLicenceHandle,
SLS_CHALLENGE *pChallenge)

NB. The user name passed to the CSheriff constructor, or the SLS_Request function cannot be longer than 31 characters (32 including the null-terminator).

Extended version of SLS_Request with extra parameter dwUserID which can be used by the application to provide extra information about the user. The user ID can be retrieved by calling SLS_QueryUserInfo.

4.2.3.33 Set Licence

HRESULT WINAPI SLS_SetLicence(
LPCSTR pszProductID,
LPCSTR pszReference,
LPCSTR pszLicenceKey)

The **`SLS_SetLicence`** function asks the SLSAPI to authorise the product's licence policy.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszReference

Points to a string that uniquely identifies the application's Reference Code.

pszLicenceKey

Points to a string which contains a Licence Key.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

The application calls the SLS_SetLicence to programmatically issue a licence policy with its licence key. LicenceKey is issued to the user by the publisher with the user's reference code which is generated by a call to the SLS_GetReference.

4.2.3.34 SetOptions

HRESULT WINAPI SLS_SetOptions(
LPCSTR pszProductID,
SLS_HANDLE lLicenceHandle,
SLS_OPTIONS *pOptions,
SLS_CHALLENGE *pChallenge)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

lLicenseHandle

Specifies the handle to the licensing context. This parameter must be a handle created with the **SLS_Request** function.

Options

Points to an options structure.

pChallenge

Points to a challenge structure from which the challenge will be verfied.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

4.2.3.35 Set Permissions

HRESULT WINAPI SLS_SetPermissions(
LPCSTR pszProductID)

Parameters

pszProductID

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

Remarks

Sheriff requires write permission to its registry keys and licence files. This API function SLS_SetPermissions helps developers to set the write access permission to all relevant registry keys and files. SLS_SetPermissions must be called in the admin account, but only once for all. Typically, it is called at installation.

Note also SLS_IsAdminAccount, SLS_IsLimitedAccount. These two functions help developers to check whether the current user account is an admin or limited account.

4.2.3.36 Set Publisher Data

HRESULT WINAPI SLS_SetPublisherData(
LPCSTR pszProductID,
LPCTSTR lpszPublisherData);

Although Sheriff provides a comprehensive set of licensing features including time/unit metering, expiration control and feature access control, sometimes publishers might want to control their licences in special ways that would be difficult to implement with the standard features - Set/Get Publisher Data is designed for this purpose. Publisher data is simply a space up to 32 bytes that Sheriff uses to keep any data that publishers might specify. The format and meaning of the data is up to the publisher.

Publisher data is kept together with the product's Licence Key in the licence file. It is set by calling **`SLS_SetPublisherData`** and retrieved by calling **`SLS_GetPublisherData`**

Remarks

The sole function of the Publisher Data edit box in the Sheriff Licence Generator is to allow the publisher's data to be logged together with the details of the Licence Key. In other words, the publisher's data entered into the License Key Generator does not become part of the Licence Key, it is only logged in "slsgen.log". To use the publisher data you have to call **`Sls_SetPublisherData`**.

4.2.3.37 Set Session Options

HRESULT WINAPI SLS_SetSessionOptions(
LPCSTR pszProductID,
DWORD dwSessionOptions,
SLS_CHALLENGE *pChallenge);

pszProductID

Points to a string that uniquely identifies the application's Product ID.

dwSessionOptions

Specify one or a combination of the following options:

```
#define SLS_BYPASS_MACHINE_ID                   0x0001
#define SLS_BYPASS_MACHINE_SIGNATURE            0x0002
#define SLS_BYPASS_DISK_SIGNATURE               0x0004
#define SLS_BYPASS_FILE_SIGNATURE               0x0008
```

pChallenge

Points to a challenge data which should have been created by calling SLS_CreateChallenge with dwSessionOptions as the seed data.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned indicating the cause of the error.

Remarks

Sheriff is very rigorous in locking the licence to its hosting machine. Some locking mechanism include Machine ID, Machine Signature, Disk Signature and File Signature. Some developers wish to relax some locks under certain circumstances. The API function SLS_SetSessionOptions can be called to temporarily bypass some locks within a session.

Disk Signature is the signature of the hard disks.

Machine Signature is the signature of other hardware components including include CPU, BIOS, Motherboard etc.

Machine ID = Machine Signature + Disk Signature + Signature of OS

File Signature is the signature of licence files.

4.2.3.38 Terminate Licence

HRESULT WINAPI SLS_Terminate(
LPCSTR pszProductID,
LPSTR pszTerminationCode)

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

pszTerminationCode

Points to a string containing the termination code.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

See also SLS_VerifyTerminationCode in the Extended API.

4.2.3.40 Update Licence

HRESULT SLS_Update(
LPCSTR pszProductID,
SLS_HANDLE lLicenceHandle,
SLS_UPDATE *pUpdate,
SLS_PERMIT *pPermit,
SLS_CHALLENGE *pChallenge)

The **SLS_Update** function reports the up-to-date licence usage to the licence system and synchronizes between the licensed application software and the licence system.

Parameters

pszProductID

Points to a string that uniquely identifies the application's Product ID.

lLicenseHandle

Specifies the handle to the licensing context. This parameter must be a handle created with the **SLS_Request** function.

pUpdate

This parameter is used only if a policy is in effect that consists of unit metering licences (and if you choose to implement such a licensing policy in the application). The *UnitsConsumed* parameter in the *Update* structure specifies the total number of units consumed in this handle context since the initial call to the **SLS_Request** function. The *UnitsReserved* parameter specifies the total number of units to be reserved.

pPermit

Provides a structure for receiving licence permit. *Permit* data members are explained in the Permit data structure.

pChallenge

Points to a challenge structure. The challenge response will also be returned in this structure.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error.

The typical error status codes are:

SLS_E_LICENCE_SUSPENDED
SLS_E_LICENCE_EXCEEDED
SLS_E_LICENCE_EXHAUSTED
SLS_E_LICENCE_EXPIRED
SLS_E_LICENCE_INVALID

Remarks

Your application should periodically call the **SLS_Update** function to verify that the current licence is still valid. The maximum interval is 30 minutes.

The **SLS_Update** function verifies that the licence system context has not changed from the one expected by the

The function verifies that the licence system context has not changed from the one expected by the licensed software. The **SLS_Update** function can determine if the licensing resources granted to the specified handle are still reserved for this application. In a distributed licence system, an error might indicate a temporary network interruption. It can also determine if the licence system has released the licensing resources granted to the specified handle. An error indicates the software no longer has authorisation to execute in a typical manner.

The call to the **SLS_Update** function can also indicate that the current licensing context has expired. For example, in the case of a time-restricted licensing policy it returns the warning status SLS_LICENSE_EXPIRED.

If the number of new units requested in the *Update.UnitsReserved* parameter is greater than the number available, then the update request fails and it returns the error SLS_LICENCE_ EXHAUSTED.

If the call completes successfully, the value returned in the *Permit.UnitsGranted* parameter indicates the current total of units granted.

If **SLS_Update** returns an error, it does not return a challenge response.

If any error is returned, a call to the **SLS_Release** function is still required.

4.2.3.40 Verify Challenge

HRESULT SLS_VerifyChallenge(
SLS_SECRET *pSecretArray,
int nSecrectSize,
const BYTE *pbStream,
int nStreamSize,
SLS_CHALLENGE pChallenge)

The **`SLS_QueryUserInfo`** function verifies the specified challenge.

Parameters

pSecretArray

Points to an array of product secrets.

nSecretSize

Size of the secret array.

pbStream

Points to the buffer of input data stream.

nStreamSize

Size in byte of the input data stream buffer.

pChallenge

Points to a challenge structure from which the challenge will be verified.

Return Values

SLS_SUCCESS if the function is successful. Otherwise an error status code is returned to indicate the cause of the error. Two possible errors are SLS_E_CHALLENGE_UNSUPPORTED and SLS_E_CHALLENGE_ILLEGAL.

4.2.4 CSheriff Class Reference

CSheriff is a shrink-wrapper class of the Sheriff Licensing System API. The reference lists the member methods and explains in brief their functionality. Detailed discussion on API functions can be found in Sheriff API Reference.

Construction

```
CSheriff(LPCTSTR lpszProductID,LPCTSTR lpszUserName);
```

Construct a CSheriff object. lpszProductID points to a null-terminated string which identifies the product, lpszUserName points to a null-terminate string containing the user's name.

NB. The user name passed to the CSheriff constructor, or the SLS_Request function cannot be longer than 31 characters (32 including the null-terminator).

Set Product Secrets

```
Void SetSecrets(SLS_SECRET *parySecrets,int nSizeSecrets=4);
```

Sets Product Secret Codes. *parySecrets* points to an array of secrets, by default the size of the array is 4. Product Secrets are required only if you want to implement challenge protocol.

Example:

```
SLS_SECRET Secrets[4]=

{

        "0763-1985-3207-6621

        "1854-2076-4198-7532

        "2482-1593-2604-4927

        "3739-0628-9517-9618

};


CSheriff *pSheriff=new CSheriff("9758-3050-1918-9292-6466","John Smith");

Sheriff->SetSecrets(Secrets);
```

Get User Reference Code

```
BOOL GetReference(CString &strReference);
```

Returns a one-time user reference code.

Authorise Licence Key

```
BOOL SetLicence(LPCSTR pszReference,LPCTSTR lpszLicenceKey);
```

Authorise licence key on the user's machine. Reference must be provided through *pszReference* which is returned by calling GetReference.

Register Product

```
BOOL Register(LPCTSTR lpszProductName, LPCTSTR lpszLicencePath);
```

Registers a new product on the user's machine. l*pszProductName* points to a string identifying the product's name, *lpszLicencePath*points to a full path where licence files will be stored or have been stored.

License Product Programmatically

```
BOOL License(SLS_LICENCE Licence);
```

License is called to issue a licence from the application without user's intervention. The difference between SetLicence and License is that SetLicence requires a Reference Code and Licence Key, whereas License can issue a licence straight away. SetLicence must be called in challenge mode. SetLicence is often called once by application installation program to issue a trial licence.

Request Licence to Run the Application

```
BOOL Request(SLS_REQUEST Request,SLS_PERMIT &Permit);
```

Request is called by the application to apply for permission to run the application.

Update Licence State

```
BOOL Update(SLS_UPDATE Update,SLS_PERMIT &Permit);
```

Update is called periodically by the application to notify Sheriff that the licence is still being used and also update Sheriff on the usage of the licence.

Release Licence

```
BOOL Release(SLS_RELEASE Release);
```

Release is called by the application to notify Sheriff that the application exits. Sheriff will then release the licence and make it ready to be taken by other applications.

Succeeded or Failed?

```
BOOL Succeeded();
```

Succeeded indicates whether the last function call is successful or not.

Get Last Error Code

```
HRESULT GetLastError();
```

Returns the error code caused by last operation.

Get Last Error Message

```
void GetLastErrorMessage(CString &strError);
```

Returns the last error message.

4.3.1.1 Extended API - SlsLocalEx.dll

There are five functions available in the extended library, namely:

1. SLS_GenerateLicenceKey
2. SLS_GenerateRemovePassword
3. SLS_VerifyTerminationCode
4. SLS_VerifyStatusCode
5. SLS_GetVersionEx

1. Generate Licence Key

int SLS_GenerateLicenceKey(
LPCSTR lpszProductID,
SLS_SECRET *ppSecretArray,
int nSecretSize,
LPCSTR lpszReferenceCode,
SLS_LICENCE *pLicencePolicy,
LPSTR lpszLicenceKey)

Parameters

lpszProductID

Points to a string that uniquely identifies the application's Product ID.

pSecretArray

Points to an array of product secrets.

nSecretSize

Size of the secret array, must be set to 4.

pszReferenceCode

Points to a string containing the user's Reference Code.

pLicencePolicy

Points to a SLS_LICENCE structure containing licence features that to be issued to the user.

lpszLicenceKey

Points to a buffer in which the Licence Key is to be placed. The minimum size of the buffer is 64 bytes.

Return value

Definition	Value	Meaning
SLSKEY_SUCCESS	0	Success
SLSKEY_ERROR	1	General error
SLSKEY_E_BAD_PRODUCT_ID	2	Bad product ID
SLSKEY_E_BAD_REFERENCE	3	Bad reference code
SLSKEY_E_BAD_PID_REF	4	Reference code does not match product ID
SLSKEY_E_BAD_LICENCE_TYPE	5	Bad licence type
SLSKEY_E_BAD_LICENCE_DATE	6	Bad licence date

SLSKEY_E_BAD_SECRET_SIZE	10	Bad secret size
SLSKEY_E_BAD_SECRET_1;	11	Bad secret 1
SLSKEY_E_BAD_SECRET_2	12	Bad secret 2
SLSKEY_E_BAD_SECRET_3	13	Bad secret 3
SLSKEY_E_BAD_SECRET_4	14	Bad secret 4

2. Generate Remove Password

int WINAPI SLS_GenerateRemovePassword(
LPCSTR lpszProductID,
SLS_SECRET *ppSecretArray,
Int nSecretSize,
LPCSTR RemoveReference,
LPSTR lpszRemovePassword)

This function generates a password for removing licence files. The remove reference code must be provided in **lpszRemoveReference**, then the password will be returned in the buffer pointed by **lpszRemovePassword**.

SLS_SUCCESS is returned if the functions succeeds.

3. Verify Termination Code

int WINAPI SLS_VerifyTerminationCode(
const char *pszProductID,
const char *pszTerminationCode,
SLS_LICENCE *plicInfo);

This function is used to verify the termination code given by the user who wish to terminate his/her licence. The termination code is passed through **pszTerminationCode**, then the licence features before it was terminated is returned to the structure pointed by **plicInfo**.

SLS_SUCCESS is returned if the functions succeeds.

4. Verify Status Code

int WINAPI SLS_VerifyStatusCode(
const char *pszProductID,
const char *pszStatusCode,
SLS_LICENCE *plicInfo);

This function is used to verify the licence status code. The status code encrypts the current licence features. This function verifies whether the status code is valid and if so returns the licence features in the **plicInfo** structure. Software publishers can use this function to check the licence status at any time they wish, such as before the use's licence is to be renewed.

5. Get Version Numbers

HRESULT WINAPI SLS_GetVersionEx(
int *pnMajorVersion
int *pnMinorVersion);

This function returns the current version numbers of the API libraries.

The major version number is returned in the pnMajorVersion variable and the minor version number in the pnMinorVersion variable.

4.4.1 Introduction to Sheriff ActiveX Control

The Sheriff ActiveX Control enables you to protect your applications very easily while still benefiting from Sheriff's high level of security and comprehensive features. This guide leads you through the integration of the Sheriff ActiveX Control with a VB application; it also demonstrates various ways of using Sheriff to protect your applications. Although this guide uses Visual Basic for demonstration purposes, the procedure should be more or less similar in other programming languages such as VBA or Delphi.

The Sheriff ActiveX Control is designed to be used in two modes:

1. *Automatic Mode* you can protect your application with minimal programming. In this mode, Sheriff controls the pace of protection and fires the necessary events that your application should handle.

2. *Advanced Mode*: Sheriff acts in passive way and let your application control the pace of protection.

Control Installation

Before you can use any ActiveX control you need to register it with the Windows operating system. To register Sheriff ActiveX control, run the following command:

```
regsvr32 SheriffLocalCom.dll
```

Once the control is registered, you can then add the component to your toolbox or palette:

- *Visual Basic*: Click on Project/Components, select Sheriff Licensing Control and press OK. The Sheriff control will appear on toolbox.

- *Delphi*: Click on Component/Import ActiveX Control. Select Sheriff Licensing Control and press Install, then OK. The Sheriff control should appear in ActiveX palette.

4.4.2 Automatic Mode Quick Start

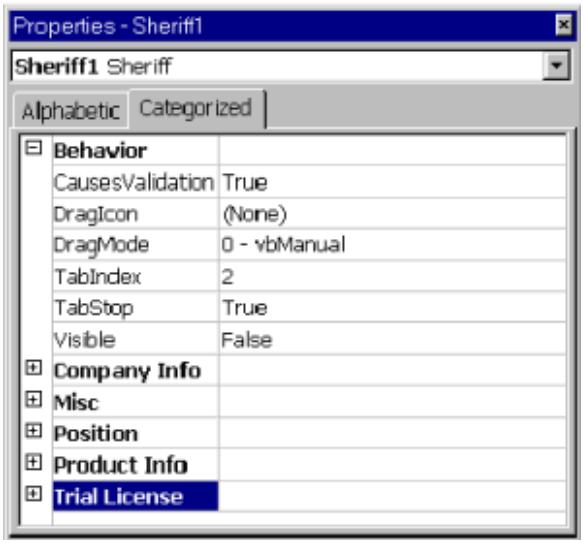
We'll demonstrate how easy it is to create a demo/trial application with the Sheriff ActiveX Control.

Step 1

From the toolbox, drag and drop a Sheriff ActiveX control anywhere in your application's main form. Since Sheriff will be acting as an invisible control, turn the "Visible" property to "False".

Figure 1 below shows the properties of the Sheriff control.

Fig. 1

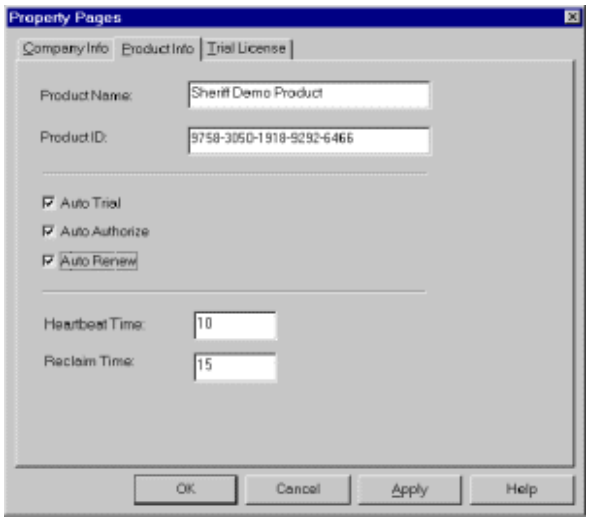


As you can see from the Figure 1, Sheriff categorises its properties into three types: Company Info, Product Info and Trial License. You can set those properties in the Properties Inspector window, as shown in Figure 1. Or you can set properties in with the property pages that Sheriff also provides.

Step 2

In the Property Inspector window, double click on "(Custom)". You will now see the property pages; click on the Product Info tab (see Figure 2 below).

Fig. 2



There are three options you need to set: Auto Trial, Auto Authorize and Auto Renew.

1. With Auto Trial on, Sheriff will detect if the application has ever been licensed on the machine on which the application is being executed. If the application has not been licensed, Sheriff will automatically issue a trial licence that is pre-defined by the application's publisher. To define the trial licence, click on the "Trial License" tab as discussed in step 3.

2. With Auto Authorize on, Sheriff will also detect if the application is licensed on the machine. If the machine is not licensed, instead of automatically issuing a trial licence, Sheriff will present a licensing dialog box to the end user as shown in Figure 5. Company information such as company name and address etc can be set in the "Company Info" property page, as discussed in step 4.

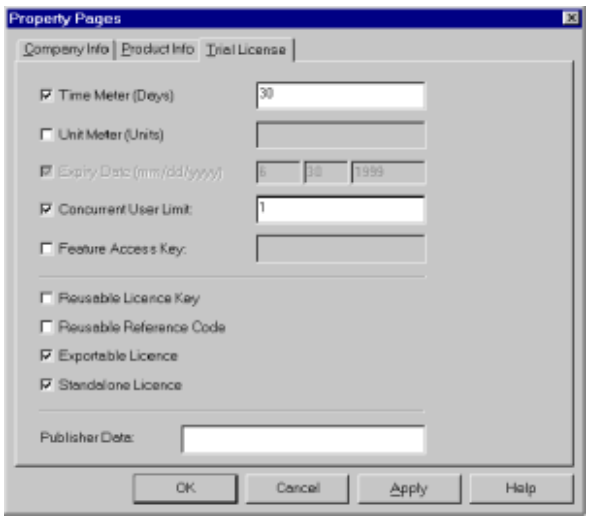
3. Similar to Auto Authorize, as soon as the application's licence expires Sheriff will automatically present a renewing dialog box to the end user and asks the user to renew his licence with the application's publisher.

Step 3

If, in Step 2, you decided to let Sheriff automatically issue a trial licence you need to define your trial licence.

Click on the "Trial License" tab to bring up the licence properties that you can defined for your trial licence (see Figure 3 below):

Fig 3.

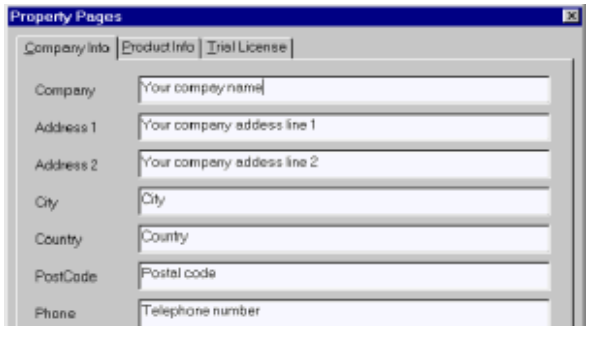


Step 4

If, in Step 2, you decided to let Sheriff automatically handle licence authorisation and/or renewal with its own dialog boxes, then you need to set up your company's information (to be displayed in the licence authorisation and renewal dialog boxes).

Click on the "Company Info" to display the property page (see Figure 4 below):

Fig. 4



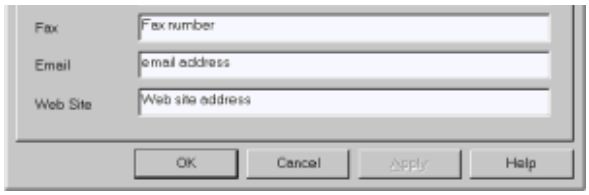
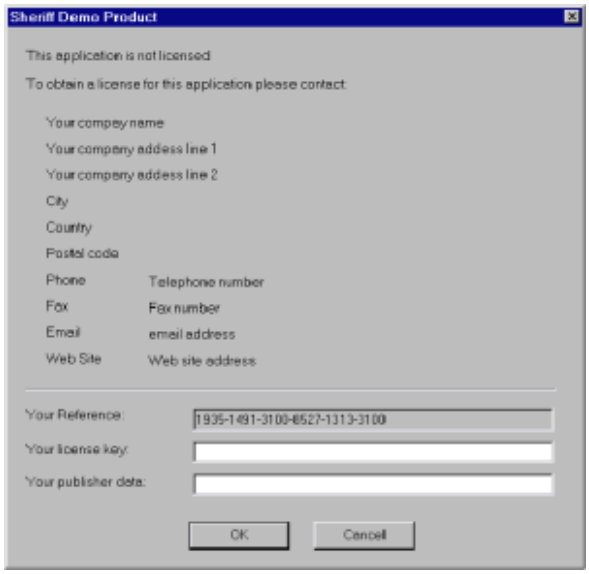


Fig. 5



Step 5

Now that you have decided your licensing options, defined your trial licence and filled up your company information, it is time to write some code to put your application under Sheriff's protection.

To run Sheriff in the Automatic Mode, the minumum requirement is one method you need to call and one event you need to handle.

The method to call is `CheckLicence`. Edit the form load event of your application main form and enter the following code:

```
Private Sub Form_Load()
   Sheriff1.CheckLicence
End Sub
```

Step 6

We're almost done apart from one event from Sheriff that you need to handle.

If you have decided to go for Auto Trial, you need to handle **`OnChallenge`** event. Sheriff fires up `OnChallenge` before it can issue a trial licence to verify the identity of the application

Add `OnChallenge` event handler to the main form and enter the following codes:

```
Private Sub Sheriff1_OnChallenge(ByVal ChallengingData As
String, ByRef ChallengedData As String)
   'we have to support challenge
   Dim Challenger As Object
   Set Challenger = CreateObject("Sheriff.Challenger")
   'Initialize the Challenger with secret codes
   Challenger.Secret1 = "0763198532076621"
   Challenger.Secret2 = "1854207641987532"
   Challenger.Secret3 = "2482159326044927"
   Challenger.Secret4 = "3739062895179618"
```

```
    hr = Challenger.CreateChallenge(ChallengingData,
ChallengedData)

End Sub
```

Please note that the Secret Codes used in the demo are for evaluation. You need to replace them with your own Secret Codes when you have purchased a Sheriff Serial Number.

And that's all, your application is now protected!

4.4.3 Automatic Mode Event Handlers

Now your application is protected by Sheriff under its Automatic Mode, you might want to add some more features to your protection. Basically this adding some event handles to handle the events that might be fired up by Sheriff.

1. The first event you might want to handle is **OnTooManyUsers**. This event is fired up by Sheriff when the concurrent user limit defined in the licence has been reached while a user is trying to login. In your event handler, you might firstly want to display a message box to say that there are too many users currently using the licence and then either exit the application or let the application continue to run for that user but only in demo mode with limited features.

   The event handler might look like this:

   ```
   Private Sub Sheriff1_OnTooManyUsers()
     MsgBox "There are too many users"
     'take appropriate action'
   End Sub
   ```

2. In the event of an error, Sheriff will fire up the **OnError** message. This event message comes with two parameters: **lErrorCode, bstrErrorMsg. lErrorCode** is the error code of the problem and **bstrErrorMsg** is the error message.

   The event handler might look like this:

   ```
   Private Sub Sheriff1_OnError(ByVal lErrorCode As Long, ByVal
   bstrErrorMsg As String)
      MsgBox bstrErrorMsg
   End Sub
   ```

3. If the Auto Trial is set on, Sheriff will automatically issue a trial licence (should the machine has never been licensed to run the application) ie. It is the very first time the user attempts to run the application. If a trial licence is issued successfully, Sheriff will then fire up the **OnTrial** message. If you wish to take any action when the trial licence is issued, such as presenting a message box to confirm that the user is now allowed to enter into the evaluation time, then you can handle **OnTrail** event.

   The event handler might look like this:

   ```
   Private Sub Sheriff1_OnTrial()
      MsgBox "You're now entering evaluation time"
      'take appropriate action'
      '...
   End Sub
   ```

4. During the evaluation period you might want to check the usage of the licence and remind the user of the current licence status. For instance, you may check to see how many days the user is allowed to use the application and how many days are left. This task is accomplished by providing an event handler for the **OnStart** message. Sheriff fires up the **OnStart** message once the call to the **CheckLicence** method succeeds.

   The event handler might look like this:

   ```
   Private Sub Sheriff1_OnStart()
      MsgBox "OnStart"
      hr = Sheriff1.QueryLicenceInfo()
      'take appropriate actions
      '...
   End Sub
   ```

5. If you elected not to let Sheriff automatically issue a trial licence or present its authorization dialog box when it detects that the machine has not been licensed, then Sheriff will fire up a **OnProductNotAuthorized** message. You need to handle this event in your code and provide your own dialog box or take whatever action that is appropriate to your requirements.

   The event handler might look like this:

```
Private Sub Sheriff1_OnProductNotAuthorized()
   MsgBox "Product is not authorized"
   'take appropriate actions
   '...
End Sub
```

6. Finally, if you decided not to let Sheriff automatically present its renewal dialog box as soon as it detects the licence has expired, then Sheriff will fire up an **OnLicenceExpired** message. You need to handle this event in your code and provide your own dialog box or take whatever action that is appropriate to your requirements.

   The event handler might look like this:

```
Private Sub Sheriff1_OnLicenceExpired()
   MsgBox "Licence Expired"
   'take appropriate action'
   '...
End Sub
```

4.4.4 Advanced Mode

If you call the **CheckLicence** method, you put Sheriff into "Automatic Mode". Under the Automatic Mode, Sheriff checks the status of the current licence and fire up appropriate messages whenever it is needs to. The application only needs to handle these message appropriately. Automatic Mode is easy to understand and implement and most of developers will want to use the Automatic Mode. However, some applications might have a protection requirement that is more complicated that that provided by Automatic Mode. These applications may need to control the time a licence should be issued and perhaps to perform some complicated tasks that make using Automatic Mode unsuitable. If this is the case for your application, then the Advanced Mode is what you need.

The key difference between the Automatic Mode and Advanced Mode lies in the call to the **CheckLicence** method: Automatic Mode calls this method to start, whereas Advanced Mode never calls this method.

Another difference you might notice is that Sheriff handles the heartbeating automatically under Automatic Mode. In fact, Sheriff does fire up an **OnHeartbeat** message on every heartbeat. Usually, there is no need for the application to handle this message. On the Advanced Mode, however, the application has to handle the heartbeating. This is usually done by inserting a timer control and call the **UpdateLicence** method on the event of timer message.

Should you decide to use the Advanced Mode, you may want to read the topic "Developer's Guide" for a complete knowledge of how it works and what procedures needed to follow. Although some of the methods are slightly different in name, the general principle is the same. A complete list of methods, events and properties are listed in the topic "Technical Reference". A demo project in VB 6.0 is also provided.

4.4.5 Technical Reference

This reference lists all of the properties, methods and events of the Sheriff ActiveX Control.

Properties

The properties are listed in their categories, namely Company Info, Product Info, Trial License and Miscellaneous.

Company Info

All of the properties in this category are for providing information about your company. This information is used and displayed on the licensing dialog box if Auto Authorize or/and Auto Renew options are set on.

1. CompanyName
2. Address1
3. Address2
4. City
5. Country
6. Post Code
7. Phone
8. Fax
9. Email
10. WebSite

These properties are self-explanatory.

Product Info

1. ProductName

   The name of your product.

2. ProductID

   The ID of your product. Sheriff ActiveX Control comes with a pre-defined evaluation product ID that can be used for your evaluation and testing. You need to obtain a product ID of your own if you decide to use Sheriff in your commercially released product.

3. AutoTrial

   Instructs Sheriff ActiveX control to automatically issue a trial licence.

4. AutoAuthorize

   Instructs Sheriff ActiveX control to automatically authorize a licence with a built-in licensing dialog box. Your company information will be displayed in the licensing dialog box as well as the user reference code, licence key and publisher's data.

5. AutoRenew

   Instructs Sheriff ActiveX control to automatically renew a licence with a built-in licensing dialog box. Your company information will be displayed in the licensing dialog box as well as the user reference code, licence key and publisher's data.

6. HeartbeatTime

   Defines the interval of heartbeating time. Heartbeating is a way of notifying Sheriff that the application is alive and, in the case of unit metering, updating Sheriff of the usage of the unit meter.

7. ReclaimTime

   Defines when Sheriff can reclaim a dead licence. A dead licence is the licence that was requested and held by an application but its heartbeat has ceased for a period of time that is longer than the HeartbeatTime.

Licence Info

1. Type

   Type of the licence. Its values are explained in the SDK and can be found in the slsapi.bas. With ActiveX control you don't need to manipulated it directly, instead you will want to set the following properties such as IsReusableKey, IsTimeMeter etc which are a breakdown of the possible values of the licence type.

2. IsReusableKey

   A reusable licence key makes the licence reusable on the same PC given that the hardware components do not change. A reusable licence key allows the user to recover his/her licence in the event that the licence is damaged or lost by accident. Typically you would only want to issue a reusable licence key with time or expiration restrictions.

3. IsReusableRef

   Sets the user reference code to be reusable.

4. IsExportable

   Defines whether or not a licence is exportable.

5. IsStandalone

   Defines whether a licence is a standalone or network licence. A standalone licence is locked to the machine from which the licence is authorised and therefore is not sharable with other users.

6. IsTimeMeter

   Define whether the licence is limited by a number of days that is defined by the **Meter** property.

7. IsUnitMeter

   Define whether the licence is limited by a number of units that is defined by the **Meter** property.

8. IsSetExpiration

   Defines whether the licence will expire by the predefined date that is set by the **EndDate** property.

9. IsSetCoUsers

   Define the maximum concurrent users that can use the licence simultaneously.

10. Meter

    Defines the maximum number of days if the **IsTimeMeter** is set, or the maximum number of units if the **IsUnitMeter** is set.

11. EndDate

    Defines the expiration date if the **IsSetExpiration** is set.

12. CoUsers

Defines the maximum number of concurrent users if the **IsSetCoUsers** is set.

13. AccessKey

    Defines the feature access level.

14. State

    Returns the current state of the licence when **QueryLicenceInfo** method is called. Please refer to the API Reference for its description.

15. StartDate

    Returns the start date or purchase date of the licence when **QueryLicenceInfo** method is called.

16. MeterUsage

    Returns the usage of the meter when **QueryLicenceInfo** method is called.

17. ActiveUsers

    Returns the number of active users when **QueryLicenceInfo** method is called.

18. Publisher Data

    Sets or returns the publisher's data.

Miscellaneous

1. LicencePath

   Defines the location of the licence files. If LicencePath is not given, Sheriff ActiveX control will create and locate licence files in the directory where it is installed.

2. UserName

   Provides the name of the user who is running the application.

3. MajorVersion

   The major version number of the control

4. MinorVersion

   The minor version number of the control

Methods

Apart from those methods that return boolean, all of other methods return an error code that indicates success or failure. A list of error codes can be found in the SLSAPI.bas and are explained in the API reference (see on-line help: Sheriff.chm).

1. CheckLicence() As long

   Call this method to check licence and run Sheriff ActiveX control in Automatic Mode.

2. GetReference(byref ReferenceCode As String)

   To get the user's reference code.

3. SetLicence(ReferenceCode As String, LicenceKey As String)

To issue a licence with the user's reference code and its licence key.

4. RequestLicence(byref AccessKey As long) As long

   To request a valid licence to run the application. AccessKey returns the feature access level defined by the licence.

5. UpdateLicence() As long

   To update licence on heartbeat.

6. ReleaseLicence() As long

   To release licence as soon as the application does not need the licence.

7. RequestLicenceUnits(UnitsReserved As long, byref UnitsGranted as long,

   byref AccessKey as Long) As long

   To request a valid licence and reserve a number of units to run the application. AccessKey returns the feature access level defined by the licence.

8. UpdateLicenceUnits(UnitsReserved As long, UnitsConsumed As long,

   byref UnitsGranted As long) As long

   To update licence on heartbeat with the number of units consumed. In addition, to modify the number of units the application wants to reserve.

9. ReleaseLicenceUnits(UnitsConsumed As long) As long

   To release licence when the number units has been consumed in the session.

10. IsExportableLicence() As Boolean

11. IsStandaloneLicence() As Boolean

12. IsReusableLicence() As Boolean

13. IsReusableReference() As Boolean

14. IsLicenceValid() As Boolean

15. IsProductLicensed() As Boolean

16. IsProductRegistered() As Boolean

17. IsProductInstalled() As Boolean

18. IssueLicence(bstrChallengeData As String) As Long

    To issue a licence. The licence features should be set via the licence properties such as **Meter, EndData** etc. You need to support challenge to be able to issue a licence. Please refer to the Advanced demo for codes of how to make a challenge.

    IssueLicence will also set the publisher's data.

19. ImportLicence(bstrReferenceCode As String, bstrImportKey As String) As Long

    To Import a licence.

20. ExportLicence(bstrReferenceCode String,byref bstrExportKey As String) As Long

    To export a licence.

21. RemoveLicence(bstrChallengeData As String ,Options As Long) As Long

    To remove a licence. The options are explained in the API reference. You need to support challenge to be able to remove a licence. Please refer to the Advanced demo for codes of how to make a challenge.

22. TerminateLicence(byref bstrTerminateCode As String) As Long

    To terminate a licence.

23. QueryLicenceInfo() As Long

    To query current licence information. Licence information is returned in the licence and licence info properties, such as Type, State, EndDate, StartDate etc.

    QueryLicenceInfo will also return the publisher's data.

24. GetLastError(byref LastError As long) As Long

    Returns last error code.

25. GetLastErrorMessage(byref ErrorMessage As String)

    Returns last error message

26. GetErrorMessage(ErrorCode As Long ,byref ErrorMessage As String)

    To retrieve the error message given an error code returned by any other method.

27. GetChallengeData(byref bstrChallengeData As String) As Long

    Used to make a challenge when calling **IssueLicence** or **RemoveLicence**.

28. GetStatusCode(byref bstrStatusCode As String, byref ErrorCode As Long) As Long

    To retrieve the current status code of the licence.

Events

Sheriff only fires up the following event messages when running in the Automatic Mode.

1. OnTrial
2. OnStart
3. OnChallenge
4. OnHeartbeat
5. OnError
6. OnProductNotLicensed
7. OnProductNotAuthorized
8. OnLicenceExpired
9. OnTooManyUsers

Please refer to the previous sections for a description of these messages.

4.5.1 Extended ActiveX Control - SlsLocalComEx.dll

SlsLocalComEx.dll is the ActiveX version of the Sheriff Extended API (SlsLocalEx.dll). As an ActiveX control, it is much easier to implement in scripting or programming environments like VB, VBA, VBScript and JavaScript. The ASP demo project shows how to use the extended Sheriff ActiveX control in an Internet Software Registration system.

Properties

1. Type

   Type of licence. Its values are explained in the SDK. With an ActiveX control you don't need to manipulate it directly, instead you will want to set the following properties, such as IsReusableKey, IsTimeMeter etc., which are breakdowns of the possible values of the licence type.

2. IsReusableKey

   A reusable licence key makes the licence reusable on the same PC given that the hardware components do not change. A reusable licence key allows the user to recover his/her licence in the event that the licence is damaged or lost by accident. Typically you would not want to issue a reusable licence key with time or expiration restrictions.

3. IsReusableRef

   Sets the user Reference Code to be reusable.

4. IsExportable

   Defines whether or not a licence is exportable.

5. IsStandalone

   Defines whether a licence is a standalone or network licence. A standalone licence is locked to the machine from which the licence is authorised and therefore is not sharable with other users.

6. IsTimeMeter

   Define whether the licence is limited by a number of days that is defined by the **Meter** property.

7. IsUnitMeter

   Define whether the licence is limited by a number of units that is defined by the **Meter** property.

8. IsSetExpiration

   Defines whether the licence will expire by the predefined date that is set by the **EndDate** property.

9. IsSetCoUsers

   Defines whether the licence has the concurrency limit that is specified by **CoUsers**.

10. Meter

    Defines the maximum number of days if the **IsTimeMeter** is set, or the maximum number of units if the **IsUnitMeter** is set.

11. EndDate

    Defines the expiration date if the **IsSetExpiration** is set.

12. CoUsers

    Define the maximum concurrent users that can use the licence simultaneously, if the **IsSetCoUsers** is set.

13. AccessKey

    Defines the feature access level.

14. ProductID

    The ID of your product. Sheriff ActiveX Control comes with a pre-defined evaluation Product ID that can be used for your evaluation and testing. You need to obtain a Product ID of your own if you decide to use Sheriff in your commercially released product.

15. Secret1, Secret2, Secret3, Secret4

    Product Secret Codes.

16. ReferenceCode

    Product Reference Code

17. MajorVersion

    The major version number of the control

18. MinorVersion

    The minor version number of the control

Methods

1. GetLicenceKey As String

   Before calling this method to generate the licence key, you need to make sure that the properties are initialised properly, such as the ProductID, ReferenceCode, Secret Codes and the licence features that you wish to issue. **GetLicenceKey** returns the licence key on success.

2. Succeeded As Boolean

   Succeeded is called to check whether the previous method succeeded or failed.

3. GetLastErrorCode as long

   If the method **Succeeded** returns false, you can call **GetLastErrorCode** to return the error code.

4. GetLastErrorMessage as String

   Or, if the method **Succeeded** return false, you can call **GetLastErrorMessage** to get the error message.

5. GetRemovePassword(bstrRemoveReference As String) As String

   This method generates a password for removing licence files. The remove Reference Code is provided in bstrRemoveReference; the method returns the password. Method **Succeeded** can be called to check whether or not this method has succeeded.

6. VerifyStatusCode(bstrStatusCode As String) As Boolean

   This method is called to verify the licence Status Code which is provided by bstrStatusCode. If the method succeeded true is returned otherwise false is returned. Licence features are returned in properties.

7. VerifyTerminationCode(bstrTerminationCode As String) As Boolean

   This method is called to verify the Termination Code given by the user who wishes to terminate his/her licence; the Termination Code is passed through bstrTerminationCode. If the method succeeded true is returned otherwise false is returned. Licence features prior to termination are returned in properties.

4.6.1 Sheriff ISR - Internet Software Registration

Functions

In an Internet Software Registration system, there are three main functions that can be implemented:

1. Reference Code generation
2. Licence Key authorisation
3. Licence Key generation

Manual & Automatic Systems

A simple manual system implements only the third function i.e. Licence Key generation.

- The user enters a Reference Code in a form at the publisher's web site then clicks on the submit button
- A licence key is generated and displayed on a web page.
- The user manually enters the licence key in their Sheriff-protected application.

In an automatic system all functions, such as entering codes, are accomplished without any user intervention. In other words, the entire process is automated so that, for example, once a credit card transaction has been completed the software will be automatically registered on the user's machine.

Please note that this documentation does not go into the details of how to set up credit card authorisation etc. Normally the user will select the desired software features from a form on the web site and on submitting this form a second form giving purchasing information will be displayed. The second form could, for example, calculate the purchase price of the software based on the user's chosen features and prompt the user to initiate a credit card transaction, however the mechanism for this will vary with the publisher's individual requirements.

Introducing Sheriff ISR

Sheriff ISR enables you to distribute licence keys via the Internet using Microsoft Internet Information Server (IIS). On the client side, Sheriff ISR provides this facility by an ActiveX control (SlsLocalComNet.dll) and on the server side by Sheriff Extended ActiveX Control (SlsLocalComEx.dll) via Active Server Pages (ASP).

Components

Name	Location	Type	Notes
SlsLocalComNet.dll	Client	ActiveX Control	Currently used only with Sheriff ISR.
SlsLocalComEx.dll	Server	Extended ActiveX Control	Use with ASP.

Overview

In the following we have assumed that if you want an automatic system, however you could choose to run the Extended ActiveX control on the server in a manual system.

Steps 1-7 (in Procedure, below) explain how to implement an automatic system via ActiveX controls & Active Server Pages. A registration form enables users to select the licensing features they want to buy. An ASP page then processes the form to generate the licence keys, which it does by calling the Sheriff Extended ActiveX control.

Demo

The demo application located in the folder "SDK\ISR" shows how to implement a simple ASP page by using the Sheriff Extended ActiveX control. RegisterForm2.htm is a sample registration form and Register2.asp is the sample ASP page. The demo also shows how to automatically generate the Reference Code and authorise the user's PC with the licence key generated by the ASP page, which is done by calling the Sheriff Internet ActiveX

control (SlsLocalComNet.dll) on the user's PC.

Microsoft has released useful information about ActiveX security and best practises here.

Procedure

Step 1
SlsLocalComNet.dll must be installed and registered on the client machine. Software publishers can distribute SlsLocalComNet.dll with their software packages and install & register it as part of the installation process. Alternatively, SlsLocalComNet.dll can be downloaded from Web sites and registered on the client PC by running "Regsvr32". Note that some browsers cannot run ActiveX controls.

SheriffEx.dll must be installed and registered on the server machine.

Step 2
The software publisher sets up an HTML form that enables users to select from the the various licensing options. RegisterForm2.htm is an example of this type of registration form.

Step 3
To automatically generate the Reference Code, you need to call GetReferenceCode method from the SheriffNet ActiveX control by using VBScript or JavaScript. RegisterForm2.htm calls GetReferenceCode as soon as the page is loaded, so that the Reference Code is automatically filled up. Following is the script found in the RegisterForm2.htm.

```
<SCRIPT LANGUAGE="JavaScript">
function loadPage()
{
  Sheriff.ProductID="9758    -3050-1918-9292-6466"
  var RefCode;
  RefCode=Sheriff.GetReferenceCode()
  if(Sheriff.Succeeded())
    document.RegForm.RefCode.value=RefCode
}
</SCRIPT>
```

Step 4
Once the user has selected the licence options to purchase, he/she will click on the "Submit" button to submit his/her order. The contents of the form will then be transmitted to the ASP page on the server specified by the ACTION attribute in the registration form. For example, in the RegisterForm2.htm, register2.asp is specified to process the form.

Step 5
In the Register2.asp, the registration form is parsed to get the options that the user has selected together with the Reference Code

Step 6
To generate the licence key, Register2.asp calls GetLicenceKey method from the SheriffEx ActiveX control with the Reference Code and licensing options.

Step 7
Once the licence key is successfully generated, you can automatically authorise the client PC with the key. To do this, you need to call SetLicenceKey method from the SheriffNet ActiveX control that is already installed on the client PC. If you want to provide the optional "publisher data", call SetPublisherData method. Following is the script found in the Register2.asp.

```
<SCRIPT LANGUAGE="JavaScript">

function SetLicenceKey()
{
    Sheriff.ProductID = "9758         -3050-1918-9292-6466"
```

```
    Sheriff.SetLicenceKey("<%=RefCode%>","<%=LicenceKey%>")
    if(Sheriff.Succeeded()==true)
  {
      if("<%=PubData%>"!="")
        Sheriff.SetPublisherData("<%=PubData%>");
  }

    if(Sheriff.Succeeded()==true)
  {
       alert("Your machine is successfully licensed");
  }
   else
  {
      var ErrorMessage;
        ErrorMessage=Sheriff.GetLastErrorMessage();
       alert("Error: "+ErrorMessage);
  }
}
</SCRIPT>
```

4.6.2 Client Side ActiveX control - SlsLocalComNet.dll

SlsLocalComNet.dll contains a subset of functions of the Sheriff ActiveX control and is designed specially for Internet applications, such as an Internet Software Registration system. This enables software publishers to automate the generation of user Reference Codes and the authorisation of licence keys.

To implement a Sheriff ISR automatic system, SlsLocalComNet.dll must be installed and registered on the user's PC.

Properties

ProductID

The ID of your product. Sheriff ActiveX Control comes with a pre-defined evaluation Product ID that can be used for your evaluation and testing. You need to obtain a Product ID of your own to use Sheriff in your commercially released product.

Methods

1. GetReferenceCode As String

   Returns user's reference code of the PC to which the licence will be authorised.

2. SetLicenceKey(RefCode as String, LicKey as String)

   Authorise the PC with the licence key and its user reference code.

3. SetPublisherData(PubData as String)

   Saves the publisher's data in the licence file on the PC.

1. Succeeded As Boolean

   Succeeded is called to check whether the previous method succeeded or failed.

2. GetLastErrorCode As long

   If the method **Succeeded** returns false, you can call **GetLastErrorCode** to return the error code.

3. GetLastErrorMessage As String

   Or, if the method **Succeeded** return false, you can call **GetLastErrorMessage** to get the error message.

5.1.1 Notes

Applications

SlsDemo	The Demo Application provides a simple example of a Sheriff-protected progam.
SlsPsn	The Product ID & Secret Codes Generator is used by registered Sheriff software developers to generate unique Product IDs & Serial Codes for their applications (normally one Serial Number per application is required).
SlsGen	The Licence Generator is used by publishers of Sheriff-protected products to generate licence keys for their users.
SlsAdmin	The Administrator is used by end users of Sheriff-protected products to manage their licence keys.
SlsServer	Sheriff application that runs on a network server.

Developers: Please Note

The full functionality of the SlsAdmin & SlsGen applications is replicated in the API and Extended API (via SlsApi and SlsApiEx respectively). This enables you to choose whether or not to distribute the Administrator to your end users as a stand-alone application or to integrate some or all of its functionality within your own program.

The Administrator On-line Help

The Sheriff Administrator on-line help is provided both as a topic in the Sheriff on-line help that you are currently viewing and also as a stand-alone application (SlsAdmin.chm).

If you want to distribute the Sheriff Administrator On-line Help please note:

1. The Compiled Help File

   MS HTML Help is a single (.chm) file HTML help system, using its own Microsoft HTML Help viewer (hh.exe), currently available for Windows only and using MS Internet Explorer 3.02+. Therefore your users need to have Microsoft Internet Explorer 3.02 or above and the HTML Help viewer set up on their computers.

2. The Viewer

   The MS HTML Help viewer is included in Windows 98 and is widely available but it is possible that some of your users will not have it. This means that you may need to distribute the viewer with your product.

   You can redistribute the HTML Help Viewer components as needed by redistributing the HTML Help Viewer Installation and Update package (Hhupd.exe). The package consists of a self-extracting executable program that will set up the Help Viewer on a user's computer or update an existing setup of the Help Viewer. Hhupd.exe is available in 28 languages, for both x86 and alpha platforms and can be downloaded from the HTML Help Download Page at the Microsoft Web site. Note that Hhupd.exe will not run on a Windows NT system unless you have administrator privileges.

3. MS Internet Explorer

   It is not required that Internet Explorer be used as the system's default browser, or that the Internet Explorer icon be visible on the user's desktop.

Password Protection

Both the Licence Generator and Administrator feature the option of password protection but please note that the security provided is very basic.

5.2.1 Sheriff Product ID & Secret Codes Generator

Serial Number

Every Sheriff-protected product requires a unique Product ID and Secret Codes, which are generated from the Serial Number issued to you at the time of purchasing Sheriff. To generate your Product ID and Secret Codes enter the Serial Number in the Product SN box then click on Generate.

The Product ID is used to identify your product to Sheriff; you must register your Product ID in the Sheriff Licence Generator's Registry before you can issue licence keys to your users.

Secret Codes prevent third parties from generating licence keys for your product - even if they have access to the Sheriff Licence Generator. As an additional precaution the Secret Codes are also available in an encrypted format, designed to deter hacking. Whether you include the numeric or encrypted Secret Codes in your product is up to you.

Evaluation Product ID & Secret Codes

If you have not purchased a Serial Number you can use the evaluation Product ID and Secret Codes to evaluate Sheriff.

5.3.1 Licence Generator Overview

You use the Sheriff Licence Generator to generate licence keys for your customers.

What your customers have to do

1. When your customers install (or run) your application for the first time they are presented with a dialog box (that you can customise) that gives them a unique Reference Code for your application running on their machine.

2. They pass this code to you (by phone, fax, or email).

3. You give them in return a Licence Key that they enter into the dialog box. This then licenses the application to run on their machine (or network).

What you have to do

1. Your customer supplies you with a Reference Code that they get when they install (or run) your application for the first time.

2. You use the Licence Key Generator application (SlsGen) to create a Licence Key for the customer, using the supplied Reference Code.

3. You create the Licence Policy

5.3.2 Evaluation Product ID & Secret Codes

These are provided for evaluation purposes only and permit the generation of an unlimited number of 30-day licence keys. To fully protect your product you will need to purchase a unique Serial Number.

Please note the following:

- Using the Evaluation Product ID will, of course, lead to certain limitations within your program E.g. The IsTimeMeter property will be set to True, since the licence keys must expire after 30 days.
- Use of the encrypted Secret Codes is optional - refer to the Demo to see how the encrypted secret codes can be used in your application. You should enter the plain Secret Codes in the Sheriff Licence Generator.
- The Product ID and Secret Codes for the Sheriff Demo can be found in topic 6.2.1.6

Product ID

9758-3060-1918-9292-6467

Plain Secret Codes

Secret Code 1: 0763-1985-3207-6722

Secret Code 2: 1854-2076-4198-7633

Secret Code 3: 2482-1593-2604-4028

Secret Code 4: 3739-0628-9517-9719

Encrypted Secret Codes

SLS_SECRET g_arySecrets[]=

{

{0x76,0x3E,0x6D,0x36,0x14,0x14,0x6D,0xB8,0x1D,0xCA,0xE7,0x45,0xFB,0xB3,0xE0,0x13,0x94,0x27},

{0x00,0xB7,0x17,0x00,0x43,0xFC,0xC0,0x7F,0x8F,0xA2,0x49,0x89,0xB3,0x44,0x5E,0x15,0x50,0xEF},

{0xAC,0x2E,0x39,0x6F,0x45,0xD5,0x41,0x11,0x43,0x41,0x10,0xE6,0x6F,0x21,0x4C,0x1D,0x8A,0xB1},

{0x79,0x52,0x9A,0x49,0x97,0x8B,0x1C,0x15,0x64,0xB7,0xE8,0x45,0x0F,0xFA,0xF0,0xE3,0x38,0xAA},

};

5.3.3 Registering a Product

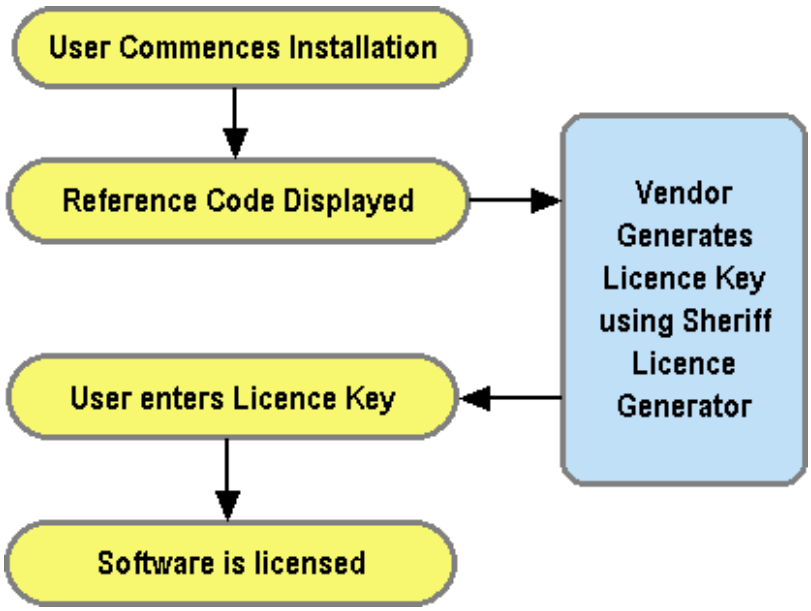
Introduction

To generate licence keys, a product will need to be *registered* in the Sheriff Licence Generator. Registering a product requires a Product ID and Secret Codes, which are embedded in the product by a software developer.

- Select File from the drop-down menu, then Register Product. The Register a New Product dialog box appears.
- Type in the name of your product and type or paste the Product ID and the four lines of Plain (numeric) Secret Codes.
- Click OK (you can review the product details any time by selecting the Properties button).
- When your product has been registered you can start generating licence keys.

5.3.4 Licensing a Product

Overview



Generating a Reference Code using SlsAdmin

The Reference Code is displayed either on installation or on first use, depending on the design of the publisher's (P's) application).

The user (U) must generate a Reference Code on the hardware on which the licence file is to be created, which could be a standalone PC or network file server.

Alternatively U can generate a Reference Code using the Sheriff Administrator (SlsAdmin) as follows:

- U opens the Product Registry by selecting Licence | Registry (or by clicking the Registry button on the toolbar).
- The Registry displays details of the registered products, including Product ID and Licence Path.
- U selects the product and clicks on the License button and the Reference Code is displayed.
- U informs P of the Reference Code.
- (Optional) U informs P of the Status Code.

Generating a Licence Key using SlsGen

The publisher (P) issues the user (U) with a Licence Key as follows:

- P enters the user's name.
- P selects the appropriate product from the drop-down list.
- P enters U's Reference Code.
- P defines the Licence Policy by enabling the licence features and selecting the options.
- P clicks on the Generate button to produce a Licence Key for the User.
- U enters the Licence Key in the Licence Product dialog box and clicks Save. The product is now licensed.

5.3.5 The Licence Policy

As you can see, part of the licensing process involves creating a Licence Policy. The Licence Policy defines exactly how the user will be able to operate your Sheriff-protected application, for example whether use of the application will be restricted to a particular machine or whether the licence will expire after a certain number days.

Please note the following important points:

- Details of every licence you issue are saved to a log file (SlsGen.log), to which you can append your own comments.

- Generally, renewing a licence overwrites the existing policy. For example, if you want to extend a user's licence from thirty days to sixty you must take into account the number of days already used. In this example, if the user had already used up fifteen days, you would issue a licence key for forty-five days. You can check the number of days that have been consumed by using the Verify Status feature. Alternatively you can use the Extend Existing Licence option when issuing the licence key.

- Where security takes priority over flexibility we recommend that you define a Licence Policy where the licence is "Standalone" (i.e. locked to a single machine - cannot be shared over a network) and where the Move and Export functions are disabled.

- While a "Standalone" licence is very secure and, among other things, locks to the speed of the processor, Intel has introduced a technology for mobile computing known as "SpeedStep", which is implemented on the Pentium III mobile processor. Because it is possible to configure SpeedStep enabled processors to vary the speed at which they operate (in order to conserve battery power) a Standalone licence may give rise to error messages. In this case the Standalone box should *not* be checked - simply issue a policy for one concurrent user.

  NB. With Sheriff 2.41 and above this problem should not occur with any kind of licence key.

Licence - Time Meter

Software runs for a specified number of days (E.g. 21 days).

Licence - Unit Meter

Each chargeable event is known as a 'unit' (E.g. an execution of the software, viewing/printing a document, or any specified period of time). Note that any additional days/units you *will overwrite* those already issued. E.g. if you first issue a licence key for 180 days and during this time the user requests that the licence be extended to a total period of one year, you must take into account the units already consumed when issuing the new licence. This is not a problem because you can verify the user's Status Code prior to issuing the new licence key.

Licence - Expiry Date

The licence will expire on the selected date.

Licence - Concurrent User Limit

Limits the maximum number of concurrent network users. If this box is not checked then the number of concurrent users will be unlimited. If you check the box and the limit is set to "1" you have the option of restricting usage to a single machine i.e. the licence will be tied to the machine on which it is installed and cannot be shared over a network (see Standalone Licence Key below).

Note that unless the Disable Export option is "on" (see below) licence keys up to the maximum number of permitted concurrent users will be exportable to other machines and the number of concurrent users permitted on the source machine will be decremented accordingly (see the Sheriff Administrator help for further details).

Licence - Feature Access Key

Software can be licensed as shareware, demo software or buy-as-you-need.

Licence - Extend Existing Licence

Enables publishers to extend rather than replace an existing licence, so that the client's existing licence policy is not overwritten when the new Licence Key is issued. This is used when the clients' licence is controlled by day or unit metering and the publisher wishes to incrementally add new days/units to the existing licence.

Options - Reusable Key

The user should keep a record of the licence key number. Provided you enable the 'Reusable Key' option when defining the Licence Policy, the user will be able to regenerate the key on the original hardware using the Licence Administrator (this can be done if, for example, the licence file is accidentally deleted). The 'Reusable Key' option can only be enabled if the key is to last indefinitely (not metered). Note that even a reusable key *cannot* be restored if the user's hard disk has been reformatted.

Options - Reusable Reference Code

This option is only available when Expiry Date metering has been selected. It is used when you want to be able to extend the date on which the licence will expire without first contacting the user for a new Reference Code. In this way you can at any time extend the period for which your application will run simply by generating a new licence key for the user.

Options - Disable Export

You can disable the user's ability to export a licence key to another machine, so locking the licence to one machine only.

Important!

- For security reasons export must be disabled when you issue a Reusable Key (the Disable Export option is automatically invoked when you issue a Reusable Key).

- In a network configuration the maximum number of licence keys that can be exported is equal to the maximum number of concurrent users defined in the Licence Policy. Therefore if you permit an unlimited concurrent users and Disable Export is *not* set, an unlimited number of licence keys could be exported.

Options - Disable Move

You can disable the user's ability to move a licence to another machine using the Move function. This prevents a user from permanently relocating a licence to another machine E.g. when they upgrade their hardware.

Options - Standalone Licence Key

When the Concurrent User Limit is set to "1", checking the Standalone Licence Key box ensures that the protected application can only execute on the machine on which it is installed i.e. the application is not available over a network and the licence file must reside on the same machine as the application.

Comments

You can add any comments to the log file (SlsGen.log), an ASCII text file which automatically keeps track of all transactions in the Licence Generator.

Publisher Data for User (optional)

If a publisher feels that the Sheriff Licence Key does not provide enough features, the Licence Key can be supplemented by Publisher Data. Publisher Data is saved with the Licence Key and is in effect an an extension to the Feature Access Key facility. However, unlike the Feature Access Key, Publisher Data can be up to 32 bytes long and is entered by the user at the time of entering the Licence Key.

NB. The publisher's data may be saved or retrieved by using the appropriate Sheriff API functions. The sole function of the Publisher Data edit box in the Sheriff Licence Generator is to allow the publisher's data to be logged together with the details of the Licence Key. In other words, the publisher's data entered into the License Key Generator does not become part of the Licence Key, it is only logged in "slsgen.log". To use the publisher data you have to call the API function **Sls_SetPublisherData**

5.3.6 Verify Status

Use Verify Status to display the features of a user's existing licence prior to issuing a new key. This is particularly important if you want to renew a licence that has not already expired as the new licence policy will overwrite the existing one. In such cases you will need to take into account any days/units etc. that remain on the old policy.

The user must run the Sheriff Administrator (SlsAdmin), the publisher must run the Licence Key Generator (SlsGen).

Sheriff Administrator

- The user (U) selects the Registry and the appropriate product from the combo box.
- U clicks on License and then on 'yes' in Confirm Buy dialog box.
- The status code of the existing licence is displayed.

Licence Key Generator

- The publisher (P) clicks on the key in the top left hand corner and chooses Verify Status Code from the drop down menu.
- P enters U's status code (and optionally the user's name and any comments in the Comments box) and clicks Verify.
- The licence features are displayed and details will be appended to the Licence Generator Log file (SlsGen.log).

5.3.7 Terminating a Licence

Generally a user's licence will terminate automatically, for example if unit metering has been enabled and all the units have been consumed, however there may be instances where it is necessary to terminate a licence manually.

For example, a user may wish to change a hardware component but be unable to use the 'import' facility because there is no other system available (for more about 'import' see the Licence Administrator help). In such cases the user requests the publisher to generate a new licence key but before doing this the publisher must verify that the original licence has been terminated.

The user (U) terminates a licence using the 'Terminate' option in the Sheriff Licence Administrator (SlsAdmin.exe). Terminating a licence generates a termination code that can be verified by the publisher (P) using the Verify Termination Code facility in the Sheriff Licence Key Generator.

Administrator

- U clicks the Licence menu and selects Terminate.
- The Terminate Product dialog box displays the currently registered products, U selects a product to terminate and clicks on the Terminate button.
- When U has confirmed that the product should be terminated, the termination code is displayed.

Licence Key Generator

- P runs the Licence Key Generator, clicks the key icon at top left and selects Verify Termination Code on the drop down menu.
- P enters U's Termination Code in the dialog box - it is not possible to complete the entry of an invalid code.
- When the code is successfully entered P adds any comments in the Comments box.
- P clicks on the Verify button to display the licence properties.
- Details of the licence termination are appended to the Licence Key Generator Log file (SlsGen.log).

5.3.8 Removing a Product

The Remove Product facility enables you to completely remove a Sheriff-protected product, for example where you want to reinstall the product again from scratch on your development machine.

Normally this option is not made available to the end user - use Terminate instead. By using the Terminate procedure you ensure that a product is removed from the user's machine without by-passing the Sheriff protection (E.g. which protects against reinstallation), whereas Remove completely undoes the installation. The procedure is password protected.

Administrator

- Click on Licence | Remove.
- In the Remove Products dialog box select the product and click Remove.
- A Reference is displayed.

Licence Key Generator

- Click the key in the top left hand corner.
- Select Generate Remove Password.
- Enter the Reference Code and click on Generate to generate a password.

Administrator

- Enter the password and click on OK. The product is now removed.

5.4.1 Administrator Overview

With the Administrator you can:

- Register and/or license Sheriff-protected products.
- Monitor licence usage (E.g. by networked users).
- Manage networked users (Eg. suspend a user's access).
- Maintain portable licences (using the import/export facility).
- Regenerate a licence file that has been deleted or corrupted.
- Terminate a licence manually and provide verification to the publisher
- Remove a licensed product (used by software developers for debugging)

All Sheriff-protected products can be managed from a single Administrator, even those from different software publishers.

5.4.2 Licensing & Registering a Product

Overview

- *Licensing* a product creates a Licence File. The Licence File contains details of the licence policy E.g. when the licence will expire, how many concurrent users are permitted etc. Without a valid Licence File the product cannot run. The Licence File is often created as part of the product installation process but it can be created manually using the Sheriff Administrator (see *To Licence a Product* below).

- *Registering* a product creates the appropriate key in the Windows Registry. The key contains important information about the Licence File i.e. its name and location. Registration is also normally a part of the installation process, however it is also possible to register a product manually. This is usually necessary when a product has already been installed on a network server and you wish to run the product on a workstation.

Important Note: Reusable Keys

If you have been issued with a 'Reusable' Licence Key it is important that you make a note of the number . This will enable you to regenerate the Licence File *on the same machine** in the event that it is overwritten or damaged. Enter the Licence Key number into the Licence Product dialog box and click Save to regenerate the Licence File.

* Provided the hard disk has not been reformatted or the operating system reinstalled.

To License a Product

As explained above, licensing a product creates a Licence File and normally this is not necessary since the file will have been created automatically as part of the installation process. If you need to create a Licence File manually, follow these steps:

1. Click on the Registry button in the Sheriff Administrator to bring up the Register Products dialog box.

2. Enter your product name in the Product Name box and your Product ID in the Product ID box and specify a path where you would like to create the Licence File.

3. Click on the License button. The License Product dialog box now displays a Reference Code and prompts you for a Licence Key, which is obtained from the publisher. When a valid key has been entered the product will be licensed.

To Register with an Existing Network Licence

The Register procedure is normally only used to register a product with an existing network Licence File, which is usually located on a file server. In other words, it "tells" the Sheriff protection where to find a valid Licence File on the network:

1. Click on the Registry button in the Sheriff Administrator to bring up the Register Products dialog box.

2. Enter your product name in the Product Name box and your Product ID in the Product ID box and specify the path to the Licence File (you can use UNC to specify the path).

3. Click on the Register button.

5.4.3 Importing/Exporting a Licence

Overview

The *import* facility enables you to operate a Sheriff-protected product on a machine for which you do not have a licence, unless this facility has been disabled by the software publisher.

To understand this topic it is important to appreciate the difference between the ***source*** machine and the ***target*** machine, since different rules apply to each. The ***source*** is the machine on which the licence was originally registered i.e. the machine for which the publisher issued the original licence key. The ***target*** is a machine to which you export a licence from the source machine.

To run a Sheriff-protected product on a ***target*** you may import a licence from the ***source*** provided that (a) the software publisher enabled this facility when defining the licence policy and (b) you adhere to the operational rules set out below.

You can import/export both standalone and network licences. All import/export activity is recorded in the log file (SlsAdmin.log).

NB. If you want to permanently export the ***source*** licence you should use the Move facility. For example, if you wanted to export a licence to a laptop for a few days, you would use the export facility but if you wanted to upgrade your file server you would use the move licence facility to relocate the licence to the new server.

Operational Rules

1. The product must be registered and licensed on the ***source*** machine and also registered on the ***target*** machine before you can import the licence.

2. You can define the features of the exported licence, provided these remain within the terms of the licence policy.

3. When either the users or units have been fully exported the application cannot execute on the ***source*** until the licence is imported back from the ***target.***. For example, you have a licence with two users and ten units on the ***source*** and you export one user and ten units to the ***target***; what remains on the ***source*** is one user with zero units - meaning that the user will be unable to run the application on the ***source*** as there are no available units.

4. A licence can only be imported either to a machine that does not already have a valid licence or to the ***source*** machine.

5. If the licence on the ***source*** is modified after a licence is exported, then the exported licence cannot be imported back (reclaimed) by the source machine - Sheriff sees a modified licence as a different licence. For example, if the publisher issues more units to the source machine, then the exported licence cannot be reclaimed.

6. A licence can only automatically restore on the ***source*** machine - there is no automatic restoration on the the ***target*** machine. Note that a licence cannot automatically restore on the source if the source licence is modified between the time of exporting and restoration.

Exporting Different Types of Licence

What happens to a licence when it is exported depends on the licence policy (i.e. on the licence's attributes, such as the type of metering in force and number of concurrent users allowed), as set out in the table below.

Concurrent Users	The number of concurrent users on the source machine will be reduced by the number exported until the exported licence expires or is imported back to the source machine.
Time Limited (Metering /	As soon as an exported time-limited licence expires, the source licence will be automatically reinstated. For example, you can set the exported licence to expire after seven days and

Expiry Date)	when it does the licence is automatically restored on the source machine (a licence *cannot* automatically restore except on the source machine).
Unit Metering	It is not possible to automatically reinstate a unit metered licence. To recover the unconsumed units in the exported licence, the user has to manually import the licence back to the source machine.

Procedure

Target

1. *Register the product* you wish to use by clicking the Registry button on the toolbar to open the Sheriff Registry (proceed to 1(b) if the product is already registered). Type in the name of the product, the Product ID and select a path for the licence file. Close the Registry.

2. *Import the product* by clicking the Import button - the Import Product dialog box displays. Select the product you want to import, then click the import button - an Import Licence dialog box displays a Reference Code.

Source

1. Click the Export button and the Exporting Licence dialog box is displayed.

2. Select the product to be exported and click the Export button. The Export dialog box is displayed.

3. Enter the Reference Code referred to above and define the Licence Policy by selecting the appropriate features (E.g. Expiry Date). Note that the available features will depend upon the attributes of the policy. Click on Generate and a licence key is displayed.

Target

1. Enter the licence key into the Import Licence dialog box.

2. Click on Import and the product is enabled.

5.4.4 Moving a Licence

Overview

Moving a licence is a special kind of exporting. When you want to permanently relocate a licence from one machine to another, you use the Move function. The Move function is also useful if you want to upgrade your hardware, since the licence can be moved temporarily to another machine and then moved back again.

For example, if you were upgrading your file server you would want to move the current licence to the new server. In this instance, don't forget that you would have to re-register the licence on the client PCs.

Procedure

In this example, PC1 is the original machine and PC2 is the machine to which you wish to transfer the licence.

PC1 : Select Licence | Move and choose the product licence that you wish to move.
PC1 : Click on Move and you will see the Move Licence dialog box.
PC2 : Generate a Reference Code
PC1 : Enter the Reference Code into the Move Licence dialog box.
PC1 : Click on Generate to generate a Licence Key
PC2 : Using the Licence Import facility, import the licence

Notes

- An exported licence cannot be moved, it can only be imported back to the source machine..
- All exported licences should be reclaimed prior to moving the licence on the source machine..
- The software publisher may disable the move facility.

5.4.5 Terminating/Removing a Licence

Generally licences terminate automatically, for example if unit metering has been enabled and all the units are consumed, however there may be instances where it is necessary to terminate a licence manually.

For example, you may wish to change a hardware component but be unable to use the 'move licence' facility (see previous topic) because there is no other system available. In such cases you must request the publisher to generate a new Licence Key for your software but before doing this the publisher must verify that the original licence has been terminated.

Procedure for Terminating a Licence

- Click the Licence menu and select Terminate.
- The Terminate Product dialog box displays the currently registered products. Select a product to terminate and click on the Terminate button.
- When you have confirmed that the product should be terminated, the Termination Code is displayed. Make a note of this so that it can be verified by the publisher. When the product is re-installed the publisher will issue a new Licence Key.

Removing a Product Licence

The licence removal facility is normally only used by software developers and is documented in the Sheriff Licence Key Generator on-line help.

5.4.6 Networking: The Basics

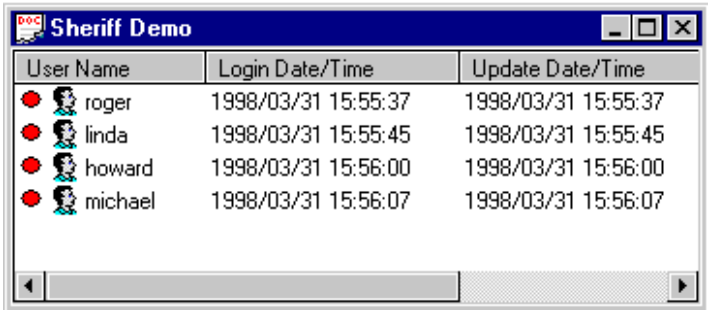
To provide network access to a Sheriff-protected product you need to ensure that the licence file is in a shared location - either a mapped drive or else a drive that is accessible via UNC.

The product has to be registered on every workstation on which it is to be run. Run the Sheriff Administrator on each workstation to specify the Product ID and the path to the licence file. See topic Licensing & Registering

5.4.7 Networking: Licence Monitoring

You can monitor active programs and which active users have access to them.

Run the Sheriff Administrator on the server (the machine on which the shared licence is installed) then click on Monitor in the toolbar to display a list of software programs for which you are licensed. Select a product from those listed in the Product window and click on Monitor. The Properties for the selected product are displayed. Click on Close and the active users are displayed.



In this example four users are running the Sheriff Demo program.

- Login Date/Time shows when an active program was opened by a user.
- Update Date/Time shows when an program was last verified (see below)

A red circle indicates that the program has been verified by the Sheriff Licence System (SLS) and granted permission to run. Usually an active program updates SLS with its current status every five minutes or so but the specified interval varies according to the program.

A red circle changing to white indicates that the program has not been verified by SLS during the specified interval and will be terminated unless the situation is corrected.

5.4.8 Networking: Delete Users

Highlight the user you wish to delete and click on the delete button. You will be prompted to confirm the deletion.

Normally you will only do this you when you have reached the limit of authorised concurrent licences and you wish to grant access to another user. For example, if you have a concurrency limit of four users and it is urgent that John is granted access, you could choose to delete Michael to make way for John.

Note that deleting a user does not prevent that user from running the software program again - it temporarily suspends the user. Also, the effect of deleting a user may be delayed, depending on how frequently the program checks its licence file.

5.5.1 Sheriff Server

Full instructions for running SlsServer are given in topic 3.4 Networking.

Please note that the SlsClock application is now obsolete.

6.1 Installation

How compatible is Sheriff with InstallShield?
InstallShield can call a Windows DLL, so it can also call Sheriff. Just call the Sheriff DLL from your InstallShield script code.

In a competitor product there have been instances where it couldn't retrieve a volume name or number when installing and security was breached allowing the installer to install yet another copy of the same software. Do you know if your product might encounter this problem?
No, volume number is only one of the parameters that Sheriff uses to generate the machine fingerprint. The algorithm that Sheriff uses to uniquely identify a machine is much more sophisticated than simply checking for the hard disk volume number.

What components do I need to distribute to my users?
If you are using the traditional DLL, **`SlsApi.dll`**.
If you are using ActiveX, **`Sheriff.dll`**
If you are using Sheriff ISR in automatic mode, **`SlsLocalComNet.dll`**

Is the name of the Sheriff licence file "Licence.SLS" fixed and unchangeable?
Yes

6.2 Serial Numbers & Product IDs

One Serial Number creates a single, unique Product ID. If I have more than one product, do I need more than one Product ID?
Generally, yes. If you had only one Product ID then if more than one product was installed on a single machine each product would have to share the same Licence Policy.

How many Licence Keys can I distribute for my product and do I have to pay a royalty?
Once you have purchased a Serial Number for your product you can issue an unlimited number of Licence Keys without any further payment.

Upon purchase of the full product is there any additional documentation, examples etc. or is everything provided with the evaluation kit?
Everything is included in the evaluation kit.

6.3 Demo & Upgrade Licences

How is the licensing of different versions of an application handled?
I have an app v1.0 that interfaces with Sheriff to provide protection. I create a new version, v1.1 that has some new features. If I simply copy the new EXE over the old, and the new EXE has the same Product ID, Secret Codes etc, its still going to work. If my "upgrade" deletes the Sheriff files (ActiveUser.SLS etc) then it won't work any more, of course. 'Feature access control' can also be used to control versions of an application (see Topic 4.2.2.3).

If we put our software on our web site, can we use Sheriff to control rights based on how long users should use the software demo after downloading it from our web site?
You can use Sheriff to control rights based on how long they can use the software after they have installed it - it is not possible to control their use of the software from the time they download it.

I need to have the capabilities to authorise a distributor to open my software *n* times only within 30 days. Is this possible?
Yes, you can combine 'date' and 'unit' metering to achieve this result.

When a user tries to install my program on another machine, the application should revert back to "demo" mode with specific function(s) turned off. Is this possible?
It depends on how the application defines "demo" mode. For example, the application could be coded so that if there is no valid licence found then it runs in demo mode automatically.

I am looking for a software only licensing solution. We are issuing multiple versions of our software and would like to let our customers purchase "limited upgrades", for example if a customer has 10 licences purchased, he can upgrade only 2 of them. The licensing program must be "smart" enough to let only 10 copies of the program run simultaneously, of which maximally two can be newer, upgraded version. Can this be done with your software?
Sheriff can be used to limit the number of concurrent users. Through the Sheriff API, you can check the number of active users at any time. If you decide that only 2 users out of 10 can run in full feature mode and the rest in trial mode, you can query Sheriff for the active users to check whether the number of active users is less than 2 and decide whether the current instance should be running in full mode or trial mode accordingly.

6.4 Compatibility

We keep getting errors on a laptop computer
Some laptops do not maintain a constant processor speed. It may vary slightly and this could cause a problem for Sheriff when it is locked to the processor speed (which is the case if the Licence Key has been generated for a Standalone installation). Also, there is a problem with some laptops using the Intel mobile Pentium III chip; this is a dual speed processor that uses "SpeedStep" technology. The problem is that when Sheriff is using a "Standalone" licence it locks to the processor speed, so the solution will be to issue a new Licence Key for the laptop but do not check the "Standalone" box. However, the key should be defined for just one concurrent user if it is to be used on a standalone machine - the fact that a key is not defined as Standalone does not prevent it from being used in a standalone configuration.

NB. With Sheriff 2.41 and above this problem should not occur with any kind of licence key.

The documentation says that in a network environment, the licence file must be in a location where all users can access it. In a Novell environment, we cannot put the licence file on the server because Novell does not recognise long file names. What options do we have?
You could put the licence file on a workstation instead. As long as the users have write access to it there shouldn't be a problem. Also, I think I'm correct in saying that you can implement long file names by following the procedure set out in Novell Technical Information Document 2929574 (search for 2929574 at www.support.novell.com).

Regarding Novell & Linux file systems. We have our application running at some clients on Novell & Linux file systems (8+3 filename). Sheriff stores the required licence files under the Product ID directory (E.g. 1234-5678-1914-9999-6666). What can we do?
Make sure your file system recognises long file names:

- For Linux you need to mount your FAT partition as vfat rather than msdos
- For Novell 3 you need to:
    1. Load os2.nam
    2. At the console prompt ADD NAME SPACE OS2 TO VOLUME
    3. Reboot workstation
    4. Place "Load OS2" in the startup.ncf file.

6.5 Licensing a Product

We have a customer who is reporting that he is getting an error message when trying to register our application as a client. I have looked at my code and traced the problem that he is seeing to a SLS_E_SYSTEM_ERROR return code from SLS_Request. He tells me that the license file looks all right, and should be fully accessable. Under what conditions can the SLS_Request function return this error code?
Sheriff issues SLS_E_SYSTEM_ERROR if

1. it fails to open one of the licence files
2. it fails to lock a section in one of the licence files

If it worked before, then it is unlikely to be the first possibility. Look in the licence folder where all licence files are kept and see if there is a file with the extension ".lok". If found, delete it and it should then work. ".lok" is a temporary file that only exists during a licence file is being locked. It should be automatically deleted by the system. The only chance that is left in the disk would be that the machine crashed while a file is being locked. The life span of a ".lok" is usually only in the scope of millisecond.

Could you elaborate more on unit metering and concurrency and the differences between them?
Unit metering allows you to rent your application, your users have to buy units from you and when they run your application the units get consumed. Different features in your application may consume different numbers of units E.g. Viewing a document may consume one unit but printing a document may consume two units. *Concurrency* controls the number of users that can run you application simultaneously. A user is defined as one user name on one PC. In other words, multiple instances of the same user on the same PC is regarded as one user i.e. A one-concurrent-user licence permits only one user to run your application at a time. The user can either be local or remote - see Network Licences Licences FAQ for questions about remote users).

Using the demonstration Product ID I cannot seem to test metered units or concurrent usage. Although I can generate the Licence Key, after installation its properties do not reflect the metered information or number of concurrent users. Furthermore, after I compile your demo application it fails because "Licence units are undefined." How can I test the licence features? What am I doing wrong with the demo application?
With the evaluation product ID, unit metering and concurrency are enabled. The reason why you get "Licence units are undefined." is due to the way that the demo application is implemented - it expects the licence to be unit metered. In other words, the Licence Key you issue for the demo application should be unit metered.

If a licence is set to expire on 01/01 will it still be usable on that date or will it expire at the end of the previous day?
The Expiry Date is inclusive, so a licence set to expire on 01/01 will still work on that date. However, Sheriff has to be able to work across time zones and uses GMT for its internal clock. Therefore the licence could expire earlier or later than midnight on 01/01, depending on the local time zone.

How common is it for the license server computer to give write access to its registry by all workstations? This doesn't seem terribly secure.
With Sheriff, workstations do not need to have access to server's registry but they do need write access to the licence file.

Is it safe to deploy a trial version of our software that automatically generates a licence with the following characteristics:

- **the licence expires after 30 days (day metering)**
- **FULL version of the product (no limitations of any kind during the trial period)**

Yes, it is safe provided that your application checks to see if such a trial licence has not ever been installed on the PC before it automatically issue a trial licence. The API function "SLS_IsProductLicensed" does just that.

Each application on our CD should have its own token (should be protected independently regardless of the other applications). Is this possible?

Yes. Each application should have its own unique Sheriff Product ID. This is used to distinguish that product from other Sheriff-protected products and permits multiple protected products to be installed on a single machine.

Is it possible for multiple applications to be activated all at once or one by one at different periods of time.
There is no problem with this as the protection for each app is independent (i.e. each has its own Product ID). Multiple users and applications are administered by the end user using the Sheriff Licence Administrator application (SlsAdmin.exe).

If we need to have a pre-determined Licence Key (like the CD-Key in most Microsoft products) with a 'Expiry Date' and 'Number of User' set, is it possible?
No, because the Licence Key is unique to each system it has to be created on the system itself. However it is possible for a Licence Key to be created on installation (i.e. self-generated) without reference to the publisher, although this is less secure.

When registering products using the Sheriff Licence Key Generator and subsequently issuing customer Licence Keys is all this information kept on a permanent basis on the computer that is using the Sheriff Licence Key Generator EXE file (SlsGen.exe)?
Yes, the information about licence keys that you, the publisher, have issued is kept in a log file and there's no problem about backing up or restoring this - or the Sheriff Licence Generator (SlsGen.exe) - to another machine.

Once supplied with the Reference Code for one of our products can we generate Licence Keys for multiple products? i.e. can we secure multiple products via one email transfer thus: user sends us one Reference Code, we generate Licence Keys for multiple products and send back, user uses the set of Licence Keys to licence the products. We ask this because of the concern over lengthy telephone conversations involved when installing multiple applications.
The answer is "No". A Licence key is bound to both the specified product and the machine. The Reference Code contains both the signature of the product and the machine. One Reference Code can be used to generate only one Licence Key, otherwise one could purchase a Licence Key and use it on many different machines.

How do I get my software from a trial licence into a networking licence? After a potential customer has evaluated the software, I want him/her to be able to buy a network, floating licence. What reference code do I need then? I want to register the application programmatically, since I do not want the hassle of distributing the Sheriff Admin tools. Can you explain what I need to do then? Do I need all reference codes from all machines potentially running our software? (which seems unworkable to me). Do I need the reference code from one PC? Or is there no need to supply a reference code?
To switch a user from a trial licence to a full licence you just need to issue a new licence key. If you don't want to distribute SlsAdmin to your users then you need to implement licensing functions in your application which involves two API calls, i.e. SLS_GetReference and SLS_SetLicence. The actual steps are something like this:

1. If the user wants to buy a new licence, he/she will select a menu item or something similar from your application. Your application will then call SLS_GetReference to generate a new reference code of the machine.
2. The user will then contact you by phone or email and tell you his/her reference together with the licence options he/she wants to purchase
3. You will then run SlsGen to generate a licence key for the user and send it back to the user
4. The user will have to enter the licence key into some sort of dialog box your application provides. Your application will then call SLS_SetLicence to authorise the licence.

As far as network licence is concerned, it is sitting on a network server. The only reference code you need to get is from the server PC. Once the network licence is authorised then it can be shared by other workstations across the network.

What can the user do between generating the Reference Code and entering the Licence Key? E.g. Exit our installation program? Reboot his machine? We ask this because of the concern that the user will have to leave his computer turned on at the Licence Key prompt until the Licence Key is obtained from us.
The user does *not* have to leave the machine turned on once the Reference Code has been generated. The Reference Code remain unchanged until a valid Licence Key has been authorised against the Reference Code on that machine. In other words, the user can install the application, get the Reference Code and email it to you and turn the machine off. When the user has got the Licence Key back from you, he/she can then turn the

machine on and the Reference Code will be the same. He/she can then enter the Licence Key and at that point the Reference Code will change.

(a) Is it possible to tell whether a Reference Code has been generated on the same machine?
The first 12 digits are the user's machine signature, the next 4 digits are the product signature and the remaining 8 digits are the run-time signature. It is possible to tell whether or not two Reference Codes are generated from the same machine by comparing the machine signatures.
(b) If the hard disk is reformatted, will it still give the same machine signature?
No. The machine signature is also bound to the Operating System.

I am implementing a Terminate procedure. I note that SlSAdmin.exe tells you to save the code. Why? Do you use it later if you want to reinstate the key? I thought License() or SetLicence() could be used for that.
The only reason that SlsAdmin tells the user to save termination code is that a software publisher, such as yourself, might want to confirm that the licence was genuinely terminated. You can verify a termination code by running SlsGen and select Tools|Verify Termination Code.

If we have the user send us a termination code as proof of termination, then do you have functionality such that I can input that code and parse it into the server code, AccesFeature, ConcUser, Etc? Or would I have to do it by hand?
The API function you need is called "SLS_VerifyTerminationCode". It is located in the Extended Library SlsLocalEx.dll and documented in the topic "Developers' Guide|Windows Library|Extended Library.

I have been able to successfully export a licence from a source machine and to import it to a target machine. However, I am never able to import the licence back to the source machine - the error I get when trying to import a licence back to the source machine is "Import licence key has expired".
In the "exporting licence" dialog box there is a field called "expiry date" which defaults to the current date. Unless you change the expiry date the licence you export will have the expiry date of the date it is created.

Is it possible for us to provide our customers with an non-expiring licence but they can enter an expiry date in the licence when they temporarily export that licence?
Yes. For example, a user can export a licence key from the 'pool' of keys on the server. The exported key will expire after the period set by the user and the key on the server will be revived at that time.

Can the publisher place limits (e.g. 3 months…) on the maximum time that an exported licence can 'live' in the exported state, while the main licence does not expire?
No. Only the user has control of an exported licence key.

Why is the Publisher Data accessible to a user? We assumed this would be useable only by the publisher to securely transfer certain useful values via the licence key codes assigned in SlsGen.exe. User access seems to thwart that...
Publishers may want the users to report the data to them. You could try encrypting the data if this is a problem. Note that the Licence Key does not pass these values to the client - Publisher Data is stored with the Licence Key but does not form part of it.

I currently trying incorporate additional information into the installation about the customer, and the Publisher Data field seems to be the idea tool to do this but whatever I place in this field within the Licence Key Generator, I always get the same value back from the API or the Sheriff Administrator...
The sole function of that Publisher Data edit box is to allow the Publisher Data to be logged together with the Licence Key. In other words, the Publisher Data entered into the License Key Generator does not become part of the Licence Key, it is only logged in the "slsgen.log". To use the Publisher Data you have to call "Sls_SetPublisherData" function, similar to the way that you would need to call SLS_SetLicence to set the Licence Key.

Let's say that I want to sell my software through a distributor. I want to give him the ability to issue a limited number of keys and then he would have to come back to me for more. Does your software handle this situation and if so, is only one license purchase required?
You will need to write your own clone of the Sheriff Licence Generator (SlsGen) application and protect it with Sheriff. You will need two Serial Numbers, one for the SlsGen clone and another for your product. To implement this you must use a Sheriff extended library, such as SlsLocalEx.dll, and call SLS_GenerateLicenceKey in order to issue licence keys with your own application. You will also need to control the number of keys that can be

generated, which means that you have to somehow meter the application (which is why you need to protect your licence generating application too, using a separate Serial Number).

We are developing a product which is version based and we need to be able to give a trial period for all versions:

1. **For the Version 1, we have a 30 day eval period. So, from the application we check the licence and, using this 30 day information, we distinguish between registered and unregistered users. Is there any way for us to provide the trial version again for Version 2?**
2. **For people who pay for Version 2, is there a way to avoid going through the registration process again. Can they enter the same licence key they registered for Version 1?**

1. A trial Licence Key is generated (issued) in the same way as a normal Licence Key. So it is up to you to generate a Licence Key that fufills the requirements of a "trial" licence
2. Normally, to generate (issue) a new Licence Key you will have to ask the user to generate a new Reference Code. However, Sheriff does offer an option by which the user can re-use the Reference Code. In the Licence Generator you'll find an option called "Reusable Reference Code". This option is only available when Expiry Date metering has been selected. It is used when you want to be able to extend the date on which the licence will expire without first contacting the user for a new Reference Code. In this way you can at any time extend the period for which your application will run simply by generating a new Licence Key for the user.

6.6 Managing Licence Keys

If a user damages or overwrites the licence file, do I have to issue them with a new licence key?
Normally you would but two of Sheriff's features enable you to get round this either by creating a Reusable Key or a Reusable Reference Code. Note that you would not create a Reusable Key if your product is time or unit metered, since this would enable your user to bypass the protection, whereas a Reusable Reference Code is only created if your product is expiry date metered.

- See topic 5.3.5 *The Licence Policy* for further info on the Reusable Key and Reusable Reference Code options.
- See topic 4.2.3.6 *SlsAPI Data Types & Structures* for further info on SLS_TYPE_REUSABLE_KEY & SLS_TYPE_REUSABLE_REF

If the user needs to move the application to another machine, will they will have to call the publisher?
No, this is not necessary. There are two options:

1. The user can *export* a licence key to another machine
2. The user can *move* the licence file to another machine

Export should be used when the original licence file will be retained on the original machine i.e. the user intends to import the licence key back at a later date or else there are a number of concurrent users who are sharing the file and one or more keys are to be exported to another machine. Move should be used when the user wishes to permanently move the licence file to another machine. Note that any exported licence keys should be reclaimed prior to a move as they cannot be reclaimed afterwards.

Why is there a distinction between the *move* and *export* functions?
The distinction exists because of restrictions placed on the export facility - for further information see the Sheriff Administrator help.

If the user defragments the hard disk, will the licence file become corrupted?
No, Sheriff permits disk deframentation without any problems.

If I have a number of concurrent users sharing a licence file, what is the position regarding export of licence keys?
Each user is represented by a separate key. So if you have ten concurrent users then up to ten licence keys could be exported. ***Caution!*** If you remove the concurrent user limit (i.e. left it blank) then an unlimited number of users can share the licence and an unlimited number of keys can be exported. You can, of course, prevent any keys from being exported by enabling the Disable Export option when defining the licence policy with the Sheriff Licence Generator.

Is the transfer of a protected application between two different PCs software oriented or is an external element (i.e. floppy disk) required?
The export of a licence key from one machine to another is entirely via software. The user generates a code on the import machine (similar to a Reference Code) and this is then validated by the exporting machine. The user uses the Sheriff Licence Administrator program to do this.

Can we keep a copy of the licence keys granted to the customer in the event that there is a problem at the customer's site - such as a corrupted or overwritten licence file - and we can get them up and running again with the absolute minimum of fuss?
Yes, you can give them a 'Reusable' key, although you would not want to issue a reusable licence key with day or unit metering. If they make a note of the key number then they can restore the licence PROVIDED that their hardware remains the same and the hard disk has not been reformatted.

Are there any limitations imposed on making backups of a hard disk or other similar system maintenance operations?
Generally, you can backup or restore licence files normally unless you are using time metering. In this case, you cannot restore a licence file if the original has been used for more than 15 minutes since it was backed up. This restriction is placed on the backup procedure in order to prevent users from by-passing time metering.

My customer says he wants to upgrade his hardware and he needs another licence for the new system - how do I know he's telling the truth?

1. You can ask him to terminate the existing licence first. Using the termination facility provided by the Sheriff Licence Administrator (SlsAdmin.exe) he informs you of the termination code; you use the Sheriff Licence Generator (SlsGen.exe) to verify whether he has actually terminated the existing licence before issuing a new licence key.
2. It is possible to tell whether a new Reference Code has been generated on the original hardware by comparing the first twelve digits of the code (the *machine signature*) with the original Reference Code. If the machine signature is the same then the hardware is unchanged.

What if they terminate the original licence and then they re-enter the original licence key? In such cases, they will have two machines running our code, without paying for one of them.
It is not possible to re-enter the original licence key twice on the same machine because the Reference Code changes as soon as a valid key has been entered. This is the case even though original licence has been terminated.

One problem we have had to deal with in the past is when we get a phone call/email saying from a customer saying that "our disk crashed, send us another key". Many times, we had no way to check it out. We had to give another key. What do you do in this case?
You will note that the User Reference Code is 24 digits. The first 12 digits are the user's machine signature, the next 4 are the product signature and the remaining 8 are the run-time signature. Thus it is possible to tell whether two Reference Codes are generated from the same machine by comparing the machine signature.

Do I have to ask the customer for a Reference Code every time I issue a new key?
Not if you use Expiry Date metering. There is an option that enables you to issue a 'Reusable Reference Code' so that when the software expiry date draws near you can issue a new key without any need to contact the customer.

If my customer wants to change his hardware, how does he do this?
Your customers can change any hardware component other than motherboard components - although they *can* upgrade RAM - or hard disk . Should they want to upgrade the motherboard or hard disk they can either move the licence temporarily to another machine or else you must issue them with a new licence key.

My customer wants to upgrade his operating system - can he do this without asking for a new licence key?
Yes, *upgrading* the o/s does not require any special precautions. If the o/s needs to be *reinstalled* from scratch then the current licence must be terminated and the customer must be issued with a new key. Alternatively, the existing key can be moved to another machine then then moved back again when the new o/s has been installed.

When my customer moves a licence to a new machine, can he then restore the original licence from a backup?
No. The original licence cannot be restored.

When my customer moves a licence to a new machine, can licences that were exported prior to the move be imported to the new machine?
No. They cannot be imported to the new machine, therefore it is important to reclaim any exported licence prior to the move.

When my customer moves a licence to a new machine, will the licence have exactly the same policy as the original licence from the publisher?
No. It will be as if the publisher had issued a new licence but with the policy as it was when the licence was moved i.e. exported keys that have not been reclaimed will be taken into account (although they cannot be reclaimed after the move).

Can an exported licence be moved to another machine?
No. An exported licence cannot be moved to another machine. An exported licence can only be imported back to the original machine.

6.7 Network Licences

For questions about Novell networks, please see FAQ Compatibility.

So how can I restrict execution of my program from a remote PC? Let's say that I've installed the software at PC "A" and another user on PC "B" sees it on the network, how can I stop him from using it?
Sheriff implements machine binding to protect your software from unaurthorised use. When you issue the licence key (using SlsGen) you have the opportunity to decide how many concurrent network users can use the software application simultaneously. So, if your software is installed on the server you can specify that, for example, only two network users can use it at the same time. This is a "floating" licence. However, if you want to restrict use to just the machine on which the application is installed you can issue a "Standalone" licence. This means that other machines on the network will not be able to use the application at all. So you have complete control over how your application is used.

Is there the possibility of defining the maximum number of concurrent users on a LAN?
You define the maximum number of concurrent users with the Sheriff Licence Generator application (SlsGen.exe) when you issue a licence key. You can monitor the number of active concurrent users with the Sheriff Administrator application (SlsAdmin.exe). To do this SlsAdmin should be run on the server PC where the licence is installed.

Should I use Sheriff Administrator on each end-user PC to configure the licence path or can we register the licence path by program (i.e. during the installation)?
With Sheriff Version 3 it is no longer necessary register the path to the Sheriff licence file, since this has been replaced by the SlsServer application.

Is it possible to sell an application with two licences, for example, and without changing the exe file, add more licences?
You can add more licences by issuing a new Licence Key to your customer i.e. they will have to give you a new Reference Code and you will give them a new Licence Key that is valid for the new number of users. You do not have to ship your users a new exe file.

Is the only way to add new network licences to a licence database by terminating the current licence and issuing a new licence key?
Yes. To alter or extend the licence policy a new licence key has to be issued.

If I want to control concurrent invocations of an application by a single user, what would you suggest? Consider the situation of someone licensing a multi-cpu server and running hundreds of batch jobs as the same user.
Multiple instances of an application on a PC by the same user are counted as just one concurent user. Note that different users on the same PC are counted as multiple users and the same user on two or more different PCs is also counted as multiple users.

You say that any network operating system is supported. Could you clarify?
With Sheriff 3 the N/OS is limited to Windows.

Does Sheriff create much additional network traffic if there are many network users (E.g. 100 concurrent users)?
No. The extra traffic created by Sheriff is minimal. It is only the heat-beating message, which is a very small (16 bytes) packet in size. The frequency of the heartbeat is determined by the Update Licence function.

Does it matter if the clocks on the machines on the network are out of synch?
With previous versions of Sheriff this was of importance; with Sheriff 3 this is no longer the case, so the answer is 'no'.

How is the user's name encoded into the licence?
User's name is not encoded in licence key - the machine signature ('digital fingerprint') is encoded.

I am wondering how counting works. I presume that it counts concurrent users regardless of the

identities of these users?
Sheriff counts concurrent users regardless of their identity. Normally your Sheriff licence database will be located on a server to which all prospective users will have write access. This is a typical network configuration where the licence file is locked to a server to which network users have access. It is often referred to as a "floating" licence configuration because as long as the maximum number of users is not exceeded, any user who has access to the licence file on the server can run the application. When a user terminates the application then the number of available licence is incremented by one.

With floating licences, you specify the maximum number of concurrent users (i.e. the maximum number of users who can simultaneously run your application). An alternative configuration is called "standalone", where the protected application can only run on the machine on which the licence file is installed.

What I cannot understand is that the key seems to be machine-specific. Does that mean that if we sell a 50-user license each potential user (i.e. in general many more than 50) has to call us to get a key? Also, if another user happened to buy a single-use license and runs it on the same network, would he interfere with the 50-user license count?
In a situation where you have 50 network users and you want to add another ten users, you will have to issue a new licence key to overwrite the old.

If you licensed a single additional user - Fred - on the network you could either issue a new key for the server (as explained above) or you could issue a standalone key for Fred's workstation. Fred would install your software on his own workstation and the Sheriff licence database would reside on that machine (the fact that the machine happened to be connected to a network would be irrelevant). In this example the other network users would not be able to share Fred's licence (so the answer is "No", Fred's licence would not interfere with the 50 licence count).

I can see only one user being connected (via a Terminal Server Client) even though there are two clients running the application.
That is because with TS all of the instances of your application are actually running on the same server - as far as Sheriff is concerned those instances are all running on the same hardware and if the user names are identical then those instances are regarded as one concurrent user running multiple instances of the same application on the same hardware. You will notice that the user count reads two while there is only one user shown.

6.8 The Sheriff API

Can I use Sheriff to protect my application without resorting to C++?
Yes, SlsApi.dll is a standard Windows DLL that can be called by any Windows programming language including Delphi, Visual Basic and VBA. In fact, Sheriff SDK provides ready-to-use templates/classes for Delphi and Visual Basic.

What library and header files will I need to include in my project?
It depends on your programming language. In Sheriff SDK there are three directories under the API directory, namely VC, VB, Delphi & VFP. In general you have to include all of the files from the relevant directory in your project. To distribute your software, you only need to include SlsApi.dll.

What is the best location for the Sheriff DLL (SlsApi.dll)?
It does not matter, usually the DLL is located in the same folder as the application.

What name for the folder for the licence file?
It should be the same name as the Product ID number.

I understand that it is possible to bypass the need for the user to call in for a licence number, using the `SLS_License` function?
`SLS_License` is designed for issuing trial/demo licence only. Although it does not stop you from using to issue a full licence, we do recommend that user has to contact you for a proper Licence Key. However, your contact can be made by phone, fax, email or even your web-site (with Sheriff 1.61 and above you can set up a web page that can automatically issue licence keys, though this requires an NT server).

Can there be "soft" and "hard" landings, meaning the user gets a warning when they are approaching their limits of logons?
Yes, how your application behaves when the limit is approaching is decided by your application. By using the API function `SLS_GetLicenceInfo` you can retrieve the states of the licence, including the usage, at any time.

I've tried to install my protected software and then to change the system clock to extend the time. The Sheriff API detected the time change but now, each call to `SLS_GetReference` generates an error message : "Invalid System Time" and the software is then locked for eternity. What's happening?
As soon as Sheriff detects any system or licence parameter being tampered with it locks up the licence database to prevent further usage. The way to unlock your licence database is to either change your system clock back to the correct time or delete the licence database files and start it again. Please note that Sheriff allows up to 24 hours time difference between workstations.

I have implemented time metering in my application. What happens if the user tries to cirumvent this by turning back the system clock?
If day/date metering has been implemented and the user puts the clock back more than 24 hours then the error code SLS_E_SYSTEM_TIME will be generated. What action your program takes if this happens is up to you. When the user puts the time forward, the system will work again. However, should the user put the clock forward and then put it back, Sheriff will also consider this to be an attempt to circumvent the metering.

Monitoring the registry in Windows NT shows that upon registering an application registry entries are created pointing to the location of the licence files. Thus if the registry entries and licence files are deleted it appears to be possible to reset any demo version of software and begin the demo period again. Is there any mechanism which will detect this occurrence and provide protection against this action?
Function `SLS_IsProductInstalled` is provided for this purpose. Before calling `SLS_License` to issue a demo licence, call `SLS_IsProductInstalled` to check whether the product has been previously installed. Sheriff Ver 1.1 and above enhance this feature to provide additional protection against re-installing a demo licence.

I have a question about the IsProductInstalled function: Under what conditions does it return false? I want to use it as a test before I use the License function to create a 30-day trial. I have found that it returns true even when I run the demo application on a machine that has never had Sheriff installed on

it. Is this a limitation of the demo product key provided?
IsProductInstalled returns as true soon as a Sheriff-protected product has been installed on a PC. To clear it - i.e. to make it return false again - you have to "remove" the product. This can be achieved by running "SlsAdmin" and selecting the "licence|remove" function. Please note that you'll also need to run "SlsGen" to generate a "remove password" in order to remove a licence.

You have indicated that the API function `SLS_IsProductInstalled` prevents users from installing multiple times on the same PC (or, I assume, same network). Is there, however, a means of preventing multiple installations to other PC's? Is that what is meant when it generates a machine "fingerprint"?
When a Sheriff-protected product is installed on a particular machine it is bound to that machine, which can identified by my means of the machine 'fingerprint' contained in the unique Reference Code that is displayed at the time of installation. Normally, a user will quote their Reference Code to you, the publisher, and you will issue a Licence Key to the user enabling him to run the application according to the terms of the Licence Policy (unlimited use, unit metered etc.). However, because the application is bound to the machine that it is installed on, it cannot be copied from that machine to other machines. It can, however, be installed to other machines provided the correct installation procedure is undertaken (i.e. the publisher issues a unique Licence Key for each of those other machines). So it's up to you to decide how many machines the application is installed on. `SLS_IsProductInstalled` function is typically used in conjunction with `SLS_License`. `SLS_License` will be used only when you wish to issue a trial licence with some limitations - such as limited features or limited life span.

I need to implement a licence remove facility to verify that the user terminated the licence.
SLS_Terminate combined with SLS_VerifyTerminationCode (from the Extended API) should do the trick.

Suppose I have a customer who has already bought a 10 users licence. Now, the customer wants to upgrade the licence to a 20 user licence. Beside using SlsAdmin to terminate the licence, can I terminate the licence inside my program by calling `SLS_TERMINATE` ? What's the effect of calling `SLS_TERMINATE` in my program? Do I have to stop the timer object (for heartbeating)? Do I need to run `SLS_RELEASE` after I run `SLS_TERMINATE`? (I assume I do not need to because the licence is already terminated.)
Yes, you can call the SLS_Terminate to terminate the licence. After the licence is terminated, the licence files will be completely removed. The correct procedure to terminate a licence should be:

- `Call SLS_Release` (this is not essential but it is always a good practice to close files before deleting them).
- `StopHeartbeat`
- `CallSLS_Terminate`

However, you don't have to terminate an existing licence in order to upgrade it. You can always overwrite it by issuing a new licence. Again, you should release the licence and stop the heartbeat before renewing it.

`OnChallenge` event: we are not sure when and why the event is issued. We understand it is making sure the data passed between our application and Sheriff has not been tampered with but how do we use it?
As explained in the "ActiveX | Automatic Mode Quick Start | Step 6", OnChallenge event is only fired if you put Sheriff in automatic mode. The purpose of OnChallenge event is to challenge the host application in order to verify whether or not the host application is the software publisher's original. OnChallenge makes sure that only the software publisher can issue a trial licence automatically.

Having looked at the demo source in VB, I don't know why the secrets array has 72 codes, and also where do these come from. The `SetSecrets` method in the Sheriff class takes this secrets array and then pads out another 56 codes with zero.
The VB demo uses the encrypted Secret Codes, each code is 18 bytes in length and there are four Secret Codes hence 72 bytes in total. However, by design the maximum length allocated for each Secret Code is 32 bytes in order to leave spaces for future expansion. That is why we need to pad 14 bytes to the end of each code.

When I use the `GetReference` function after the a successful `SetLicence`, the value has changed. Am I doing something wrong or is this normal?
Nothing is wrong. This is a feature that prevents a Reference Code from being re-used.

What is the justification for `OnProductNotLicensed` event and what does it mean when is it triggered?

`OnProductNotLicensed` event is triggered when Sheriff detects that the product is installed but it has not been authorised with a valid licence i.e. there is no valid licence detected on the PC for the product.

What's the difference between `SLS_TYPE_REUSABLE_KEY` & `SLS_TYPE_REUSABLE_REF`?

`SLS_TYPE_REUSABLE_KEY` produces a reusable *Licence Key*. A reusable Licence Key can be reused on the same machine any number of times as long as the Licence Policy permits this. For example, if the Licence File is deleted by accident the user can simply re-license the machine with the reusable Licence Key without the need to contact the publisher.

`SLS_TYPE_REUSABLE_REF` makes the *Reference Code* reusable, which enables the software publisher to save the Reference Code for renewal purposes. If the Reference Code is not reusable, when a user's licence is due for renewal the publisher has to first ask the user for the current Reference Code before issuing a renewal licence. With a reusable Reference Code, the publisher can simply reuse the previous Reference Code to issue a Licence Key without first needing to contact the user.

NB. You would not create a Reusable Key if your product is time/unit metered, since this would enable your user to bypass the protection.

What is the difference between the functions Query Licence Info (`SLS_QueryLicenceInfo`) and Get Licence Status (`SLS_GetStatusCode`) and when would you use them? For example, if you wanted to display the current status of the licence to the user (E.g. 10 days left to run), which would you use?

The difference is that QueryLicenceInfo returns the licence properties in non-encrypted format where as GetLicenceStatus returns them encrypted. If you want to display the current status of the licence to the user you need to call QueryLicenceInfo.

I have downloaded the evaluation version and I am having trouble using the unit meter feature. I call `SLS_Update` with an `SLS_UPDATE` record with UnitsConsumed set. When I later call `SLS_QueryLicenceInfo`, the units consumed again equals 0. Any ideas or sample source code? I am very interested in buying such a product.

Units are deducted from the licence only at the time when `SLS_Release` is called. `SLS_Update` causes units to be deducted from licence's pool instead of the licence's meter.

6.9 ActiveX Control

After I make the call to Sherif1.CheckLicence, with the autotrial mode set to false, and the user clicks the "Cancel" button on the authorization screen the program is activated anyway. I notice this is the case on your demo program as well. How do I trap on that condition and not allow the program to become active?
You have to trap events such as OnProductNotLicensed etc. fired up by Sheriff and implement the appropriate "landing" policy. In the case of our demo application, it implements a "soft landing" policy. When the user decides not to give a valid licence key by clicking on the "Cancel" button, the demo lets the user continue to use the application but in a kind of "demo" mode. In other words, the application needs to check if the "cancel" button is pressed and if so to take the desired action.

I am developing a VB application using the Automatic mode ActiveX control - do I have to call RegisterLicence? Where is licence information stored if RegisterLicence is not called?
You don't have to call RegisterLicence. Sheriff by default creates its licence in the location where the Sheriff.dll is installed. However, if you wish to specify your own directory then you will need to call RegisterLicence.

The user requires a licence key in order for the application to work at all. Therefore, do I only need Auto Authorize checked for the Automatic mode to work?
Yes, Auto Authorize will be fine. You might also want to set Auto Renew.

Assuming that I only have Auto Authorize checked, which OnXXXXX events do I need to handle? I think I only need to handle OnError, OnProductNotAuthorized and OnProductNotLicensed. Please confirm.
You have to handle all of the events in order to properly protect your application. Those events are documented in Automatic Mode Event Handlers.

How do I implement unit metering using the ActiveX control?
To implement unit metering in your application with Sheriff ActiveX control you'll have to use it in the Advanced Mode, the Automatic Mode does not support unit metering. There are the following methods involved:

RequestLicenceUnits(UnitsReserved As long, byref UnitsGranted as long,
byref AccessKey as Long) As long

To request a valid licence and reserve a number of units to run the application. AccessKey returns the feature access level defined by the licence.

UpdateLicenceUnits(UnitsReserved As long, UnitsConsumed As long,
byref UnitsGranted As long) As long

To update the licence on heartbeat with the number of units consumed. In addition, to modify the number of units the application wants to reserve.

ReleaseLicenceUnits(UnitsConsumed As long) As long
To release licence when the number units has been consumed in the session.

If there is no units left, the error code "SLS_E_LICENCE_EXHAUSTED" will be returned by RequestLicenceUnits.

6.10 Sheriff ISR

What files do I need to distribute in order to implement ISR in automatic mode?
You must distribute SlsLocalComNet.dll, which must be registered on the end user's machine, preferably as part of the program installation procedure.

How do I manually register the SheriffNet ActiveX Control on a user's machine?
Run the command `REGSVR32 SlsLocalComNet.dll` from the DOS prompt after making sure that REGSVR32.EXE is located in the system path.

6.11 Sheriff 3

What's the difference between Sheriff 2 and Sheriff 3?
Sheriff 3 utilises client/server architecture, which differs from Sheriff 2 where a shared file was located on the server. In other words, with Sheriff 3 there are separate Sheriff client/server applications whereas in Sheriff 2 the Sheriff applications only ran on the client. Please read Sheriff Help topic 3.4 for further details.

Is Sheriff 3 compatible with all versions of Windows since Windows 95 or are there some restrictions on either the client or the server o/s?
Both the client and server should work with all versions since Windows 95, however we only support Windows 2000, XP and Vista. The service only works with Windows 2000, XP and Vista.

Do the same evaluation Product IDs and Secret Codes still work
Yes.

Is the registry path still "HKEY_LOCAL_MACHINE\SOFTWARE\ACUDATA\SHERIFF\PRODUCTID"?
Yes

Will my upgrade be as simple as using your new header files and replacing "SlsApi.dll" with "SlsLocal.dll" ?
Yes, that is correct.

In other words, I don't have to distribute and deal with your server exe's?
Yes, that is correct.

And, of course, my new 3.0 application will recognise existing 2.8.7 licenses on customer machines?
Yes.

Where is the slsapi.lib file located?
There are now two versions of header files named slsLocal.lib and slsRemote.lib located in \API\DLL\Local and \API\DLL\Remote

The inc and lib directories have local and remote subdirectories. What is the difference between local application and a remote application?
A local application is what used to be called "standalone application" - an application running on a local machine. A remote application is also called a "network application" which talks to a Sheriff license server running on a remote server.

Is the old method of issuing standalone licenses still supported? That is, can I still access local licenses as files or does all traffic have to be directed through the SlsSevice?
Yes, the old method of issuing standalone licences are still supported in the same way as it use to be and with the same API functions. All you need to do is to link your app to SlsLocal.dll

All of our products have a network license option. This would seem to indicate, then, that I would need to know in advance whether I need SlsLocal.dll or SlsRemote.dll. In other words, I'd need a registry key or something similar to tell my app whether to load the local or remote flavor, and go from there. The only other option would be to install the SlsService on all the machines and direct the SLS_Connect command at localhost. Is this correct?
Yes, that is correct. We assume that your software knows the type of the licence, network or standalone. You can indeed treat a standalone licence as a network licence by running SlsServer on a local machine and direct SLS_Connect to the localhost.

We have several cases where users have a mix of network and standalone licenses. The main application license may be network, but specific libraries may be standalone. Is it even possible to support this case? Can I use both SlsLocal and SlsRemote at the same time? Is the SlsRemote DLL capable of talking to multiple servers?
It is possible to support both standalone and network licensing at the same time. One way to do that is to treat the standalone licence as a "local network licence". By doing so, you only need to use SlsRemote. The other way would be to dynamically load SlsLocal.dll or SlsRemote.dll at the run time depending on the type of the licence:

you load up SlsLocal if the licence is local, SlsRemote if the licence is network.

When am I supposed to call SLS_Connect? The documentation does not say anything about this.
SLS_Connect is called to connect the client to the Sheriff server, i.e. you only need to call it for a networking licence. SLS_Connect should be the first call before any other API can be called.

I don't seem to be able to call SLS_Remove when I am not connected. I call this to clear out the registry entries when something goes wrong sometimes, and if something has gone wrong, I will not be connected.
For network version (SlsRemote.dll), SLS_Remove removes registry and/or files on the server. If that is what you intend to do, then you have to run it locally on the server machine in cases where the connection is broken.

If the server is set to a different port, do I have to set the Clients to that same port when I connect? This will be an additional piece of information needed at setup time.
Yes. Like any client/server application, the client has to know where the server is.

If the licence files are written to by the server application, do the clients still need to know the path to the licence files? If so, then presumably the SLS_Registry routine will still need a path string? It would be nice if we didn't have to specify this, as it is sometimes a source of confusion for our customers.
No, the client does not need know where the licence files are. In the network environment, functions like SLS_Register, SLS_Remove are called to administer the server not the client. This may be a bit confusing for people who are used to Sheriff 2 where everything is local - even the so-called network licence file in Sheriff 2 is kind of local to the client. Please remember, with Sheriff 3.0 and a network licence, everything is remote on the server. All API functions actually work on the server.

So the clients no longer need write access to the licence file?
That is right. There is no need to have the shared folder either. The client does not need to have any direct access to files on the server. It is a true client/server framework.

7.1 Sheriff User Agreement

You should read carefully the following terms and conditions of the User Agreement in its entirety before either downloading, installing, integrating, evaluating or using the Sheriff software and/or components. Do not download, install, integrate, evaluate or use the software unless you agree to be bound by the definitions, terms and conditions of this Agreement.

An Agreement between Licensing Technologies Limited of 22 Wycombe End, Beaconsfield, Buckinghamshire HP9 1NB, England (hereinafter referred to as "LTL") and the software user (hereinafter referred to as "the user").

Whereas LTL is a company based in the United Kingdom in the business of software development and related services and has designed and developed a suite of software applications, known as collectively as "Sheriff", used by software publishers and developers to restrict the unauthorised copying of software applications and the user wishes to make use of Sheriff for this purpose

Now it is hereby agreed as follows:

Definitions

The term "Software" shall refer to the files supplied via electronic download or on diskette(s) or other media and/or to any and all copies, upgrades, modifications, versions, editions, functionally-equivalent derivatives, or any parts or portions thereof, including any electronic, on-line or printed documentation. The term "Software" includes the Sheriff Serial Number, Product ID and Secret Codes.

Unless stated otherwise the term "Sheriff" shall include all components of the suite of software programs known as the Sheriff Software Development Kit (The "Sheriff SDK") including, but not limited to, all files, code, libraries, executables and documentation.

"Install" means the action of loading the software onto a permanent storage device (such as a hard drive).

"Use" means to run the software by loading into such computer's temporary memory.

Grant of Licence

LTL hereby grants the user a non-transferable, nonexclusive license to use the software on the terms and conditions contained herein.

Term of Licence

This Licence is effective until terminated, either expressly or by implication, by the either LTL or the user.

Ownership of Software

LTL is the sole owner of the software and all rights not specifically granted in this licence are reserved by LTL.

The Users' Obligations

The user warrants that all information supplied to LTL concerning the identity, address and other details of the user shall be true and that any additional information that can be reasonably considered relevant to the terms of this Agreement shall be disclosed by the user to LTL, whether or not LTL has specifically requested such information.

Under this licence the user agrees to:

- Except as otherwise provided for in this Agreement, keep all copies of the software solely in his/her possession.
- Use reasonable efforts to protect the software from unauthorised use, reproduction, publication or distribution.

- Use the software only for the purpose set forth in this Agreement and for no other purpose.
- Without charge transfer to LTL all property in any comments, reports of difficulties or problems with the software and all suggestions for improvement.

Under this licence the user may not:

- Distribute the software, except as provided for under the terms of this Agreement.
- Sublicense, rent, lease, lend or otherwise transfer the software or the user's rights under this licence without the prior written consent of LTL.
- Inform, directly or indirectly, any third party of the the user's unique Serial Number, Product ID or Secret Codes.
- Remove or obscure any copyright and trademark notices.

Software Installation

The software may be installed and used on any computer that is owned or normally used by:

- the user
- the user's customer(s)
- an employee of the user or the user's customer(s)

PROVIDED THAT where the user integrates Sheriff with a product (application, database etc.) belonging to the user's customer(s):

- the user shall give notice to the customer(s) of the terms of this Agreement and the customer(s) shall likewise be bound by its terms.
- the user shall purchase a separate (full-price) Serial Number for each of the user's customers.
- where a customer of the user has more than one product, the user shall purchase an additional (discounted) Serial Number for each additional product.

Sheriff Licence Administrator Application

Notwithstanding the terms of this Agreement, the Sheriff Licence Administrator application (SlsAdmin.exe) and its associated documentation may be freely distributed without any restriction whatsoever.

Applications Not Developed by LTL

LTL makes no proprietary claims of any kind, whether in copyright or otherwise, to any software applications developed by the user for either the user or any customer of the user.

Contractual Relations with the User's Customers

Any contractual arrangement that exists between the user and his/her customers shall not extend to LTL and the user hereby agrees to indemnify and hold LTL harmless against any claim, expense, loss, damage, or other liability, including (but without limitation to) legal fees, arising out of or connected with the customer's use of the Sheriff Licence Administrator, the Sheriff Licence Generator and/or their associated components.

Support of Software

For a period of 30 days following the date of this Agreement LTL shall provide the following support:

- Unlimited support by electronic mail (email)
- Free software upgrades
- Free bug fixes

After the period of 30 days has elapsed LTL shall have no further support obligations of any kind under this Agreement and such support services as LTL supplies to the user shall be under the terms of the ancillary Sheriff Software Support & Maintenance Agreement, the terms of which shall be supplemental to the terms of this Agreement.

Any supplemental software code and/or documentation provided as part of the Support Services shall be considered part of the Software and subject to the terms and conditions of this Agreement. With respect to technical information provided to LTL as part of the Support Services, LTL may use such information for its business purposes, including product support and development.

Return of Software upon Termination.

At the end of the term of this licence or upon termination by either party the user shall either destroy or immediately return all copies of the software to LTL unless otherwise agreed.

Upgrades & Bug Fixes

After a period of 30 days has elapsed LTL is under no obligation to provide the user with upgrades to the software or to inform the user when upgrades are available otherwise than under the terms of the Sheriff Software Support and Maintenance Agreement.

The user accepts that the circumstances under which software products are deployed can vary widely and that it is impossible for LTL to envisage all of these circumstances. Situations may arise in the field in which defects in the software ('bugs') are uncovered. The user accepts that the only remedy is to inform LTL of the nature of the problem and the circumstances under which it arose. LTL shall then take all reasonable steps to remedy the problem within a reasonable period of time. The user accepts that in some cases it may not be possible to remedy the defect due to the unique nature of the circumstances under which the software has been deployed.

Disclaimer of Warranties and Limitation of Remedies

"Unregistered software" is software that is supplied to the user free of charge (i.e. without monetary payment by the user). The user acknowledges that unregistered software is being made available to the user on an AS IS BASIS FOR EVALUATION PURPOSES ONLY, and that LTL makes no warranties, express or implied, with respect to the software.

Purchase of a Serial Number registers the software and indicates that the user is satisfied that the software is suitable for his/her requirements. LTL cannot provide refunds after registration. Orders for Serial Numbers are filled immediately - once placed, they cannot be cancelled.

LTL specifically disclaims all implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. It is for the user to evaluate and test the software and to determine its capabilities.

In no event will LTL be liable for special, incidental or consequential damages arising from the use, or inability to use, the software, even if LTL has been advised of the possibility of such damages. In particular, LTL shall not be liable for the loss of information or data arising from the use or inability to use the software either by the user or the user's customers.

In all cases LTL's maximum liability for any claim arising from the licence or use of the registered software, including but without limitation to claims based upon LTL's negligence, shall be limited to the replacement value of a single copy of the software. The remedy set forth above is exclusive and in lieu of all others, oral or written, expressed or implied.

No developer, distributor, employee, or agent is authorised to extend any warranty or modify or extend the above remedy in any manner.

Indemnification

The user acknowledges that its use of the software is personal to the user and is not intended to be used for the benefit of any third party (other than the user's customers). The user shall indemnify and hold LTL harmless against any claim, expense, loss, damage, or other liability, including (but without limitation) legal fees, arising out of or connected with the user's use of the software under this Agreement.

Remedies for Breach

In the event that the user breaches any warranty, covenant or other term of this Agreement, whether express or implied, LTL shall be entitled to immediately terminate this Agreement and the licence hereunder and to seek all

legal and equitable remedies available to it.

The user acknowledges that breach of any of the terms of this Agreement may result in irreparable and continuing damage to LTL in an amount not readily ascertainable and for which there would be no adequate remedy at law. In the event of such a breach, LTL shall have the right to seek immediate injunctive relief, in addition to its other remedies hereunder.

Non-Waiver

The failure or delay of any party to require performance of, or to otherwise enforce, any condition or other provision of this Agreement shall not waive or otherwise limit that party's right to enforce or pursue remedies for the breach of any such provision or condition. Any waiver by any party of any particular condition or provision of this Agreement, including this non-waiver provision, shall not constitute a waiver or limitation on that party's right to enforce performance or pursue remedies for the breach of any other condition or provision of this Agreement.

Severability

If any court of competent jurisdiction finds any term of this Agreement, or of any other document or instrument referred to or contemplated in this Agreement, to be invalid or unenforceable, such determination shall not affect the validity and enforceability of the remainder of the Agreement.

Paragraph Headings

All paragraph headings in this Agreement appear for convenience of reference, and shall not affect the meaning or interpretation of the Agreement.

Assignment

This Agreement may not be assigned by the user without the prior written consent of LTL. Subject to this restriction, this Agreement is binding upon and shall inure to the benefit of the heirs, successors, assigns, legatees, devisees, bankrupt estates, administrators, personal representatives and executors of each of the parties.

Governing Law

This Agreement shall be governed by and construed in accordance with the Laws of the England & Wales.

Amendments

This Agreement may be amended or modified only by a written instrument executed by the parties that expressly states the intent of the parties to modify or amend this Agreement.

Entire Agreement

This Agreement constitutes the entire agreement between the parties pertaining to the subject matter of the Agreement and supersedes all prior discussions, negotiations, understandings, representations and agreements, whether oral or written. All terms of this Agreement are contractual and not mere recitals.

Electronic Execution

Execution of this Agreement may be evidenced by downloading, installing or using the software.

7.2 Sheriff Software Support & Maintenance Agreement

An Agreement between Licensing Technologies Limited of 22 Wycombe End, Beaconsfield, Buckinghamshire HP9 1NB, England (hereinafter referred to as "LTL") and the software user (hereinafter referred to as "the user").

Whereas LTL is a company based in the United Kingdom in the business of software development and related services and has designed and developed a suite of software applications, known as collectively as "Sheriff", and the user has entered into a licensing agreement ('Sheriff User Agreement') with LTL for the use of Sheriff

Now it is hereby agreed as follows:

In addition to any pre-existing obligations under the Sheriff User Agreement LTL shall, for a period of one year from the commencement of this Support & Maintenance Agreement, supply to the user the following services:

- Provide access to all maintenance and other upgrades - including major version upgrades - without charge and shall inform the customer as soon as such versions and upgrades become available.
- Provide the customer with guaranteed 24 hour response by email to communications from the customer.
- Use all reasonable endeavours to replace any defective materials and resolve customer reported software defects and report promptly and regularly to the customer on all steps being taken to rectify such faults.
